Introducing Software System Course to Engineering Undergraduate Students - An Experience Report

Karre Sai Anirudh[1], Abhinav Gupta[1], S Lalit Mohan[1] and Y Raghu Reddy[1]

[1]*Software Engineering Research Center, International Institute of Information Technology, Hyderabad, India*

Abstract

Introducing the fundamentals of software systems to early undergraduates is a tedious journey for the instructors. It is crucial for students as it lays the foundation and establishes a perception of the upcoming computing courses. With rapid advancements in technology and considering the changes in workforce practices, it is required to upgrade the course curriculum to up-skill the students in contrast to prevailing times. This paper discusses our journey towards upgrading a decade-old introductory course on Software Systems for early undergraduate students. We present our approach towards updating the course curriculum and its delivery modes to ease learnability among early undergraduate students. We implemented mixed pedagogy methods over the years and captured student feedback to evaluate our approach.

Keywords

Software Systems, Early Undergraduate, Computing Education, Pedagogy

1. Motivation

Technology is changing at an ever-increasing pace, says the leading research and advisory firm Gartner in its report [1] on 'Master Today's Technology Trends.' It also presents empirical evidence on how once called emerging technologies matured faster than ever before. Considering these observations from industry, it is in the capacity of academia to identify trends and upskill the future workforce. Reviewing this in an Indian context, NASSCOM - an Indian national consortium of IT giants observed that the IT workforce might become obsolete unless the reskilling programs are promoted in academia. They claimed that the shelf-life of prevailing skills is diminished to 2-3 years in recent times [2]. Academia can upskill coming generations by upgrading the teaching content in line with emerging technology and trends. In contrast, it is practically not logical enough to teach and upskill freshman undergraduates on every available emerging technology. To begin with, getting them confident in computing fundamentals will help them excel further. Undergraduates who pursue engineering curriculum face the following challenges once they graduate and enter into workforce [2]:

- Fear of programming due to lack of clarity in introductory concepts
- Fail to crack programming interviews as they are not exposed to such evaluation conditions
- Unable to think inside-out in terms of problem-solving as they never experienced competitive programming
- Lack confidence on upskilling at later part of the career due to academic setbacks
- Switch careers into non-technical roles to escape technology productivity

When the fundamental concepts are unclear, it creates a sense of despair for students entering the workforce. We can control such challenges by skilling them with innovative curricula and thoroughly comprehending their skillset. This paper discusses our journey towards addressing these challenges by upgrading our foundation curricula amongst students planning to the entire IT workforce. We at IIIT Hyderabad managed to conduct an introductory two-level course on the foundations of software systems for freshman undergraduates for about two decades now. Since 2018, we have constantly been transforming the curricula and evaluation system so that the aspirants remain confident about their foundations on software systems. In the rest of the paper, we provide details about the progression of our introductory software system course over some time. We also discuss the student's outlook towards the upgraded course structure and illustrate their experiences towards taking this foundation course.

2. Software System Course

Introduction to Software Systems (ISS) is a foundation course formulated in 2018 by merging and updating traditional programming and software systems courses. This course is offered to first-year students at IIIT Hyderabad to learn tools and processes to build simple

Proceedings of 4th Software Engineering Education Workshop (SEED 2021) co-located with APSEC 2021, 06-Dec, 2021, Taipei, Taiwan

saianirudh.karri@research.iiit.ac.in (K. S. Anirudh);
abhinav.gu@research.iiit.ac.in (A. Gupta);
lalit.mohan@research.iiit.ac.in (S. L. Mohan);
raghu.reddy@iiit.ac.in (Y. R. Reddy)

software systems.

Background: Traditionally, this course was offered as two 4-credit workshop courses called *IT-Workshop1* and *IT-Workshop2*. These two workshop courses spanned across two semesters during the first year of the undergraduate engineering program. These courses cover conventional knowledge on assembling and disassembling the computer hardware, installing operating systems, technical aspects of hardware-software-firmware integration, SHELL programming, and web technology concepts focusing on HTML, CSS, and Javascript. This course was designed in the early 2000s to provide broader exposure to freshman undergraduate students on hardware and web technology. These courses are offered for about two decades. Instructors followed collaborative learning and hands-on-based pedagogical methods. With time, the scope and scale of the technology revolutionized the face of software systems in practice. The curriculum and pedagogy approach of traditional courses became outdated. Thus there was a need to upgrade existing course curricula to meet the expectation of current trends of software systems in practice.

Curricula Changes: Considering the trends of technological innovation and software adoption in the IT market, the 'Introduction to Software Systems' course is designed. This course aims to replace the *4-credit* IT Workshop-I and IT Workshop-II with a regular *2-credit* course to provide foundations on programming and technology. Our approach to the course is ***'Practice-Theory-Practice'*** to build software systems while teaching concepts. The scope and extent of the course concepts are discussed below:

- *OS Concepts*: Foundation concepts on operating system, scheduling, memory management and Hardware-Software integration.
- *SHELL*: SHELL command-line interface and introduce basic Linux commands to interact with operating system. We also cover SHELL programming concepts with details on control flows, functions and file handling.
- *Web Technologies*: Concepts on WWW, HTML, CSS, Server-Client examples, JavaScript programming, and JS Libraries like bootstrap.js, aframe.js, and node.js.
- *Database Systems*: Concepts on data and database systems. ANSI SQL Commands to perform CRUD operations, joins and advanced SELECT statements.
- *Python*: Python programming with a focus on keywords, Control Flow statements, Functions, File handling, SQL Alchemy, Flask, etc.
- *Software Engineering*: Software engineering concepts, flavors of programming languages, concepts on object oriented programming, networking (OSI layers, Network appliances, Wired and Wireless communication) and cyber security (OWASP Top 10, Static testing and penetration testing).

The contents of this introductory software systems are finalized after a thorough review. These curriculum contents are gauged based on their relevance and impact on upcoming graduation courses. The foundation concepts of an operating system and Linux commands are practiced first as it becomes the basis for programming. Gradually, we introduced web technologies as it helps students to select desired elective courses on the web and be ready for internship. Considering the increased interest of AI and machine learning, python programming concepts are introduced to students.

Course Delivery: This course is covered for about eleven weeks during an academic semester. There are two one-and-half hours instructor-led classroom sessions per week and three hours of lab sessions per week led by teaching assistants. Each instructor-led class has a live-programming hands-on session with a small class activity. This class activity is conducted to understand the learning levels and engagement of students during the classroom session. There are various graded activities like bi-weekly code assignments to test and explore the concepts through self-learning, Homework to investigate and knowledge discovery from existing resources, Surprise quiz to test their learning consistency. We also conducted mid-term and final term papers, which tests students' programming skills through an automated test evaluation system.

Pedagogy: Pedagogy is a method or a practice of teaching, especially as an academic subject or theoretical concept. As the curriculum contains varied concepts, we adopted different pedagogical approaches to teaching various concepts [3]. Primarily, we followed the *'High-Tech Approach to Learning,'* i.e., we allowed laptops to practice and understand the lecture content with internet access in a classroom setup. We used a learning platform called *'Moodle'* to share learning content, code snippets, do-it-yourself activities, practice questions, assessments, and evaluation, etc. Following are a few of the methods followed while covering specific course concepts.

- *Direct Instruction* - This followed approach to introduce theory concepts from operating systems, software engineering, database systems, and networking. We made every possible attempt to improve the interaction between students by prob-

ing questions while introducing a keyword or a concept to concentrate on the instructor.

- *Inquiry-Based Learning* - Especially to teach programming and code-flow, this will let students build their knowledge by exploring and questioning programming phenomena introduced by the instructor. Students are taught about a code concept, and the instructor asks them to make changes or accomplish specific tasks from the beginning point. Students should do some practice, research, participate in the discussion, and share their insights on suggested code snippets. We followed this approach especially to teach SHELL, Web technology concepts like HTML, CSS, Javascript, and its related JS libraries along with Python programming and SQL statements.
- *Personalized Learning* - Lab Activities are conducted under a personalized learning setup. Teaching assistance introduces a concept in the lab for students who are divided into groups. Students are assigned a small lab activity which is expected to be completed by the end of the lab session. Since the assessment is tailored to the individual, students advance at their own pace during the lab session and spend extra time as needed. During this journey, teaching assistants help them address queries regarding the concepts dictated during the particular session.

Assessments: We conducted assessments in three modes - classroom-based assessments, online-code assessments, and Self-learning assignments.

- All classroom-based assessments are debugging assignments where we provide defective code snippets, and we ask students to suggest corrections to address the defects through visual inspection. These assessments include SHELL, HTML, CSS, Javascript, SQL, and Python program snippets. Some other classroom-based assessments include multiple-choice questionnaires with automated grading.
- All coding exams are conducted through the online code assessment platforms like CodeChef[1] and HackerEarth[2]. We publish the list of questions on this platform and provide certain test cases to address the motive of the provided code question. These assessments are tailored to the individual students as the test cases for evaluation are personalized. In some cases, the questionnaire is generic with specific personalized input-output test cases for evaluation to avoid copy cases. In some cases, we created multiple different questionnaire sets and released them randomly to students based on their odd/even/prime number-based roll-numbers. It is to avoid plagiarism and create a sense of vigilance.
- Self-learning assignments and code-based assessments are released to students bi-weekly. Students spend some learning time solving these code-based assignments and submit them for evaluation.
- All these assessments have a pre-defined evaluation rubric with a clear division of marks and submission instructions announced ahead of deadlines. These assessments are individual contributions with clear expectations.

Evaluations: Almost all online-based assessments have automated grading. The score and results are shared with the students by the end of the assessment. Self-learning assignments are graded through one-on-one sessions between students and their respective teaching assistance. Feedback on coding style, code review, test-case evaluation, and feedback with the scope of improvement are discussed during these sessions. Not all classroom-based assessments are graded. Most of these assessments are awarded in the classroom setup or evaluated on the same day. The scores of these assessments are aggregated to total scores are awarded as final grades to students.

3. Student Feedback

This section discusses our attempts to track and record student feedback about the upgraded course content.

Survey Setup: We intended to investigate how freshman computer science students feel when exposed to software systems and develop software through this survey. We are interested in their feedback regardless of their skillset and expertise. We captured data through a survey called - *'Questionnaire for Freshman Under-Grad Students' Experience towards Programming.'* [4] As part of the survey, we captured the following information from the participants.

- First section of the survey requests participant to self-declare their current skillset and levels of participation in programming before taking up this course.
- In the second section, we asked participants about the difficulty, complexity, understandability, and intellectual effort they spent solving the tasks assigned to them during this course. The captured responses are based on a Likert scale ranging from *'Strongly Agree'* to *'Strongly Disagree'*
- In the next section, we used the Scale of Positive and Negative Experience (SPANE) [5], a survey

[1] https://www.codechef.com/
[2] https://www.hackerearth.com/

instrument, to assess subjective feelings of well-being and ill-being of the participants who undergone specific learning experiences. We repeated this scale to capture the participants' experience while learning SHELL, HTML, CSS, JavaScript, Python, and SQL. The captured responses are based on a Likert scale ranging the experiences from *'Very Rare'* to *'Very Often'*

- In the end, we captured *effort* and *Confidence* queries to understand their effort or time spent on the overall course along with confidence level on programming post completion of the course

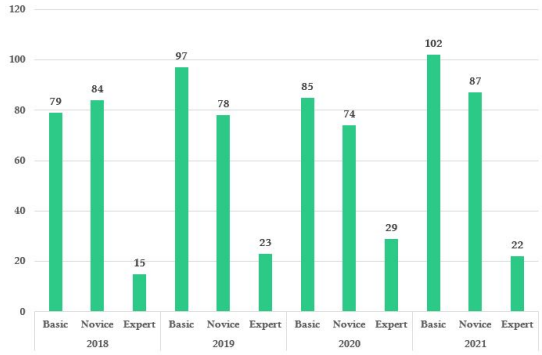

Figure 1: Self-declaration of programming skill-set across years

Participant Demography: We reached out to students who opted for this course to seek their feedback and challenges. We obtained feedback post-completion of the course through a survey questionnaire. We recorded this data for four semesters, i.e., Spring Semester 2018, Spring Semester 2019, Spring Semester 2020, and Spring Semester 2021. During Spring Semester 2018 - 221 students took this course, and 179 participated in the survey. In the Spring semester of 2019, 202 students took this course, and 198 students participated in the study. In spring semester 2020, 222 students have taken this course, and 188 students participated in the survey. In the spring semester of 2021, 211 students took this course, and 159 students participated in the survey. Overall, we reached out to 856 students in four semesters, and 724 of them provided their feedback. Out of all respondents, *36%* of them are female, and the rest are male participants.

Survey Results: We captured data for about four years and illustrated it in this section. We asked students to self-declare their programming skillset before starting this software system course as part of the survey. Fig 1 illustrates the count of participants and their skillset before joining this course grouped as *Basic users* - with no prior understanding about Software Systems, *Novice users* - with basic knowledge about Software Systems and *Expert users*- sound knowledge about Software Systems. We observed that most users are either a novice or primary users with no prior understanding of Software Systems. Few students were found to have solid software systems experience due to self-learning or previous training on computing theory before joining the undergraduate program. They are termed expert users. It is required to capture such data points to understand our target audience.

After completing the course, we asked students about their involvement with the software systems course. Fig 2 illustrates the % time spent by the users during the participation in this course. We observed that most students spent more than *50%* of the time during their participation in this course. However, certain students tend to spend only *5%* and *10%* of the time. On average, we observed that students are proactive and are focused on participating in the course. We also asked students

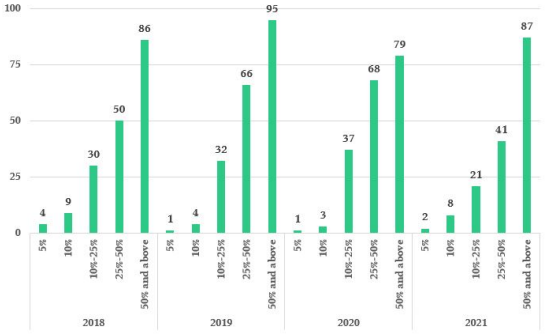

Figure 2: Count of students spending % time on coursework

about their experiences while performing tasks assigned during this course with the Likert scale for agreement ranging from *Strongly Agree* to *Strongly Disagree*. Fig 3 illustrates the student responses aggregated across years, i.e., between 2018-2021. Following are the questions and details about the student observations.

Q1: The tasks were difficult to answer - *39%* of the participants were neutral about the difficulty of answering the tasks. On the other hand, *33%* of the participants found it was not easy to answer.

Q2: The contents of the tasks were complicated - *46%* of the participants were neutral about task complexity. However, *25%* of them found the tasks to be complicated.

Q3: The tasks were challenging - *56%* of them

found the tasks to be challenging. However, the rest of them are neutral and disagree about the tasks being challenging.
Q4: The tasks were easy to work on - *49%* of the participants disagree about the tasks being easy.

Q5*:* The contents of the tasks were easy to understand - *35%* of the participants agree that the tasks were easy to understand and comprehend.

Q6*:* The tasks were easy to solve - *46%* of them disagree that the tasks are easy to solve.

Q7*:* I have put little effort into answering the tasks - *34%* of the participants disagree that they spend less amount of time answering the tasks.

Q8*:* I have not tried hard to answer the tasks correctly - *49%* of the participants have claimed that they have not tried hard to answer the tasks correctly. They have spent a significant amount of time while solving the tasks.

Q9*:* I have tried hard to answer the tasks correctly - *58%* of the participants have claimed that they tried hard to answer the tasks correctly.

Q10*:* I have made an intellectual effort when answering tasks - *45%* of participants strongly agree, and *38%* of them claimed that they made an intellectual effort while answering tasks. They were serious about tasks and solved them with a thorough review.

Q11*:* I have not mainly focused on answering the tasks - *40%* of them strongly disagree, and *39%* of them disagree that they have not responded to tasks with focus.

Q12*:* I have given my best to solve the tasks - *42%* of strongly agree, and *45%* of them agree that they did their best to solve the tasks.

Fig 3 illustrates the data captured across years. We observe participant responses group by Likert scale on the y-axis and question identifier on the x-axis. The circle in the figure indicates the response rate of the participant's responses with percentages labeled in the center. We also asked participants about their confidence levels while learning respective technologies post-completion of this course. Fig 4 illustrates the confidence levels of working on technologies like SHELL, HTML, CSS, Javascript, SQL, and Python. If we carefully observe Fig 4, most participants are confident and moderate about using a particular technology. However, *47%* of participants across the years are not confident about working with python, and *39%* of the participants are not confident about working on SQL. After interviewing the participants, we observed that students are not satisfied with working with python and SQL despite thorough lab activities for practice. Also, in terms of python and SQL - our goal was to provide a preliminary understanding of these programming concepts. Students will eventually learn SQL in detail as part of their future semester full-course on database systems and python programming as part of their future semester full-course on machine learning. This course will create a learning rigor on making the acquaintance about these programming languages and eventually help them excel in their future studies and career.

4. Observations

This section shares our observations and a few areas of improvement identified while conducting this course for nearly four years now.

Curriculum Design: Designing the course contents was challenging for instructors as the concepts involved in this course are elaborate. It was tedious for instructors to plan concluding one concept and switching to another. It required a lot of preparation and more examples to position a concept and introduce another. As it was a 2-credit course, we tried to make the concept short and simple for students to consume.

Student's Grades: *70%* of the students have scored more than *80%* of the score during our first-course delivery. With improvements in our pedagogy approaches, we observed that *78%* of students had achieved more than *80%* in our second-course delivery. During our third edition of this course, *88%* of the students have scored more than *80%* of the score. We observed that the students could consume good content if delivered with a better pedagogy organization.

Students' Observations: We received positive feedback from students on course content and execution. We learned that students faced practical challenges while participating in surprise tests and In-class activities due to technical issues. However, we could address them in a case-by-case manner. While practicing for final exams, the majority of students have approached us for practice programming exercises. We have provided them with many practice programming problems ranging from easy to difficult to test their programming ability and prepared them for their final programming exam. We also observed that a few slow learners could not catch up with the pace of class lectures and lab activities. We scheduled special TA hours for personalized training and helped come out of fear of programming.

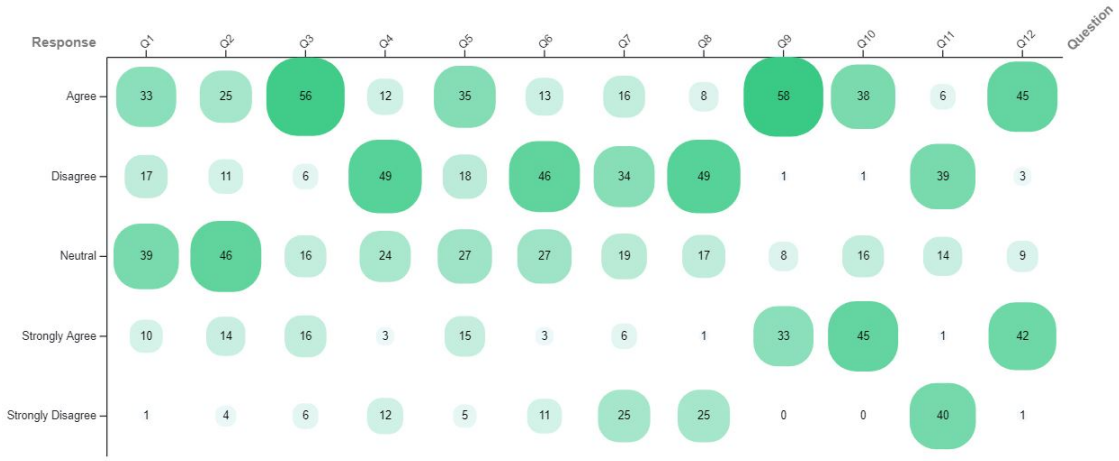

Figure 3: Overall aggregated student experiences across years

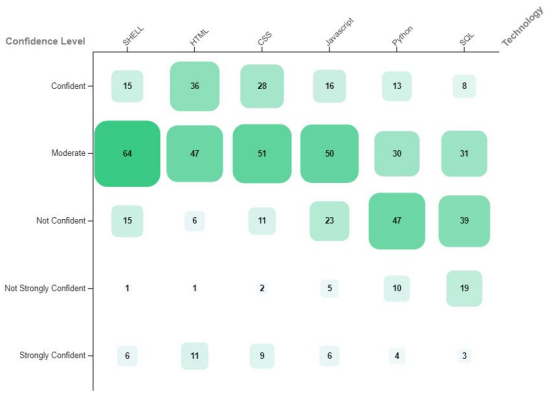

Figure 4: Overall aggregated student confidence levels across years

Pedagogy Methods: We initially practiced a few pedagogy methods that were too challenging to execute. As the enrollment was too high, planning one-one learning sessions or evaluation sessions were difficult to manage. Initially, we failed to meet the expectations of students in the one-one evaluation sessions. Then we strategized our teaching by creating groups of students with different learning levels and later used it as a platform to address doubts and queries on concepts.

Assessments: Planning assessments and evaluation was a difficult task for course instructors as we had to weigh students' learning patterns and plan them carefully. We could not opt for too challenging or too easy questions. We had to compensate for simplified questions, which required ordinary intellect to solve and score points.

Areas of Improvement: Planning the course schedule and relevant assessments together is very crucial. As there are multiple concepts to be covered during the course, it is required to sequence the classes in a particular way to avoid complexities and confusion among students. There is a possibility of pressure on students to learn all these concepts in a short time. However, course instructors and teaching assistants should address students' doubts on concepts in all available platforms. Scheduling TA hours with students is recommended to review and understand their learning levels and bring them up to speed.

5. Related Work

Researchers have conducted various studies and published various experience reports on introducing new courses to freshman undergraduate students. An early study on course upgradation by Albert Crawford shows that computing education should be reviewed and constantly updated based on current trends [6]. In the early 1990s, Charles et al. [7] have pioneered understanding freshman's perspective on computing education and their working style. This work led various other researchers to conduct curricula upgrading in different computing streams. Later Curtis Cook [8] developed a model course to introduce basic computing concepts or system engineering concepts to freshman undergraduate students. This model course contains general concepts of those

times required for novice engineers to scale up for future studies. Celina et al. [9] conducted empirical studies on understanding freshman's perceptions in Electrical/Electronic Engineering courses from four higher education institutions. Pak Kwan proposed a methodology to introduce varients of teaching methods on teaching machine learning courses to freshman undergraduate students [10]. Tom Goulding closely studied the efforts required to introduce complex systems development projects into the college curriculum [11]. Susan et al. [12] shared their experience report on upgrading age-old computing curriculum to include emerging technologies into freshman undergraduate engineering courseware. Brown et al. [13] shared their challenges and experience on upgrading and introducing new 3D modeling concepts on bio-medical education to computing graduates. Considering these experience reports, upgrading an existing course or revamping it into a new model curriculum is challenging. It requires acceptance and support from students and fellow instructors who are involved in this up-gradation process. However, most of these experience reports are unclear about the pedagogy methods they followed. Also, some experience reports did not care about participants' feedback or areas of improvement. As part of our experience, we categorically discuss students' inputs and scope of improving the course into different variants to be offered to any freshman undergraduate engineering class across India or globally.

6. Conclusion

This paper shares our experiences on formulating a two-decade-old introductory course on software systems for freshman engineering students with upgraded course curricula. We followed different pedagogical approaches to teach various concepts to students. We also conducted a multi-level assessment process to assess student performance during the tenure of this course. We captured a student feedback survey to understand students' learning experiences. The feedback showed that the learning experience was unique and different. It helped them understand various concepts quickly. With a typical semester duration of four months, we encourage instructors across India to adopt this course curriculum to educate their freshman undergraduate students about an introductory course on software systems. It will eventually aid them in understanding more complex subjects without hassle.

References

[1] B. Burke, Top strategic technology trends for 2021 (2021).

[2] P. T. of India, Shelf-life of skills now only 2-3 years, says nasscom chairman (2020). URL: shorturl.at/qFKN6.

[3] R. Bogdan, S. K. Biklen, Qualitative research for education, Allyn & Bacon Boston, MA, 1997.

[4] S. A. Karre, Questionnaire for early under-grad students' experience towards programming (2018). URL: https://forms.office.com/r/FbRy8kX4Rn.

[5] W. D. T. W. K.-P. C. C. D. O. S. . B.-D. R. Diener, E., New measures of well-being: Flourishing and positive and negative feelings, Social Indicators Research 39 (2009) 247–266. doi:doi.org/10.1007/978-90-481-2354-4_12.

[6] A. L. Crawford, Functional programming for freshman computer science majors, SIGCSE Bull. 19 (1987) 165–169. doi:10.1145/31726.31753.

[7] C. H. Mawhinney, D. R. Callaghan, E. G. Cale, Modifying freshman perception of the cis graduate's workstyle, SIGCSE Bull. 21 (1989) 78–82. doi:10.1145/65294.65303.

[8] C. R. Cook, A computer science freshman orientation course, SIGCSE Bull. 28 (1996) 49–55. doi:10.1145/228296.228305.

[9] C. P. Leão, F. Soares, A. Guedes, M. T. S. Esteves, G. Alves, I. M. B. Pereira, R. Hausmann, C. A. Petry, Freshman's perceptions in electrical/electronic engineering courses: Early findings, in: Proceedings of the 3rd International Conference on Technological Ecosystems for Enhancing Multiculturality, TEEM '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 361–367. doi:10.1145/2808580.2808634.

[10] P. Kwan, A college freshman's guide to machine learning: Short and sweet way to introduce machine learning to college freshman, J. Comput. Sci. Coll. 30 (2014) 36–37.

[11] T. Goulding, A first semester freshman project: The enigma encryption system in c, ACM Inroads 4 (2013) 43–46. doi:10.1145/2432596.2432613.

[12] S. L. Miertschin, C. L. Willis, A freshman course in emerging information technologies, in: Proceedings of the 4th Conference on Information Technology Curriculum, CITC4 '03, Association for Computing Machinery, New York, NY, USA, 2003, p. 115–118. doi:10.1145/947121.947146.

[13] A. M. Brown, D. R. Bevan, Introducing protein 3-d visualization software to freshman undergraduate students: Making connections and building skills, in: Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact, PEARC17, Association for Computing Machinery, New York, NY, USA, 2017. doi:10.1145/3093338.3093347.