AIAA Paper No. 2010-3505,
AIAA InfoTech@Aerospace Conference, Atlanta, GA, April 20-22, 2010.

A New Software Engineering Course for Undergraduate and Graduate Students

Lyle N. Long* and Oranuj Janrathitikarn+
The Pennsylvania State University, University Park, PA 16802

This paper describes a new introductory level course in software engineering for aerospace engineering students. It offers the fundamental concepts of software engineering to senior-level and graduate aerospace engineering students through lectures and a team project. The material in the lectures support the timeline of the team project. The class project differs from projects of other software engineering courses because each student works in a small group representing one process in a software development model, and each group is part of a large team. The instructor acted as a customer who indicated system needs. The project goal was to develop a small software system for a teleoperated mobile robot. The team project reinforced the concepts from the class and tried to demonstrate how software engineering works in a simulated industrial environment. The feedback from students in the class was evaluated through a survey and a focus group discussion (presented in a separate paper). Courses such as these are essential since software and computing can be 50% of the cost of modern aircraft and spacecraft. In order to make aerospace engineering degrees as useful as possible, modern curriculum must include material beyond the traditional aerodynamics, structures, propulsion, and dynamics/control areas.

I. Introduction

Aerospace engineering has historically been based upon four traditional disciplines: (i) fluid dynamics, aerodynamics, and thermodynamics, (ii) guidance, control, and dynamics, (iii) propulsion and power, and (iv) structural mechanics and materials.[1] Even the Wright brothers had to consider all these topics. These four areas are fundamental aerospace disciplines, but they are fairly mature. Modern aerospace engineering systems are heavily dependant on software which has been called the Achilles Heel of aerospace systems.[2] Many commercial aircraft, military aircraft, helicopters, spacecraft, and even ground systems are either partially or fully controlled by computers. Consequently, in the development of large and complex systems, the four traditional disciplines of aerospace engineering are not enough to fully understand these systems. Modern aerospace engineering must also include computing, information, and communication[1] as the fifth pillar or discipline of aerospace engineering. Aircraft and spacecraft can be classified as cyberphysical systems. The traditional four disciplines may have been appropriate for pre-Sputnik systems, but are incomplete to describe modern systems. The cyberphysical aspects of aircraft and spacecraft are less mature, more complex, and often more important than the other four areas. This is probably one of the reasons why aerospace companies do not hire more aerospace engineers[2] than they do.

Software engineering is considered as a part of the fifth discipline. It plays an important role in developing the safe and reliable software systems. Aircraft and spacecraft are examples of safety- or mission-critical systems, which must have reliable software. Errors in software systems could lead to devastating accidents and could result in lose of lives or failed missions. In order to prepare students to be ready for the modern aerospace workforce, software engineering should be included in the undergraduate curriculum (as well as computing, information, and communication). The Department of Aerospace Engineering at the Pennsylvania State University now offers a course called "Introduction to Software Engineering for Aerospace Engineers" (AERSP 440) (http://www.personal.psu.edu/lnl/440pub/). This course is a senior-level course for aerospace engineering students. The students are required to take this course or a course in Electronic circuits (they should really be *required* to take both!). It is also open to students from other majors and graduate students because it can be used towards the requirements of an Information Science and Technology (IST) undergraduate minor or a Computational Science

* Distinguished Professor, Fellow AIAA
+ Graduate Student, Student Member AIAA

graduate minor. Most students in the class are aerospace engineering undergraduate students. The class size has doubled from approximately 25 students in 2007 to 50 students in 2009 (and 45 in 2010).

Since we are unable to add additional course requirements to the Aerospace Engineering degree, we have found that offering Minors is an effective way of acknowledging the student's efforts in learning additional material. Our undergraduate minor in Information Science and Technology for Aerospace Engineers (http://www.personal.psu.edu/lnl/ist/) allows the students to take additional courses in very important areas. This Minor was created in 2007 and requires the students to take the following courses:

- Information, People, and Technology (IST 110, 3 credits)
- Computer Programming for Engineers using C++ (CMPSC 201C, 3 credits)
- Organization of Data (IST 210, 4 credits)
- Networking and Telecommunications (IST 220, 3 credits)

Plus two of these 3-credit courses:

- Advanced Computer Programming (AERSP 424)
- Introduction to Software Engineering for Aerospace Engineers (AERSP 440)
- Introduction to Numerical Methods in Fluid Dynamics (AERSP 423)
- Aerospace Control Systems (AERSP 460)

for a total of 19 credits. Some of these courses can also be used as technical electives for the Aerospace Engineering B.S. degree. An executive from a major aerospace company said this about the program:

> "It looks like a great curriculum...really good mix of courses, touches all the key facets of the technology. This minor should be a real plus for students entering the job market."

And the National Academy of Engineering (NAE) states [3]:

> "Given the expected role of computers in the future, it is essential that engineers of all disciplines have a deep working knowledge of the fundamentals of digital systems as well as fluency in using contemporary computer systems and tools."

This Minor allows the student to strengthen their skills in key areas and is an approach that could easily be emulated at other universities.

We also have a Minor for graduate students called "Computational Science" (http://www.csci.psu.edu/). This is available to all graduate students at Penn State, and has been in place since 1999 (originally it was called High Performance Computing Minor). There have been more than 200 graduate students who have either graduated or are currently enrolled in the program. Students pursuing a Master's degree or a Ph.D. can choose to pursue this Minor. They must take two of these courses:

- Advanced Computer Programming (AERSP 424)
- Parallel Computing (CSE 557 or NucE 530)
- Numerical Analysis I (Math 523)
- Numerical Linear Algebra (Math 550)
- Applied Statistics (Stat 500)
- Data Mining I (Stat/IST 557)

They also need to take 1 additional course for the Master's degree Minor or 3 additional courses for the Ph.D. Minor. These additional courses can be chosen from a long list of computational science courses:

http://www.csci.psu.edu/minor.html#courses

II. Software Engineering Course Content

When the software engineering course was first offered, the instruction method was based solely on in-class lectures by the instructor and some invited guest lectures from outside of academia. The student knowledge was assessed using homework and exams. In order to make the class more interesting and valuable, a team project was included in the course in spring 2009 and 2010. The challenge of designing a team project for this course is that the students have limited background in programming (*unfortunately*!) and software engineering. Therefore the project must not be too complex and has to focus upon the software engineering concepts and processes. Ideally they should be required to also take an advanced programming course (such as our AERSP 424).

The objective of the course is to introduce the fundamentals of software engineering to aerospace engineering students through five parallel sets of activities:

1. In-class lectures: The lectures given by the instructor were based on a textbook "Software Engineering" by Ian Sommerville. The software engineering aspects of aerospace engineering were also introduced. The use of other media, such as news and short video clips, was also used. There were also talks given by the guest speakers who had experience related to software engineering in practice.
2. Project: Software engineering project that requires students to follow a V life cycle model.
3. Reading assignments: The materials were chapters in the textbook and journal articles.
4. Homework: Seven sets of homework were assigned throughout the whole semester.
5. Exams: Two midterm exams were given to evaluate the learning of the material from the textbook, assigned reading, and the lectures.

A. The Lecture Material

While some of the class time is used for the project activities, a significant number of traditional lectures are also presented. This material is more relevant and interesting to the students when they can relate it to the project, and we try to synchronize the lecture material and project status as well as we can.

The course outline is shown on the next page. This course is taught in one semester, which is roughly 15 weeks and 45 class periods. The lectures follow the book fairly closely and we used the textbook author's slides as a starting point for the lectures, but have modified and added to them over the years. It is not possible to cover all the chapters, so we have chosen what we consider the most basic material. For example, Formal Methods are not discussed in any detail, but a few slides are used to make them somewhat aware of the subject matter.

In addition to the book, some additional material was added in areas such as: "Facts and Fallacies" by R.L. Glass, overviews of the various programming languages (with their pros and cons), and software standards. These are all covered if there is enough time.

It is not essential to use the same textbook used here, there are other good books (e.g. the books by Pfleeger and Atlee[10] or Pressman[11]). The students are also required to read sections from the *Software Engineering Body of Knowledge (SWEBOK)*[12], the *CrossTalk Journal*[13], and *A Gentle Introduction to Software Engineering*[14]; as well as two of the author's papers[1,2].

B. The Team Project

The team project was introduced on the first day of class and was started in the second week of the semester. In the first week of the class, the students had the general description of the project. After getting some basic concepts in software engineering, students had the freedom to choose to work on any group based on their preferences. Since the different groups main work occurs at different times of the semester, students could also select the group based on their semester schedule. The instructions given to the students were:

> *The goal of the project will be to be able to control the robot remotely to explore an indoor or outdoor area, and keep a record of it in a file on the computer. I want to be able to:*
> - *Drive the robot remotely and see what is in front of it.*
> - *Be able to type comments about what I see in a window*
> - *Comments about what is seen should be saved for later viewing along with data on the location of the robot and the time*
> - *Save images from camera to file with a file name labeled with a time stamp*
>
> *(Note that this description is meant to be free from statements telling you how to do it, that will be your job). At the end of the semester the system will be tested in a competition between the Blue and the White teams. The robot will be tested in a remote environment and you will attempt to find several objects in the field (e.g. 6 fake landmines) as fast as possible. If you run over a landmine, your test will be over. The students will control the robot from the Penn State campus. Dr. Long will take the robot to Penn State's Foundry Park for the test.*

The complete project description is at:

http://www.personal.psu.edu/lnl/440pub/project/

COURSE OUTLINE

1. Project Description (2 Lectures)

2. Introduction (7 Lectures)

    - Motivation and Overview of Software Engineering
    - Introduction to Software Engineering (Chap. 1)
    - Socio-Technical Systems (Chap. 2)
    - Critical Systems (Chap. 3)
    - Software Processes (Chap. 4)
    - Project management (Chap. 5)
    - Facts and Fallacies book by R. L. Glass

3. Requirements (4 Lectures)

    - Concept of Operations (CONOPS)
    - Software Requirements (Chap. 6)
    - Requirements Engineering Processes (Chap. 7)
    - System Models (Chap. 8)
    - Critical Systems Specification (Chap. 9)

4. Software Design (5 Lectures)

    - Architectural Design (Chap. 11)
    - Distributed Systems (Chap. 12)
    - Object Oriented Design (Chap. 14)
    - Real-time System Design (Chap. 15)
    - User Interface Design (Chap. 16)

5. Code Development (10 Lectures)

    - Rapid Software Development (Chap. 17)
    - Software Reuse (Chap. 18)
    - Critical Systems Development (Chap. 20)
    - Software Evolution (Chap. 21)
    - Computer Languages (Ada, C/C++, OOP, etc.)

6. Verification and Validation (2 Lectures)

    - Verification and Validation (Chap. 22)
    - Software Testing (Chap. 23)

7. Management (2 Lectures)

    - Managing People (Chap. 25)
    - Software Cost Estimation (Chap. 26)

8. Software Standards (3 Lectures)

    - IEEE Software Standards, DO-178B, and FAA standards

9. Conclusions (1 Lecture)

III. Project Description

The team project in this course is different from projects of other software engineering courses [3,4] because it is designed for a large group of students to work on one software project for a platform that is provided. The advantages of using one platform is that the costs are reduced as well as the workload for hardware maintenance. The other difference is that each group acts as a part of one large team. Since each group functions as one process in a software process model, only one group of the team will be a coding group which means that the project requires only few people with coding ability. The project (and the course) does not focus on programming but emphasizes the use of software engineering fundamentals to produce a software system. The details of the team project is described in this section.

Choosing the right project is somewhat difficult. The course is only 15 weeks long, and it is only 3 credits. These students are very busy with other courses as well. In addition, programming the software needs to be a relatively small portion of the project, otherwise the coding group will have too large of a role. The project, however, needs to be sophisticated enough to justify the use of all the software engineering processes.

A. Mission Scenario

The semester projects in 2009 and 2010 were to develop a software system for a mobile robot. Mobile robots, which include unmanned air vehicles, unmanned ground vehicles, and unmanned underwater vehicles, are useful in various kinds of mission either on the earth, in the atmosphere, in space, or on other planets. They are suitable for exploring remote areas and working in dangerous environments, for example, Mars Exploration Rovers.[5] The mission of the project was to operate a mobile rover to find simulated landmines in a park at the Pennsylvania State University. The students needed to develop a software system to control the rover remotely via a wireless LAN through the internet.[6] Therefore, the rover could be operated anywhere in the world under a wireless internet connection coverage. In 2009 they were asked to use Java's RMI, but this resulted in a system that required too much from the coding group. So in 2010 they were allowed to simple use remote desktop software to log into the rover and control it.

B. Objectives

The project was designed with three objectives:

1) To provide hands-on experience in software engineering through the development of a relatively small software system.
2) To simulate the real working environment in a large software company by having the students in the class work together as a team. Each student worked in a small group functioning as a part of a software development model. The instructor was the customer who outlined the system needs.
3) To emphasize the communication and collaboration skills among small groups in the software development model which are very important skills in developing large and complex software systems.

C. Hardware System

The hardware system used in 2009 was a Traxxas Stampede radio-controlled truck (Figure 1) with two on-board Logitech cameras in the front to display the real-time environment and one on-board laptop. The laptop was a Dell Mini 9 laptop. It had a 9.8-inch screen, a 1.6 GHz Intel Atom processor, a 1 GB DDR RAM, and a 16 GB solid-state drive. Its dimensions were 9.1 inches in width, 6.8 inches in depth, and approximately 1.25 inches in thickness. Its weight was 2.3 lbs without an AC adapter. It has three USB 2.0 ports and Windows XP was installed. The truck has two servos: throttle servo and steering servos. The servos were connected to the laptop via a Parallax USB Servo Controller Board. The servos were powered by a 7.4-volt Ni-MH battery pack. We found that this system was not ideal, since we only had one of them, and it had a laptop on it. This meant the students had to transfer all their software to it, including Java and driver software. It also meant that the two teams (blue and white) had to share the same laptop. This caused software conflicts and complicated both development and testing.

For these reasons, in 2010 we used a different platform: a SuperDroid rover (example shown in Figure 2). This rover platform allowed the students to develop the software on their own laptops, and then they could use that laptop on the rover. This eliminated a number of problems.

Figure 1. The R/C Truck used in Spring 2009 with the on-board computer and two web cameras.

Figure 2. Superdroid Rover used in Spring 2010.

D. Teams and Groups

A crucial skill for working in a large organization is not only collaboration in a small team but also inter-group cooperation. In order to imitate how a large software company operates, students in the class were required to work in a group which functions as a part of a large team. The interaction among each group is important to drive the project to the final goal, and for them to understand the importance of communication.

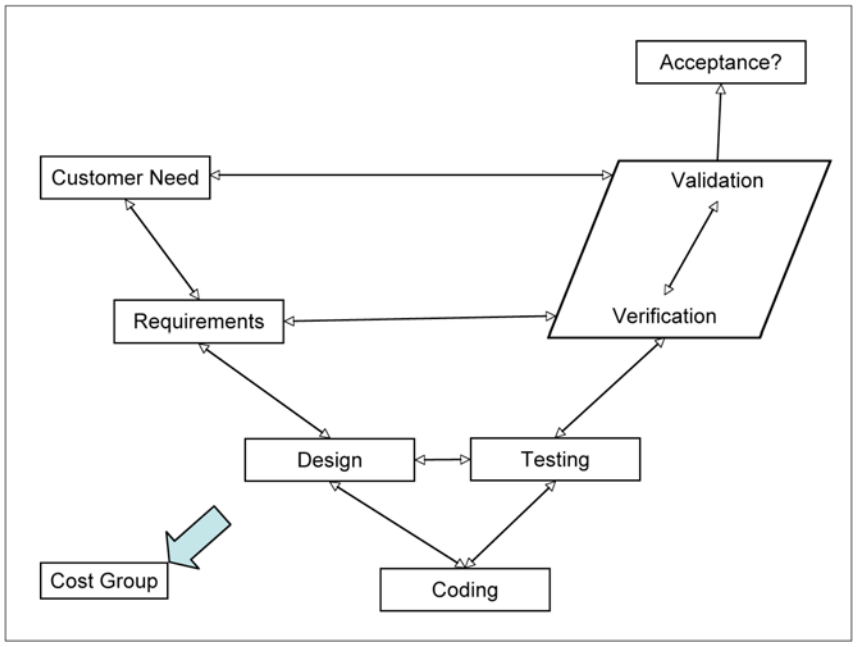

Figure 3. V-cycle used for class project. Each team had groups for Cost, Requirements, Design, Coding, Testing, and V&V.

All the students in the class (roughly 50 students) were divided into two teams, Blue and White, these two teams were competing against each other. Within each team, members were divided into six small groups according to the V-Lifecycle software engineering model, as shown in Fig. 3. Each group had approximately four to five people. The six groups were costing, requirements, design, coding, testing, and verification/validation. The role of each group is defined in the Software Engineering Body of Knowledge (SWEBOK).[7]

The instructor acted as a customer who outlined the system needs to the requirement groups of each team. After the requirement groups finished writing the formal system requirements, they presented their results to the class and their report was given to the design groups. If there were any issues, the two groups discussed them and resolved them (and possibly updated the requirements), and this was also done for the other groups throughout the process. The design group developed the software system design, flow charts and object diagrams of the system, presented their results to the class, and gave their report to the coding group. When the coding groups finished coding, they

presented their results to the class, and the code was given to the testing group. The testing group performed system integration as well as some calibration and testing. Once the system was working, they presented their work to the class and gave their results to the verification/validation group. The V&V group had to ensure that all the requirements were met and the customers needs were met. Note that the interaction among each group on the V-line of the V-Cycle is two-way communication and there were also interactions between three pairs of groups: e.g. design and testing, requirements and verification, the customer and validation. By going through the V-cycle model along the semester, the students got some experience in seeing how the software engineering processes can be used to develop a complex software system. While we used the V-cycle, they also learned about the waterfall model, the spiral model, and even agile methods.

E. Hardware Access

In addition to access to the platforms mentioned in Section C, a set of Parallax Servo Controller Boards, two servos, a USB cable, and a power adapter was provided to the coding group of each team. Therefore, both teams could develop their software at the same time. Once the coding group finished writing up the codes, the software system was handed to the testing group. The testing group of each team had to schedule appointments with the teaching assistant to test their software on the rover. Because the given set of hardware is the same as the hardware on the rover, the software system did not need to be changed.

F. Presentation, Report, and Demonstration

Since each group had roughly two weeks to perform their work, each team gave a brief oral presentation on their progress in class every two weeks. The duration of the presentation for each team was half period of the class. The presentation was closed to the other team in order to make the technical information of each team confidential. After the presentation, each group had to submit a report to the instructor as well as provide it to the next group of the V-cycle model. At the end of the software development cycle, the final product of each team was demonstrated and the final report was submitted.

At the end of the semester, the software system of each team was tested outdoor in the park by performing the mine-finding mission. Each team had one period of the class to set up the system and search for simulated mines. The representatives of the team had to control the truck in a remote closed room by using the images provided by the web cams. Once they found a mine, they marked it in the provided map and the x,y positions were saved on the computer. The team that could find the most mines within the shortest time was the winner. In 2009 the result was that one team could complete the mission while the other team had technical problems and was unable to finish the mission. In 2010, the course was not completed at the time of this paper.

G. Role of the Customer

As discussed earlier, the customer or the instructor only indicates his needs and each team has to understand and write the system requirements. The customer had the right to change the system needs anytime and the teams needed to work on his part. However, it had taken more than half of the semester to finish the V-cycle model. Therefore, the instructor had a chance to change the requirement only once at the very end of the semester. It would be very instructive to also have the customer make a major modification to the goals of the design, but due to time this hasn't worked out yet.

IV. Assessment and Evaluation

The grading of the project was based upon team documents, interactions with other groups, oral presentations, and the final competition. The points from the project were be combined with the points from homework and exams to evaluate the performance of the students.

Student feedback on the perception of the team project was collected three ways: an online survey, an interactive focus group, and the Penn State Student Rating of Teaching Effectiveness (SRTE) form. The focus group was conducted by asking 8-10 volunteers from the class to discuss and to answer questions about their experience working in the team project.

Most students showed positive comments on the team project. They could see the team dynamics and the transition of the work from one group to the other from the beginning of the V-cycle until the end of the project. They said that working in the team project helped them to understand software engineering in practice. They also suggested many ways to improve the team project in the future. In addition, one student said that the course project helped him to get a job. The detailed evaluation of the course project is discussed in Ref. 9.

V. Conclusion and Future Work

Software engineering has become very important in the aerospace industry. Therefore, academia should increase their awareness of software engineering by offering introductory level software engineering courses for students. The Department of Aerospace Engineering at the Pennsylvania State University has offered the introduction to software engineering course since 2007 and a new model of the software engineering course for aerospace engineering students was described in this paper. The implementation of a team project provides hands-on experience and helps students to have better understanding of the software engineering concepts. The course provides positive outcome and the project will be included in the future course.

References

1. Long, L.N., "Computing, Information, and Communication: The Fifth Pillar of Aerospace Engineering," *Journal of Aerospace Computing, Information, and Communication*, Vol.21, No. 1, 2004, pp. 1-4.
2. Long, L.N., "The Critical Need for Software Engineering Education," *CrossTalk: The Journal of Defense Software Engineering*, Vol. 21, No.1, 2008, pp. 6-10.
3. The Engineer of 2020: Visions of Engineering in the New Century, National Academy Press, 2004.
4. Knight, J.C., and Horton, T.B., "Evaluating a software engineering project course model based on studio presentations," *Proceedings of the 35th Annual Frontiers in Education Conference 2005 (FIE '05)*, S2H-21, IEEE, Indianapolis, IN, 2005.
5. Gustafson, D.A., "Using robotics to teach software engineering," *Proceedings of the 28th Annual Frontiers in Education Conference 1998 (FIE '98)*, Vol. 2, IEEE, 1998, pp. 551-553.
6. Erickson, J.K., "Living the dream - an overview of the Mars exploration project," *IEEE Robotics & Automation Magazine*, Vol. 13, No. 2, 2006, pp. 12-18.
7. Long, L.N., Sharma, A. and Souliez, F., "Client-Server Java Programming For Wireless Mobile Robots," *41st AIAA Aerospace Sciences Meeting*, AIAA Paper 2003-0459, Reno, Nevada, 2003.
8. Bourque, P., and Dupuis, R., *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, IEEE Computer Society, Los Alamitos, CA, 2004.
9. Brannon, M.L., Janrathitikarn, O., and Long, L.N., "Evaluation of a Team Project in an Introduction to Software Engineering Course for Aerospace Engineers," *ASEE Annual Conference and Exposition*, Louisville, KY, 2010.
10. Pfleeger, S.L. and Atlee, A.M., Software Engineering: Theory and Practice, Prentice Hall, 2009.
11. Pressman, R., Software Engineering: A Practitioner's Approach, McGraw-Hill, 2009.
12. Software Engineering Body of Knowledge (SWEBOK), http://www.computer.org/portal/web/swebok.
13. CrossTalk Journal, http://www.stsc.hill.af.mil/CrossTalk/2010/03/index.html
14. A Gentle Introduction to Software. www.stsc.hill.af.mil/resources/tech_docs/GISE.DOC