Software Engineering Education (SEEd)

Peter B. Henderson
Department of Computer Science & Software Engineering
Butler University
Indianapolis, Indiana 46208 USA
phenders@butler.edu

I have strived to sprinkle different points of view from various sources regarding software engineering education through this column. Views from academics, researchers and software practitioners – notably people with extensive experience. This column provides a different perspective, someone just starting their career as a software engineer. I won’t say anything else here as you will find our guest columnist, Jeremy Lanman, speaks for himself.

One quick announcement first. CSEE&T 2006 – Conference on Software Engineering Education and Training. April 19-21 in Turtle Bay, Oahu, Hawaii (SIGSOFT is a cooperating sponsor) is the annual conference addressing issues related to SE education.

Industrial Software Engineering: Developing Software Systems

Jeremy T. Lanman, Software Engineer
Lockheed Martin Corporation
Simulation, Training & Support
jeremy.t.lanman@lmco.com

Software Engineering is an incredibly complex interdisciplinary subject that requires intense problem solving and communication skills. It is an art and a science which requires an open mind and adaptation to various domains… in essence, a software engineer must think outside of the box! I started my path in software engineering with the undergraduate computer science program at Butler University. There I learned the common fundamentals of computer science through studying algorithms and data structures, theory of computing, databases, and of course discrete mathematics. In addition to a traditional computer science program, a software engineering curriculum was introduced. Although, nearing graduation, I decided to take a couple of these software classes, and afterwards it was clear to me that my passion was to develop software systems. I decided to learn more and enrolled into a graduate program in software engineering at Embry-Riddle Aeronautical University. At ERAU, I had specific courses in each area of the software engineering development lifecycle, and collaborative corporate internships to apply my knowledge. My most profound internship was with the aerospace defense contractor, Lockheed Martin Corporation. It was there where I truly learned how to be an “Industrial Software Engineer”.

Lockheed Martin builds massive space and defense software systems that require the implementation of thousands of requirements and billions of lines of code. Not to mention that there are very rigorous government standards forcing systems to be highly reliable. In industry, there are many factors that software engineers must consider such as quality, budget, schedule, risks, contracts and expectation management. Also, there is a level of political communication when working with the U.S. and foreign governments. All of these factors are inter-dependent, and must be addressed in order for a software project to be successful. Ignoring these issues may result in catastrophic failures that can end in law suits, loss of contracts, corporate down-sizing, or worse … human injury and loss of life. Think about it, you would not want a missile to be fired, and due to software engineering malpractice, have the wrong trajectories calculated and in return strike your house. It is important to understand the fundamental and abstract views of software engineering as a project engineering discipline. It is nearly impossible for software engineers to simply sit in a dark room pulling “all-nighters” drinking liters of Mountain Dew and “hacking” out code. Software Engineering is an iterative and incremental learning process.

A formal education aligned with industrial internships in a true software engineering discipline is EXTREMELY important to those that want (or need) to develop “real” software systems. Notice that I did not say software applications. A programmer develops software applications, a software engineer develops software systems … a BIG difference! Of course, some may argue that you can “over” engineer software through rigorous processes and documentation. However, when risks include the lives of people, or the loss of money then these efforts receive more attention. Anyone can write if-statements and for-loops; however, to really engineer a software system is an acquired skill that requires specific initial education, years of experience and continued training.

I find it very important to have a very open mind in learning various other domains. For example, I am currently working on a virtual simulation that requires “life-like” models of humans performing military tasks in a virtual environment. This requires understanding Biomechanics and Human-computer Interaction. Thus, while pursuing a Ph.D. in Industrial Engineering at the University of Central Florida, I am taking these courses to help fully understand how the musculoskeletal system works and how to apply that in a modeling and simulation software system. Moreover, I have had to learn and understand U.S. Army protocols in order to work on constructive simulation projects.

My wife, Apurva, whom also works for Lockheed Martin is a Software Quality Engineer. Her background consists of an undergraduate degree in aerospace engineering and a graduate degree in software engineering. Her job requires that she maintain various metrics and processes for all programs, and ensure that systems are compliant with various regulatory bodies such as
Software Engineering is a “discipline” and requires a formal approach in teaching and learning. It is different from most other disciplines in that you have to continue to LEARN because software engineering is always evolving with new technologies, methodologies, and domains. It is important to learn to think and adapt because that is the essence of software engineering! I am often required to integrate new systems with commercial off-the-shelf or government off-the-shelf products, legacy systems, and interfaces. For example, my last assignment required an integration of a C4ISR\textsuperscript{2} interface in one of our constructive simulations so that the system can communicate via messaging with current Army Battle Command systems being used on the field. This required quickly learning the domains of the other systems and understanding C4I architectures, designs and implementations.

In most other engineering or “cookie cutter” engineering, once something is developed, it is often reused multiple times and becomes a standard. For example, in civil engineering, a bridge is a bridge; not much different then any other bridge. Of course it may differ in appearance, but the design and implementation will be standard, and there is little “integration” of bridges. Unlike many other engineering disciplines, most software systems are custom. There is the limited availability to reusable code, but as software systems become more complex through COTS integration and the building of system of systems interfaces, we find ourselves constantly solving new atypical problems. In order to solve these new problems, we must have an understanding of the basics. That is why I like the path that the software engineering faculty at Butler University is taking to ensure that students receive the fundamentals and practice needed to engineer software systems. I especially applaud Dr. Pete Henderson’s approach to building a software engineering curriculum with a mathematics theme, and Dr. Panos Linos’s method of practice through the Engineering Project is Community Service (EPICS) program.

A very interesting artifact that I find myself often referring to is the Guide to the Software Engineering Body of Knowledge (SWEBOK) a project of the IEEE Computer Society. SWEBOK defines a set of Knowledge Areas (KA) and related disciplines that are fundamental to a formal software engineering curricu-

<table>
<thead>
<tr>
<th>Knowledge Areas:</th>
<th>Related Disciplines:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software requirements</td>
<td>Management</td>
</tr>
<tr>
<td>Software design</td>
<td>Software ergonomics</td>
</tr>
<tr>
<td>Software construction</td>
<td>Mathematics</td>
</tr>
<tr>
<td>Software testing</td>
<td>Systems engineering</td>
</tr>
<tr>
<td>Software maintenance</td>
<td></td>
</tr>
</tbody>
</table>

There are many applications for each knowledge area such as mathematics used in calculating the reliability and availability of a triple-redundant avionics subsystem. Now, some may classify this as systems engineering; however, it is the software engineer that must ultimately design and implement the software to perform such rigorous requirements. Other needs for math include graph theory, algorithms, and linear programming needed to solve complex optimization problems.

Also, SWEBOK describes the characteristics of an engineering profession composed in a study by Gary Ford and Norman Gibbs\textsuperscript{3}. I believe these characteristics are very important in order to be successful in industrial software engineering. They include the following:

- “An initial professional education in a curriculum validated by society through accreditation
- Registration of fitness to practice via voluntary certification or mandatory licensing
- Specialized skill development and continuing professional education
- Communal support via a professional society
- A commitment to norms of conduct often prescribed in a code of ethics”

Companies like Lockheed Martin are looking for “engineers” and not programmers. Industrial Software Engineers will need to be able to think, communicate, and adapt to many different domains. Most companies like Lockheed Martin manage software by calculating the Big Three: Schedule, Cost, and Quality, and each are related to the various factors discussed earlier such as government standards, customer communication, risks, contract management, and expectations management. Today it is difficult to build a high quality software system within cost and schedule. Thus, it is important to understand the tenants of software engineering and all of the underlying fundamentals, and continue to develop new domain knowledge because as software systems become more complex (and they will), it will be the engineer that must solve any new problems that arise. Hence, we need to provide future industrial software engineers the knowledge and MANY tools needed to succeed in building complex high quality software systems on time and within budget.
References

2. C4ISR – Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance protocol.