AC 2009-53: THE CHANGING WAYS OF COMPUTER SCIENCE AND ENGINEERING EDUCATION: A SUITABLE PEDAGOGY TO ADAPT BETTER

Suryaprasad Jayadevappa, PES School of Engineering
PES School of Engineering

Ravi Shankar, Florida Atlantic University
The Changing Ways of Computer Science & Engineering Education: A Suitable Pedagogy to Adapt Better

Abstract:

Computer industry workforce skills required continue to change rapidly. Newer computer technologies are introduced continually while existing ones become obsolete at a faster pace. It is a major challenge to evolve a flexible curriculum that can adapt to those rapid and substantial changes and that new graduates may be taught with. For example, when the Association of Computing Machinery (ACM) released the Computer Curriculum in 1991, networking was not seen as a major topic area. Networking was not a mass-market phenomenon then, and the World Wide Web was little more than an idea in the minds of its creators. Today, networking and the web have changed the way we do business.

Other professional organizations, in addition to ACM, such as the Institution of Electrical & Electronics Engineering (IEEE), and the American Society for Engineering Education (ASEE) are also at the forefront of addressing this challenge. They rapidly and continuously strive to provide the necessary directions in curriculum content and pedagogy. The current “Language First” CSE curriculum globally followed has stood the ground for close to five decades; but it has many inherent flaws. Some among them include emphasis on language and syntax rather than design methodology and problem solving.

We put forth a pedagogy that is flexible, practical, and is based on the “Middle-Out Approach” which is a combination of top-down and bottom-up approaches; typically one integrates these two at an intermediate architect’s level in an attempt to optimize a system. We, however, do not address these optimization and integration issues as part of our curriculum, because of time limitation. They can be undertaken as part of a Master’s program. We categorize our undergraduate curriculum into 4 core categories namely, software, hardware, human-computer interaction and networking. Further each category curriculum is divided into 4 levels: (1) Level 1 (basic/system level) courses broadly include Software Engineering, SOC (System on a Chip) overview, Principles of User Interface Design, and Internet technology; (2) Level 2 (intermediate/component level) courses include Object Oriented Analysis & Design (OOAD), System Level Design, Operating Systems (OS), and Computer Networking; (3) Level 3 (higher/expanded level) courses could include Aspect Oriented/Extreme Programming, Computer Architecture, Modern Computer Design, OS Design, Grid Computing etc; and (4) Level 4 (detailed/micro level) courses include Data Structures & Algorithms Analysis and Design, System Software & Compiler Design, Digital Design, and Advanced Computer Networks.

Exploring pedagogy alternatives to the “Language First” approach and help disseminate domain knowledge better are key to this effort. Further, the proposed curriculum provides stronger design emphasis, better match with industry’s current and future needs, and supports better adaptability. The “Middle-Out Approach” also provides a better platform for lifelong learning. The major hurdle in the implementation of proposed curriculum would be the dissemination of
the proposed approach to the CSE fraternity and availability of skilled faculty to handle the courses at various levels. Acceptance of this approach is more in the mindset of the professors; the ones who can see the inherent advantages will embrace this faster. There is also a need to develop books with a case-study approach with adequate design examples.

1.0 Introduction

The Oxford Dictionary defines “Engineering” as the practical application of scientific ideas and principles. Further an “Engineer” is defined to design and build. Over the years many disciplines of engineering have come into existence and newer ones continue to evolve in order to address societal needs. Since 1970’s Computer Science & Engineering (CSE) has rapidly evolved as one of the major disciplines with large number of students pursuing under-graduate, post-graduate and doctoral studies in search of lucrative careers.

Design is fundamental to all engineering disciplines. As with other engineering disciplines a typical computer science & engineering student must learn to integrate theory and practice, to recognize the importance of abstraction, and to appreciate the value of good engineering design. For a CSE major, design relates to software and hardware components of modern computing systems and computer-controlled equipment. CSE majors apply the theories and principles of science and mathematics to design hardware, software, networks, and processes and to solve technical problems. Continuing advances in computers and digital systems have created opportunities for professionals capable of applying these developments to a broad range of applications in engineering.

According to a survey by National Science Foundation (NSF), around 10% of Computer/IT Engineers lost jobs in the year 2005[1]. Interestingly there will be more jobs in CS/IT related jobs than any other engineering discipline as per the NSF forecast through the year 2014[2]. The bridge from job loss to the coming job gains requires quick deployment of a workforce with the right skills. Academia has a significant role in building a robust and reliable bridge for future employment and research in CS/IT field.

Further according to the Engineer 2020 report [3] by National Academy of Engineers (NAE), the important traits of an engineer of 2020 would include strong analytical skills, creativity, ingenuity, professionalism, and leadership. This forward looking document discusses the major challenges in developing an Engineer for the end of next decade. Immediate relevance among the many suggestions includes the ability to create engineers who are innovative, and capable of “lifelong learning”.

Various professional agencies have played important role in defining the curriculum & pedagogy for CSE. Key among them includes premier agencies such as the Association of Computing Machinery (ACM), the American Society for Engineering Education (ASEE), the Institution of Electrical & Electronics Engineer (IEEE), and the National Academy of Sciences (NAS). These agencies have taken leadership initiatives to assist the CSE fraternity globally.

ASEE [4] is one of the premier organizations started in 1893 to promote quality engineering education in the country and abroad. To achieve this objective ASEE conducts conferences,
workshops, symposium around the year providing the right platform for various faculties and pioneers in the field to interact and share their expertise and ideas.

IEEE Transactions on Education [5] was launched in 1963 for electrical & computer engineering education curriculum & pedagogy. The aims of the IEEE Transactions on Education are both scientific and educational, grounded in the theory and practice of electrical and computer engineering. The scope covers educational methods, educational technology, instructional materials, history of science and technology, and educational and professional development, as well as programs within electrical engineering, computer engineering, and allied disciplines.

ACM[6] is an educational and scientific society uniting the world's computing educators, researchers and professionals to inspire dialogue, share resources and address the field's challenges. ACM strengthens the profession's collective voice through strong leadership, promotion of the highest standards, and recognition of technical excellence. ACM supports the professional growth of its members by providing opportunities for life-long learning, career development, and professional networking.

ACM releases new computer related curriculum guidance approximately once every ten years [7]. This is a substantial effort that involves various leading US universities, high tech companies, and US federal agencies. They discuss relevance of course, content & pedagogy and suggest their inclusion/exclusion. The concept of “knowledge units/area” was first introduced in the 1991 [7] curriculum report; it was further enhanced in the report published in 2001 [8]. The aim or purpose of the proposed 14 knowledge units/areas was to identify and define core contents in computer related curricula. Some thoughts on pedagogical approaches were also discussed in the same report. The 2001 report comments that “Given a field that changes as rapidly as computer science, pedagogical innovation is necessary for continued success.”

2.0 Pedagogy in Computer Science and Engineering – A Quick Overview:

According to the ACM CC2001[8] report, six pedagogy focused groups were assigned to oversee the introductory courses, supporting topics and core, defining core subjects, professional practices, and UG research and computing across the curriculum. Some of the key tasks and challenges are listed below:

- **System-level perspective**- It is critical now than ever to expose the students to a holistic system to facilitate better understanding rather than viewing courses in isolation. The power in system-level conceptualization is enormous and students should be made aware of the same.

- **Appreciation of the interplay between theory and practice**- This is a very fundamental trait of every engineering discipline. The students should be able to practically apply the concepts learnt in theory in order to develop tangible products.

- **Attention to rigorous thinking**- The devil is in the detail! All engineering disciplines should ensure that the students are suitably equipped with the use of sound practices which include planning, tracking progress, measuring and generally managing quality.
Adaptability - Due to the rapid pace of changes in related technologies, graduates of CSE program must possess a solid foundation that allows and encourages them to maintain their skills as the field evolves.

Pedagogy plays an important role in imparting the right knowledge to the students. The CSE pedagogy styles pursued so far can be broadly classified as follows:

- Language First
- Breadth First
- Algorithm First
- Hardware First

The “Language first” approach has survived for over 5 decades – majority of CS/IT programs at undergraduate level focus on programming. The main advantage in this approach has been its simplicity implementation and scope for non CS/IT students to get exposed to CS/IT. There have been many discussions and deliberations about the suitability of this approach in the current context. Some of the disadvantages of the Language first approach are: (a) No design exposure to the students; (b) Lack of system level exposure; (c) Unnecessary emphasis on language syntax; (d) No support for adaptability; and (d) No support for life long learning.

The “Breadth First” approach was considered as an option to provide a more holistic view of the discipline. Many computer science educators have argued for this approach in which the first course introduces a broader range of topics in the CSE discipline. Creating a universal introductory “breadth-first” course that introduces a dynamic field like CSE is a very daunting task. Developing a successful stand-alone breadth-first implementation, however, has proven to be even more difficult. One option though has been to use the breadth-first model as a lead-in to a more conventional programming sequence. This provides the students an immediate appreciation for the breadth of computer science. The primary disadvantage of this approach, however, is that of the vastness of the introductory course.

In the “Algorithms-first” approach, the basic concepts of computer science are introduced using pseudocode instead of an executable language. Using syntax free algorithm/pseudo code encourages students to concentrate on problem solving. Further, it permits students to work with a range of data and control structures without having to contend with the various peculiarities that popular programming languages inevitably introduced. Moreover, because students are freed from the need to execute their programs, this approach permits students to experience the range of such constructs more rapidly. By eliminating some of the time spent on syntax and the details of a particular programming environment, an introductory course that follows the algorithms-first approach can include additional theoretical topics, such as estimation of efficiency and rudiments of computability. Doing so appears to be useful in two respects:

1. For non-majors, it permits some access to the “science” of computer science.
2. For computer science majors, it permits them to encounter appropriate aspects of theory from the very beginning of their course of study, minimizing the risk that they will later experience coursework in theory as an irrelevant curricular appendage.

At the same time, the algorithms-first approach has several critical weaknesses. Important among them is lack of practical experience as these algorithms/pseudo codes cannot be executed and
tested. Additionally this approach doesn’t provide the holistic view of the discipline. Finally, the algorithms-first approach requires substantial grading effort \cite{8}.

The “hardware-first” approach teaches the basics of computer engineering beginning at the machine level and builds up toward more abstract concepts. The basic philosophy behind this strategy is for students to learn about computing in a step-by-step fashion that requires as little demystification as possible. The syllabus begins with switching circuits, uses those to make simple logic circuits, arithmetic unit, and then embeds those in a simple Von Neumann machine. After establishing the hardware foundation, the courses go on to consider programming in a higher-level language. Some inherent flaws in this approach are it is less effective at encouraging students to see the holistic concepts beyond the mechanics of implementation. Further the hardware-first approach does not provide the student the true picture due to the extreme hardware bias given the critical role of software in systems.

3.0 Proposed Approach:

In a rapidly changing field like computer science, focusing too narrowly on specific applications, vendors, or implementations leaves students vulnerable to the problem of obsolescence. Addressing some of the issues with the language first approach and envisioning the requirement of Engineer 2020 \cite{3} is an interesting challenge. In this paper we propose a “Middle-Out” approach which is a combination of top-down and bottom-up approaches. Figure 1 provides the broad overview of course categories of the proposed approach, namely: (i) Software (ii) Hardware (iii) Communication (iv) Human-Computer Interaction. We can always consider a fifth category to fill-in the futuristic courses, math related course, etc.

![Figure 1: CSE curriculum broad course categorization](image)

Each of the above mentioned content groups can be divided into 4 levels, supporting the proposed “Middle-Out” approach as depicted in Figure 2. At each of these levels typical courses that can be considered are indicated. The subjects are organized in a manner such that it fosters innovativeness, design, adaptability and lifelong learning.
Some typical course composition at each level is provided below:

- **Level 1 (basic/system level):** Courses at this level are intended to provide the students with the holistic view of the discipline and the interplay among the categories. The courses provide high level view of the system giving the students an opportunity to appreciate the discipline without burdening them on minute syntactic details. Some typical courses that can be considered at this level include Software Engineering, SOC (System on a Chip) overview, Principles of User Interface Design, and Internet technology.

- **Level 2 (intermediate/component level):** Courses emphasize on design and builds on the system level courses studied at Level 1. Some typical courses that can be considered at Level 2 include courses on Object Oriented Analysis & Design (OOAD), System Level Design, Operating Systems (OS), and Computer Networking.

- **Level 3 (higher-expanded level):** Courses focus on implementation/development of modules in a system. Level 3 courses could include courses like Aspect Oriented/Extreme Programming, Computer Architecture, Modern Computer Design, OS Design, Grid Computing etc.

- **Level 4 (detailed/micro level):** Courses introduce the students to the basic concepts providing minute details of implementation. These courses would help students tweak the system for optimal performance. Some of the current core courses like Data Structures & Algorithms Analysis and Design, System Software & Compiler Design, Digital Design, and Computer Networks can be considered. Our hope and wish is that in the near future there will be appropriate tools available which would automatically make all the necessary decisions at the microscopic levels including decision on the right data.
The following can be considered as one possible implementation for a four year engineering curriculum in CSE. These can be suitably tweaked to accommodate a 2 or 3 semester plan per year.

- **Year 1:** The main objective of the courses offered at Level1 is to provide the system level view. We propose a minimum of 2 courses (CS11 and CS12) for the introductory year. These two mandatory courses should be from level L1. In the current context it would be most appropriate to include courses from Software and Hardware categories. Other options can be considered based on the department strength and degree specialization. An ideal scenario would be to have 4 introductory courses in year 1 that covers all the courses indicated in Level 1.

- **Year 2:** Emphasis in year 2 is more on design. Courses that can foster this capability in the students are considered, as can be seen in the some mentioned courses at level L2 under each category. Additionally, one suggestion is if only 2 courses were offered in year 1 then courses from the other two categories should be offered from level 1. To realize the objective of design, laboratory courses can be considered which provides the necessary exposure to students to the relevant tools and methodology.

- **Year 3:** Learning objective in year 3 is on equipping the students with implementation capability. Courses can be offered from level L3 under the 4 different categories with an option to include courses left out from the previous level L2. Additionally, the first part of the capstone project can be started. We have considered a 2 year capstone project work. The objective of the courses at Level3 is to emphasize on “implementation” of various components in a computer system. Though computer languages will be necessary, we hope that in the near future there will necessary tools that will provide the code once the appropriate design is provided at high level. For example, in a course (L2 level) on OOAD, if we capture the design (functionality, relationship, etc) using UML and sequence diagrams then suitable tools can be developed (currently some are available) to implement the design in an object oriented language like C++.

- **Year 4:** In the graduation year a student is expected to complete the capstone project with results that can be demonstrated. Additionally the student is expected to take an addition of 4 more courses at level L4. Level4 courses provides the students with greater detailing with respect to the implementation and help fine tune the developed module and system.

To get the proposed 200+ credits for a degree in CSE a student has to supplement with courses on soft-skills such as Communications, Leadership, and Ethics, and courses on technology in the global community, etc. These are not indicated here and are to be included as required by individual institutions/department.

### 4.0 Implementation Challenges of Proposed Curriculum:

Albert Shanker, the former president of the American Federation of Teachers, wrote that “educational experiments are doomed to succeed,” in part because the energy their creators bring to the experiment creates an excitement that encourages success [12]. Given enough enthusiasm,
almost any pedagogical approach will succeed as long as its proponents remain committed to that vision. The real test is whether the initial success can be sustained when the approach is adopted by others.

Major challenge would be in the mindset of faculty to accept such a diversion from the traditional practices so far. Another challenge would be in providing necessary study material such as books, tutorials, design examples, and case studies, to support the proposed approach and flow. In some cases, currently available books and material might suffice, in other cases newer material will have to be developed from scratch. We document one such course that uses UML for top-down design [9] of real-world applications. Success of this is documented with another paper where its use for ABET accreditation is presented [10]. We have also developed a bottom-up course with SystemC. We plan to incorporate TLM (transaction level modeling) so that SystemC will be useful for software-hardware codesign. Our intent is to develop a course that integrates the top-down flow with UML with bottom-up components designed with TLM and Visual Studio .NET to emphasize our middle-out approach.

The other major challenge would be in the ability to accomodate the requisite 4 levels in each of the four categories as a part of the 4 year undergraduate program. In the proposed approach a 4 course introductory option as indicated would be ideal. Currently the option of introductory courses are limited to 1, 2 or maximum 3 courses [8].

The evaluation mechanism for the proposed approach is out of the scope of this paper and needs detailed deliberation. Any evaluation mechanism derived needs to ensure that students are tested on the learning objectives of each course at each level. Evaluation details are purposefully kept out of discussion in this paper due its sheer importance and volume. Relevance and gaps in terms of tools, languages, etc can be addressed in detail separately.

5.0 Conclusion & Future Work:

The proposed curriculum is divided into 4 major areas: Software, Hardware, Networking and Human-Computer Interaction. The change in the pedagogy of CSE curriculum is due to the ordering of the courses in each of the major areas. There is shift from the current “language first” approach ensuring that the students are equipped to handle the next generation engineering challenges. The emphasis has been on system design with an opportunity for “lifelong learning”. It also encourages the students to be innovative by relieving them of the pressure of learning the language syntax, etc and concentrating on problem solving.

The proposed curriculum can be extended to accommodate the current traditional courses and also address the growing need of future engineers by adding a fifth course category. Many math courses, basic sciences courses, communication courses, and new cross-domain courses (as with colleges of Arts and Letters, and Business) that add value could be provided here. One can also include courses on optimization at the middle-out level of an architect. Further, newer advanced courses can be added at various levels with suitable changes to the higher levels. A new course added at level 4 will need no change. A new course added at level 3 might make course at level 4 obsolete and need a newer relevant course to be added. Similarly a new course added at level 1 might require changes at all the subsequent levels.
Scheme and process of evaluation of the students undergoing courses in the proposed curriculum should be taken up as a separate task.

References
2. NSF survey on jobs outlook till 2014 at webcasper  http://webcasper.nsf.gov/job_outlook
6. Association of Computing Machinery website  www.acm.org