The Critical Need for Software Engineering Education

Dr. Lyle N. Long
The Pennsylvania State University

Software affects almost every aspect of our daily lives (manufacturing, banking, travel, communications, defense, medicine, research, government, education, entertainment, law, etc.). It is an essential part of our military systems, and it is used throughout the civilian sector, including safety-critical and mission-critical systems. In addition, the complexity of many of these systems has been growing exponentially. Unfortunately, the U.S. higher education system has not kept pace with these needs. Existing undergraduate and graduate science and engineering programs need to incorporate more material on software engineering. This is especially true for aerospace engineering, since those systems rely heavily on computation, information, communications, and software. In addition, the United States needs more dedicated software engineering educational programs and professional software engineering certification programs.

Software is everywhere, from cell phones to large military systems. According to the National Academy of Science [1], "... software is not merely an essential market commodity but, in fact, embodies the economy's production function itself." The National Institute of Standards and Technology (NIST) estimates that software errors cost the U.S. economy $59.5 billion a year and software sales accounted for $180 billion [2]. Software engineering education does not get the attention it deserves, even though it is crucially important to our economy. The issues related to software engineering as a discipline and the debates which have occurred over the years, are not new and are described in [3, 4, 5].

The list of software disasters grows each year. Some of the best-known include the following: the Ariane 5 rocket (Flight 501) [6, 7], the Federal Bureau of Investigation Virtual Case File system [8], the Federal Aviation Administration Advanced Automation System [7, 9], the California Department of Motor Vehicle system, the American Airlines reservation system, and many, many more [7, 10]. The F-22 aircraft also had problems initially due to its complex software systems. Software disasters cost the United States billions of dollars every year, and this may only get worse since future systems will be more complex. Boeing spent roughly $800 million on software for the 777, and they might need to spend five times that on the 787 [11]. Aerospace systems will also include some levels of autonomy, accompanied by an entirely new level of software complexity. To help prevent future disasters, we must have more software engineers trained in rigorous technical programs that are on par with other engineering programs. The United States should not wait until there is a disaster that causes large numbers of human casualties before it acts. We do not currently have enough software engineers. We need to educate many more in the near future, especially considering the large group of engineers that will be retiring in the next 10 years [12]. In 2005, the average age of an aerospace engineer was 54 [13]. In addition, more than 26 percent of the aerospace workers will be eligible for retirement in 2008 [14].

Importance of Software in Aerospace Systems

The aerospace industry provides roughly $900 billion in economic activity and accounts for more than 15 percent of the gross domestic product and supports more than 15 million high quality jobs in the United States [14]. These aerospace systems rely heavily on software, which has been called the Achilles Heel of aerospace systems. There are numerous anecdotes and examples that illustrate the importance of computing and software in aerospace. For example:

- The Boeing 777 has 1,280 onboard processors that use more than four million lines of software; Ada accounts for 99.5 percent of this [15, 16].
- The F-22 has more than two million lines of software onboard; between 80 and 85 percent is in Ada [17].
- Some Blackhawk helicopters have almost 2,000 pounds of wire connecting all the computers and sensors.
- The wiring harness is often more complex and more difficult to design than the aircraft structure.
- Some aircraft cannot fly without their onboard computers (e.g., F-16 and F-117).
- The air traffic control system relies heavily on computers, software, and communications.
- Interplanetary robotics and spacecraft perform amazing feats, often in extreme environments.
- Autonomous, intelligent, unmanned vehicles will be even less deterministic than current systems.
- Computers are also important in the design and analysis of aerospace systems. Often this means using high-performance, massively parallel computer systems.
- Communication systems are critically important for aircraft and spacecraft; this now includes computer networking onboard, to the ground, and to other aerospace vehicles.
- Modern aircraft and spacecraft seldom work alone – they are usually part of a system of systems.

One way to measure the need for software engineers in the aerospace field is to research existing job opportunities. An Oct. 2006 review of the Lockheed-Martin Corporation Web site showed they had 536 job openings for recent graduates, including 68 openings (13 percent) in software engineering and four in aerospace engineering. Most of the aerospace engineering job openings were for structural engineers capable of performing finite element analyses. It should be noted that they do hire aerospace engineers in other areas as well (for example, aerospace control experts are sometimes listed under embedded systems). The Boeing employment Web page gave similar results. They had 298 job openings related to software. There were only three jobs that mentioned *aerodynamics* (none of which were actually jobs for aerodynamicists). When searching the Boeing site for *aerospace engineer*, it returned a listing of six open positions in

structural engineering. This is probably an indication that aerospace engineering educational programs are concentrating too much on the applied physics of aerospace engineering and not enough on computing and software. We need to work with industry and the government to redefine aerospace engineering. We need to educate students capable of designing and building the new aerospace systems that we will need in the future – which will be dominated by computing, networking, and information systems.

Software Engineering Defined

The Institute of Electrical and Electronics Engineers (IEEE) defines software engineering [3] as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software." A good summary of software engineering can be found in [18].

Software systems are some of the most complicated things humans have ever created. To design and build them, one needs to follow processes and procedures typical of other engineering disciplines [6, 19]. First, the requirements need to be carefully defined. Then the architecture of the software system needs to be developed. Once the requirements and architecture are defined, one can begin code development. The code then needs verification, validation, and testing. There are many ways of accomplishing all of these steps which are related to the type of life-cycle model used and the type of system developed. In addition, one needs to consider how to estimate costs, how to manage the people, and how to monitor the ethical responsibilities of the team. This is not unlike the steps required to design and build any complex system (e.g., bridges, aircraft, and computer hardware). The actual code development or programming can be a fairly small portion of the process [6].

Most engineers and scientists do not fully appreciate or understand software engineering. Even high school students think they can do software after they learn the basics of Java or C++ syntax. All too often, software engineering is equated with programming. This is like equating civil engineering with pouring concrete. Many people can pour concrete, but few are civil engineers and can build large, technically inspired masterpieces. Likewise, many people can program, but few can develop large software masterpieces. It is not uncommon to hear people arguing about the merits or drawbacks of the different computer languages, even though they are not well versed on the various languages. Often, they simply like the language that they grew up with and do not appreciate or understand the others. These misconceptions are especially apparent in discussions regarding Ada [20], which is still probably the best language to use for mission- or safety-critical systems. In reality, people who develop code without sound software engineering approaches are merely hackers. Of course, programmers are an essential part of software engineering, and talented programmers are quite rare and extremely valuable.

Software Engineering Education

Both the U.S. economy and national defense depend upon software, but many of these large software systems are being developed by people who have never been formally trained in software engineering. While there are some incredibly talented self-taught software engineers, we should not rely on the majority of our software engineers being self-taught. We would never build modern aircraft without aerospace engineers, and we would never build bridges or buildings without civil engineers. So why are we developing large software systems without teams of formally trained and professionally certified software engineers?

"We need to work with industry and the government to redefine aerospace engineering. We need to educate students capable of designing and building the new aerospace systems that we will need in the future ..."

Recently, Dr. John Knight, a professor at the University of Virginia, contrasted software engineering to other engineering disciplines [21]. He spoke of how 1,000-year-old cathedrals were built using the best civil engineering technology of the time and how these buildings are still standing. Civil engineering has evolved tremendously over the ages, and now we have enormous skyscrapers and spectacular bridges. This would not be possible without a vibrant civil engineering educational system, research programs, and mentoring. Similar analogies could be drawn from other engineering disciplines. A thousand years from now, people will be marveling at aerospace engineering milestones such as the Wright Flyer, the SR-71 Blackbird, and the Apollo program. All were great engineering projects in their day and are now in museums. Will there be any software cathedrals to marvel at 1,000 years from now? Or will future generations view us as hackers?

Traditional Science and Engineering Educational Programs

Many students who graduate from U.S. science and engineering programs will eventually work in software development. Unfortunately, most of them will get little or no software education. For example, 24 percent of physics graduates will be working on software five to eight years after graduation [22]. Most of them will probably receive no training in software engineering in college. Other science graduates, even outside of engineering, may also eventually work in software development. It would be very beneficial for these students to know more about software engineering before they graduate. They need more than a freshman-level course in programming. This is true of almost all the traditional science and engineering degree programs.

It is also not valid to assume that computer science graduates are software engineers, either. It is fairly easy to graduate from a computer science program with very little education in software development. Knight and Leveson describe the need for more software education in computer science and computer engineering programs and advocate for more software engineering programs [23].

The need for software education is especially critical in aerospace engineering programs. Aerospace engineers have always prided themselves on being the system integrators, but to do this you must have some understanding of the complete aerospace system you are developing. In modern combat aircraft, the electronic components account for roughly 10 percent of the weight and 33 percent of the cost [24]. So if aerospace engineers are not well versed in computing, networking, sensors, and software then they cannot understand the complete system (unless that system is 60 years old). Students need to be trained so that they can develop the next generation of aerospace systems, not old aircraft and old

spacecraft. Aerospace systems have always used the latest technology to achieve amazing performance. Future and current systems rely heavily on computers and software and students need to know that. Aerospace engineers are needed as system integrators, but this is only possible if they have some understanding of the complete system (including computing and software).

Today this goes beyond the onboard avionics since modern aircraft and spacecraft are almost always tied to other systems, but avionics is a huge part of aerospace systems. Processing power and computer memory have been increasing exponentially in military aircraft since about 1960 [25]. The F-106 had less than 20 kilobytes of memory, while the Joint Strike Fighter (JSF) could have more than two gigabytes. Avionics could account for 40 percent of the cost of the JSF. The report also states that software content in these systems has increased dramatically, and that we need more software engineers.

Computing and software are integral parts of aerospace engineering. It is now one of the key disciplines in aerospace engineering. Traditionally, aerospace engineering [26] was built upon four technology *pillars*: aerodynamics, structures, propulsion, and dynamics and control, as shown in Figure 1. These pillars are reflected in aerospace engineering curricula. All these disciplines were important for the Wright brothers and for every aerospace system since then. However, modern aerospace engineering must include five pillars, as shown in Figure 2. In [27], the authors refer to the five areas as the following: aerodynamics, materials, avionics, propulsion, and controls. Current and future aerospace systems are and will be designed using computers. They will have onboard computers and will need to communicate with other vehicles and computers. Computing (including processing, networking, and storing data) and software are essential elements of aerospace engineering, and they are the fifth pillar. In addition, this fifth pillar might be the most important pillar, and it is far less mature than the other four. Aerospace engineering educational programs have a strong emphasis on applied physics (e.g., fluid dynamics, structural dynamics, dynamics, combustion, and propulsion). Historically, there were good reasons for this, but we cannot continue to neglect the research and educational needs in aerospace computing and software.

"While computing and software is crucial for aerospace systems, existing aerospace engineering educational programs usually do not reflect this fact."

While computing and software is crucial for aerospace systems, existing aerospace engineering educational programs usually do not reflect this fact. Most aerospace engineering programs require roughly 40 courses over a four-year period, but students often take only one course related to software (a freshman-level programming course). Also, there is usually no requirement to learn about avionics, embedded systems, networking, sensors, or computer hardware. This trend carries through to aerospace engineering graduate programs as well, where the entrance exams and curricula seldom include computing, software, or avionics. They are often primarily applied physics programs.

Pennsylvania State University has been working to modernize its aerospace engineering curricula [28]. The university now offers senior-level courses in advanced computer programming (object-oriented programming, Java, C++, Ada, etc.) and software engineering (using [6]), both for aerospace engineers. Penn State also has a new course on the Global Positioning System. As of 2006, aerospace engineering students will be *required* to take either the software engineering course or an electronic circuits course. Ideally, they should take both and also be exposed to systems engineering, embedded systems, networking, information systems, sensors, and software. These additional topics could be covered in their technical electives or in graduate courses. Some of them could be covered in a minor also. The university also hopes to establish an undergraduate minor in information sciences and technology for aerospace engineering in 2008.

It should also be noted that it is difficult to teach an engineer all they need to know in four years. In fact, the U.S. National Academy of Engineering [29] recommends that the bachelor of science degree be recognized as a *pre-engineering degree*. Scientists and engineers need to continue learning throughout their lifetimes to be effective. In addition, many aerospace engineering positions require a master's degree, which allows the student to concentrate on a particular area. An excellent combination would be for a student to get a bachelor of science in aerospace engineering and then a master's degree or doctorate in software (or systems) engineering. These graduates would be extremely valuable. Another possibility is to offer an undergraduate or graduate minor in software engineering. Penn State has a popular graduate minor in computational science, which attracts students from a wide variety of science and engineering departments [30]. A similar program could be created for software engineering or systems engineering.

Figure 1: *Old Aerospace Engineering*

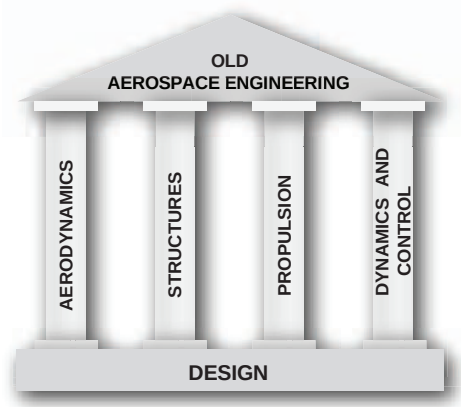
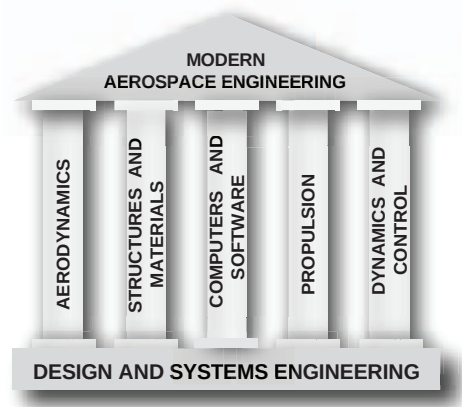


Figure 2: *Modern Aerospace Engineering*



Dedicated Software Engineering Education Degree Programs

As previously stated, existing science and engineering education programs need to include more computing and software in their curricula, but they also need more dedicated software engineering programs. These software engineering programs, however, need to include

plenty of science and engineering in their curricula (e.g., physics, mathematics, and embedded systems). They should not have an overemphasis on management, business, and processes. The Association for Computing Machinery (ACM), the IEEE, and the National Science Foundation have developed very good undergraduate curricula in software engineering [3].

Currently in the United States, there are only 10 accredited software engineering undergraduate programs [31], while there are 67 aerospace engineering programs. The United States needs many more software engineering programs. This needs to happen soon, since it takes years to start new programs and for students to graduate. In addition, the United States has an aging workforce. Some companies in the aerospace and defense business could see 40 percent of their workforce retire in the next five years [12]. According to the Wall Street Journal, organizations such as the National Aeronautics and Space Administration have more engineers over 60 than under 30.

In addition to the existing undergraduate software engineering programs, there are 109 software engineering master's degree programs and 40 software engineering doctorate programs in the United States. Few of the undergraduate or graduate programs, however, are at major research universities. In addition, few of them exist at universities included in the top 25 schools listed in the *U.S. News and World Report* rankings. Most of these programs are at relatively small schools, maybe because they are able to react more quickly to industry and society needs.

Unfortunately, change occurs extremely slowly in academia because there are few incentives to change. Government funding could and should be used to help expedite these changes. Industry leaders need to get involved and demand change as well. There needs to be internships and mentoring available. There is also a need for continuing education. At the government labs and in industry, there is a huge need for software engineering training for its existing workforce.

We also need software engineering professional certification. The IEEE has developed an excellent Certified Software Development Professional (CSDP) program [32, 33]. This is a great program, but it is not quite a software engineering certification program. Unfortunately, there is no requirement for a science or engineering background for the certification, so it is not the same as other professional engineering certification programs. In addition, at the time this article was written, there were only 575 people in the world who have the CSDP certification. Beginning in 1999, Texas began certifying software engineers [34]. In addition, the Open Group has established an information technology architect certification program [35].

Conclusion

Higher education in the United States needs to be more responsive to the software engineering needs of its industry and government labs. The United States cannot be complacent with its technological leads in any field, especially software and aerospace, which are two of the most important industries in its economy. These two industries annually provide more than $180 billion and $900 billion, respectively, to the economy. Technology has been changing at an exponential rate, and too often curricula changes extremely slowly. Software engineering needs to be incorporated into existing science and engineering programs, especially aerospace engineering curricula. We also need to create more dedicated software engineering educational programs at all levels – short courses, bachelors, masters, and doctorate levels. And finally, there also needs to be a national effort to develop professional certification of software engineers.◆

References

1. Jorgenson, D.W., and C.W. Wessner, Eds. Measuring and Sustaining the New Economy, Software, Growth, and the Future of the U.S. Economy. Washington, D.C.: National Academies Press, 2006.
2. NIST <www.nist.gov/public_affairs/releases/n02-10.htm>.
3. IEEE Computer Society and the ACM. "Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering." 2004 <http://sites.computer.org/ccse/SE2004Volume.pdf>.
4. Vaughn, R. "Software Engineering Degree Programs." CrossTalk Mar. 2000.
5. Computer Science and Telecommunications Board. Expanding Information Technology Research to Meet Society's Needs. Washington, D.C.: The National Academies Press, 2000.
6. Sommerville, I. Software Engineering. Addison-Wesley, 2006.
7. Glass, Robert L. Software Runaways: Monumental Software Disasters. Prentice-Hall, 1997.
8. Eggen, D., and G. Witte. "The FBI's Upgrade That Wasn't." Washington Post 18 Aug. 2006.
9. U.S. House of Representatives. Proc. of the Aviation Subcommittee Meeting. Washington, D.C.: 2001.
10. Leveson, Nancy G. "Role of Software in Spacecraft Accidents." Journal of Spacecraft and Rockets 41.4 (2004).
11. Winter, D.C. Presentation at the National Workshop on Aviation Software Systems: Design for Certifiably Dependable Systems. Alexandria, VA: Oct. 5-6, 2006.
12. "The Aging Workforce: Turning Boomers Into Boomerangs." The Economist 16 Feb. 2006.
13. Albaugh, J.F. "Embracing Risk: A Vision for Aerospace in the 21st Century." Frank Whittle Lecture, Royal Aeronautical Society, Jan. 19, 2005.
14. Commission on the Future of the U.S. Aerospace Industry. "Final Report of the Commission on the Future of the U.S. Aerospace Industry." Washington D.C.: National Academies Press, 2002.
15. Hafner, K. "Honey, I Programmed the Blanket." New York Times 27 May 1999.
16. Pehrson, R.J. "Software Development for the Boeing 777." CrossTalk Jan. 1996.
17. Moody, B.L. "F-22 Software Risk Reduction." CrossTalk May, 2000.
18. U.S. Air Force. Software Technology Support Center (STSC). "A Gentle Introduction to Software Engineering." Hill Air Force Base, UT: STSC, 1999.
19. Glass, R.L. Facts and Fallacies of Software Engineering. Addison-Wesley, 2006.
20. Ada Home <www.adahome.com>.
21. Knight, J.C. Presentation at the National Workshop on Aviation Software Systems: Design for Certifiably Dependable Systems. Alexandria, VA: Oct. 5-6, 2006.
22. American Institute of Physics. The Statistical Research Center <www.aip.org/statistics>.
23. Knight, J.C., and N.G. Leveson. "Software and Higher Education Inside Risks Column." Communications of the ACM 49.1 (2006).
24. Sanders, P. "Improving Software Engineering Practice." CrossTalk Jan. 1999.
25. Aging Avionics in Military Aircraft. Washington D.C.: National Academies Press, 1993.
26. Long, L.N. "Computing, Information,

Coming Events

February 4-8
28th Annual Texas Computer Education Association's Convention and Exposition
Austin, TX
www.tcea2008.org

February 13-15
2008 Conference for Industry and Education Collaboration
New Orleans, LA
www.asee.org/conferences/ciec/2008/index.cfm

February 18-22
2008 Working IEEE/IFIP Conference on Software Architecture
Vancouver, BC, Canada
www.wicsa.net

February 25-28
24th Annual National Test and Evaluation Conference
Palm Springs, CA
www.ndia.org/Template.cfm?Section=8910&Template=/ContentManagement/ContentDisplay.cfm&ContentID=18912

February 27-28
AFCEA 2008 Homeland Security Conference
Washington, D.C.
www.afcea.org/events/homeland/landing.asp

April 29-May 2
2008 Systems and Software Technology Conference
Las Vegas, NV
www.sstc-online.org

Coming Events: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.**

and Communication: The Fifth Pillar of Aerospace Engineering." Journal of Aerospace Computing, Information, and Communication 1.1 (2004).
27. "The Future of Aerospace." Washington, D.C.: National Academies Press, 1993.
28. Pennsylvania State University Curriculum Guide 2006-2007 <www.aero.psu.edu/undergrads/UG_Curriculum_Guide_2006-07.pdf>.
29. "Educating the Engineer of 2020: Adapting Engineering Education to the New Century." Washington D.C.: National Academies Press, 2005.
30. <www.csci.psu.edu/minor.html>.
31. ABET <www.abet.org/>.
32. IEEE Computer Society. IEEE Computer Society Certified Software Development Professional Program Web Center <www.computer.org/portal/pages/ieeecs/education/certification/>.
33. IEEE Computer Society. Guide to the Software Engineering Body of Knowledge. IEEE Computer Society, 2004.
34. Voas, J. "The Software Quality Certification Triangle." CrossTalk Nov., 1998.
35. The Open Group. "IT Architect Certification Program" <www.opengroup.org/itac>.

About the Author

Lyle N. Long, D.Sc., is a distinguished professor of aerospace engineering, bioengineering, and mathematics at the Pennsylvania State University. He is also director of the graduate minor in computational science. Long has a doctorate of science from George Washington University, a master's degree from Stanford University, and a bachelor's degree in mechanical engineering from the University of Minnesota. He is an IEEE Computer Society Certified Software Development Professional. Long received the 1993 IEEE Computer Society Gordon Bell Prize. He is a Fellow of the American Institute of Aeronautics and Astronautics, is a senior member of the IEEE, and has written more than 130 technical papers. In 2007-2008, he was a Moore Distinguished Scholar at the California Institute of Technology.

Pennsylvania State University
233 Hammond BLDG
University Park, PA 16802
Phone: (814) 865-1172
Fax: (814) 865-7092
E-mail: lnl@psu.edu

Letter to the Editor

Dear CrossTalk Editor,

Once again, the cover for CrossTalk is a *knockout*! Congratulations to the staff and to the editor for a great edition. I was wondering when systems engineering would be a *transitional* topic about systems engineering and software inclusion.

When I stepped down from the principal investigator position on the B-1B program, I went into software because of the lack of understanding I found between systems engineering and software. Due to my relationship with the company's system engineering manager and the software manager – and the help of CrossTalk articles that I sent them – they finally decided to have meetings in their manager's office and talked together! What a milestone *that* was! I still send CrossTalk articles to these guys, which they are grateful for because it helps keep them current on industry thinking.

This edition should be given to every systems engineer manager and staff, as well as software managers. Let's get the communication going among our contractors!

I could not have been more pleased with Dr. John W. Fischer's introduction in the Sponsor's Note to this month's issue of CrossTalk. His remarks are dead-on regarding the issues and *evolution* of system engineering with software. Gee, can you imagine what the foundation for requirements would start to look like?

Thanks for another great issue!

– Melonee Moses
Software/Logistics Management Specialist
DCMA Boeing
Long Beach, CA
<melonee.moses@dcma.mil>