SE-First: A New Approach to Software Engineering Education

Colin Maly
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, USA
cmaly@huskers.unl.edu

Suzette Person
Department of Computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, USA
sperson@cse.unl.edu

Leen-Kiat Soh
Department of Computer Sciecne and Engineering
University of Nebraska-Lincoln
Lincoln, USA
lksoh@cse.unl.edu

***Abstract*—In this Innovative Practice Full Paper, we note that the way software is developed has changed significantly in the past 50 years. Software developers today cannot just be good at writing code; they must also possess non-technical skills in order to work successfully within diverse teams and have an appreciation for the tools and processes needed to build and maintain complex systems. In this work, we describe a novel first-year Software Engineering-First (SE-First) curriculum that introduces students to the broader picture of software development while students learn fundamental computing concepts. To assess the effectiveness of our novel first-year curriculum, we compare students who completed the first-year software engineering curriculum with students who completed our traditional computer science curriculum. We assess student knowledge of computing concepts, and their self-efficacy. Initial results show that students who complete the first-year software engineering courses perform as well or better on the computing concepts test, they are more confident in their computing abilities and in the application of computing skills to their field, and they have a higher success rate in their first-year computing courses (i.e., fewer students drop the course and fewer students receive a D or F course grade) compared with students who complete the traditional first-year computing program.**

Keywords—software engineering, education, curriculum, software engineering-first model, computational thinking, attitudes, perceptions

I. Introduction

With the many advancements in software development processes and tools over the past 50 years, the issue of improving software engineering education at the undergraduate level is both relevant and timely. Software development today includes more rigorous engineering practices, more sophisticated tools, and relies on large teams of developers working across locations and time zones. Despite these changes, students in undergraduate software engineering programs typically take traditional first-year computing courses that focus on essential computer science (CS) concepts, coding fundamentals, and technical skills, rather than engineering principles and non-technical skills, such as teamwork and communication skills. This Computer Science-First (CS-First) approach limits the number of potential opportunities for students to practice non-technical, yet important skills. It also presents students with a limited view of software development in practice and does not accurately reflect how software developers in industry actually spend their time.

For decades, researchers have been studying the gap between the skills of software engineering students upon graduation and the needs of industry. Studies have identified high priority areas where recent graduates are lacking the necessary skills. These include design and architecture, testing, project management, and soft skills [1]. Project management, user interface design, and maintenance are also areas in which software development professionals are lacking [2]. The increasing popularity of the startup industry presents other demands of software engineering graduates, such as the ability to build scalable and clean applications, initiate code reviews, and to be proficient in the areas of dev-ops and cloud computing [3]. Thus, it is critical that software engineering education be designed—or redesigned—in a way that covers foundational computing skills and also prioritizes engineering processes and tools that prepare students for future professional opportunities.

Given the pressures to address the skills gap and the rapidly evolving field of computing, software engineering educators are faced with important decisions regarding which topics to cover, and how to structure the curriculum. Recent efforts to improve undergraduate software engineering curricula have varied in terms of the model(s) used, academic standing of the enrolled students, and evaluation criteria (if any). A problem-based learning approach was presented that utilizes learning environments and group work that match market realities, in order to offer an authentic glance into software engineering work in the real world [4]. This research was further strengthened by research in order to drive modern tool adoption capstone courses that better matches industry needs [5]. The use of virtual labs, especially to enhance education in universities with fewer resources, was investigated; additionally, a holistic model for evaluating effectiveness of this education model as well as others was discussed, which evaluates self-efficacy, performance, attitudes, and learning styles [6]. These research studies, however, do not implement or evaluate the proposed models.

Other attempts to improve software engineering education include a 2018 study in which a flipped classroom setting with a smart diagnosis learning system was applied to a software engineering course in Taiwan; students were instructed to watch lecture videos before class time, use a smart diagnosis system to identify strengths and weaknesses in concepts, and then actively participate in discussions during class time. Student performance, motivation, and aptitude increased in this context [7]. A

case study-based approach to a first-year software engineering course was compared to a traditional lecture-based format; the study found that students were more confident in their own abilities when enrolled in the case study-based first-year course [8]. In other work, a focus on teamwork, using a combination of cooperative and collaborative learning styles, was studied in software engineering courses; this study evaluated individual performance within the team, but did not examine student confidence or attitudes toward computing concepts [9].

Other integrated approaches have been studied in a university setting, involving an inverted classroom technique [10] and a pair-learning course structure [11], both of which increased student satisfaction in learning. However, these studies are limited to software engineering courses for junior- and senior-level students only, as opposed to first-year students.

In this work, we describe a novel approach for teaching computing topics while helping software engineering students develop non-technical skills to address the skills gap identified by industry. In fact, placing greater emphasis on non-technical skills throughout a curriculum has been demonstrated to increase the probability of students considering the end users' desires [12]. Our curriculum is based on a Software Engineering-First (SE-First) model in which students learn fundamental software engineering principles and critical non-technical skills together with coding starting in the first semester. This contrasts with the Computer Science-First (CS-First) model typically used in undergraduate software engineering programs. The CS-First model follows a traditional computer science (CS) core curriculum for the first two years, in which students focus on code writing and algorithm development, succeeded by software engineering-specific courses to build their knowledge in design, architecture, testing and other areas of software engineering. However, a key question remains to be studied: *is the SE-First model an effective way to teach undergraduate students majoring in software engineering as compared to the traditional CS-First approach?* That is, does the SE-First model adequately prepare students in terms of CS knowledge and problem-solving skills, and does it have a positive impact on their attitudes and confidence when compared with a CS-First curriculum?

In order to answer these questions and evaluate the effectiveness of our novel first-year SE-First curriculum, we designed a survey to measure performance, confidence, and perceptions of students across all our computing majors at the beginning of their second year. We then compare the software engineering students (i.e., the students who completed the first-year SE-First curriculum) with students who completed our traditional first-year computer science curriculum. We also evaluate student retention in our first-year courses to determine if student success rate is impacted by the SE-First model.

In this work, our analysis is focused on three research questions:

- RQ1: How does student knowledge of fundamental computing concepts vary between students who complete the first-year SE-First curriculum and students who complete a traditional first-year CS curriculum?
- RQ2: How do students who complete the first-year SE-First core courses compare with students who complete a traditional first-year CS curriculum in terms of attitudes and perceptions towards their computing courses and their ability to apply their computing skills?
- RQ3: How do student retention rates differ in the first-year courses between the SE-First curriculum and the traditional first-year computing courses?

We make two primary contributions with this work. First, we show the effectiveness of the first-year curriculum in our novel SE-First software engineering undergraduate program. We use objective data analysis to compare the performance, confidence, and perceptions of students who complete the first-year SE-First courses versus students who complete the traditional computer science courses. The results of this study can be used to inform the implementation or evolution of software engineering programs at other universities. Second, we analyze factors that may impact student learning and performance in first-year computing courses.

In the next section, we introduce the first-year SE-First software engineering curriculum and describe existing research related to our topic of interest. In Section III, we describe our study methodology, followed by Section IV, in which we summarize the data collected over the past three years. In Section V, we present the results of our analysis, and in Section VI, we discuss conclusions that can be drawn from the research study, the significance of the data, and relevant future work.

II. Background and Related Work

A. Background

The undergraduate program in software engineering offered by the Department of Computer Science and Engineering at the University of Nebraska-Lincoln (UNL) debuted in the Fall semester of 2016. At the time, it was one of just 40 such programs in the United States. Despite the significant costs of developing a new set of core courses to teach fundamental computing concepts in the context of software engineering, the UNL faculty believed that an SE-First model would be an effective approach for teaching software engineering. They also chose this model because it facilitated repeated exposure to software engineering and computing concepts and more opportunities to practice applying these concepts. Early exposure to software engineering concepts also presents a more accurate picture of what software engineers "do" in practice. To the best of our knowledge, our undergraduate program in software engineering is the only program that implements the SE-First model in the first two years.

In the first-semester course, Software Engineering I (SOFT 160), students learn many of the same coding fundamentals and technical skills that are taught in a CS 1 course; however, these topics are taught with an emphasis on building software that is not only correct, but also maintainable and well-tested (as measured by code coverage metrics collected by the students prior to submitting their code). In fact, before students learn to write code, they learn about software qualities beyond correctness and metrics for measuring software quality. Students also work with tools found in practice, such as version control (e.g., Git) and testing tools (e.g., JUnit, EclEmma). At the end of the semester, students work in teams to complete a capstone project that involves real-world specifications and interfaces with real-world software. In the second-semester course, Software Engineering

II (SOFT 161), students continue to learn coding fundamentals (using a different programming language), while gaining more practice with software engineering tools and working in teams. At the end of SOFT 161, students complete a team capstone project, again interfacing with real-world software. At the end of the first year, students who complete the SE-First core courses have covered much of the same foundational computing topics as students who complete our traditional first-year computing curriculum (i.e., CS 1 and CS 2), plus they have practiced basic software engineering skills (e.g., version control, documentation, test suite design).

Other notable features of our core software engineering curriculum (first two years) include a cohort model, small sections (<40 students), weekly hands-on lab sections led by an upper-level software engineering major, and software-engineering-in-practice and software-engineering-in-research presentations by industry partners and faculty, respectively. Recent studies have reinforced this need for the inclusion of industry stakeholders in software engineering curricula at the undergraduate level; in doing so, students not only receive access to greater resources, relevant knowledge, and effective development practices, but the environment also promotes innovation and fosters a startup mentality [13]. Finally, all the core courses in our program are taught by teams of instructors and include instruction and practice in teamwork and communications skills.

B. Related Work

Undergraduate software engineering education has been an evolving area of interest in recent decades. A large body of research has focused on which technologies to teach in software engineering courses, and on pedagogical techniques for delivering course material effectively. For example, a study at the India Institute of Technology in 2012 created a virtual laboratory experience for students enrolled in a software engineering course [6]. The objective of this project was to create a more engaging experience for students by allowing them to "learn by doing" and to reduce barriers and costs associated with traditional, in-person laboratories. The research team used a holistic approach for evaluating academic performance, which involves three aspects: examining the learning experience for students, examining the students' perceived learning, and examining the effect on students' learning outcomes (academic performance, self-efficacy, and learners' satisfaction).

In another study [7], researchers explored using a flipped classroom and a smart learning diagnosis system to teach a software engineering course. The smart diagnosis system uses a quiz-like format to evaluate students' specific strengths and weaknesses after lessons are completed. To evaluate the effectiveness of their approach, the researchers examined learning achievement, learning motivation, learning attitudes, problem solving ability, and usefulness of the system. The results of this study showed that students enrolled in the course involving the flipped classroom approach performed better than students enrolled in the traditional course, both in terms of knowledge of software engineering concepts as well as attitudes toward learning. Furthermore, the flipped classroom approach also significantly increased students' learning motivation. In both studies, the focus is on improving or creating a specific software engineering course, whereas the focus of our work is on redesigning the software engineering curriculum based on an SE-First model that delivers instruction and practice in software engineering concepts typically taught in traditional software engineering course, while teaching foundational coding skills to students in their first and second year of a software engineering program.

Other reported studies related to improving software engineering education have focused on instructional strategies. For example, an inverted classroom instructional technique was the focus of a study reported in [10]. In this study, Choi evaluated the comprehension levels and understanding levels of students enrolled in the non-traditional course; however, the students in this course were either in their junior (85%) or senior (15%) year at the university. In another study, a "pair-programming" or "collaborative programming" approach was offered to junior and senior students at the University of Utah [11]. In this work, researchers investigated the performance of students in terms of teamwork and collaboration, and in code quality. They also evaluated student confidence and attitudes. Although the "pair-programming" approach pushed students outside their comfort zones, 84% of students stated that they enjoyed completing projects more when "pair-programming" was involved as opposed to individual coding. In both of these studies, researchers explored existing instructional or software development techniques in the context of a single software engineering course. In this work, our focus is on the first-year software engineering curriculum, rather than on an individual software engineering course. We compare the effectiveness of an SE-First first-year curriculum with a traditional computer science curriculum to assess student learning (i.e., to determine if software engineering students are developing sufficient proficiency in fundamental computing topics), student attitudes and perceptions, and student success rates in first-year courses.

III. Methodology

To evaluate the effectiveness of our first-year SE-First software engineering curriculum, we designed a survey that is administered at the beginning of each fall semester to students in our third-semester (beginning of Year 2) computing and software engineering courses. This survey collects student demographics, student confidence in completing computer-related tasks, and student perceptions toward the field of computing. It also includes a validated knowledge test [14] covering fundamental computing concepts typically covered in first-year computing courses. In this section, we first highlight the approval process used in Section III.A. Then, we describe the design of the research survey in Section III.B. Finally, we provide an explanation of how the survey is administered and how the data are organized in Section III.C.

A. IRB Approval

This research study has been approved by the UNL Institutional Review Board (IRB). The UNL Human Research Protection Program (HRPP) has certified that the scope and methodology of this research study meet their policies and procedures, and that informed consent is obtained from all participants of the study. The consent form included an option for students to opt out of the study.

B. Survey Design

The population of interest for our study is all students enrolled in a third-semester computing or software engineering course. At UNL, these courses include CSCE 310: Data Structures and Algorithms, CSCE 310H: Honors: Data Structures and Algorithms, RAIK 283H: Honors: Software Engineering III, and SOFT 260: Software Engineering III. All students enrolled in these courses, including non-computing majors, were recruited as part of the study.

There are three main sections to the survey: demographics, self-efficacy, and computing concepts knowledge questions. The survey also asks students for permission to release their ACT or SAT score, current cumulative GPA in their computing courses, current overall GPA, and their computing and overall GPA for the following four semesters.

- *Demographics*: This section consists of nine questions, including questions about a student's gender identification, race/ethnicity, the computing course they are enrolled in, intended major, and expected GPA at graduation.
- *Self-Efficacy*: The survey includes 12 self-efficacy questions in which survey participants indicate their confidence in their knowledge of, or ability to apply, concepts covered in their first-year computing courses. For instance, students indicate their confidence in their ability to decompose problems in ways that can be solved algorithmically, write recursive routines, and use computer and software tools in their field. Student confidence levels are specified on a scale from 0 (completely unconfident) to 100 (completely confident).
- *Computing Concepts*: To assess student learning in the first-year courses, we use 10 questions from a validated computing concepts knowledge test that covers introductory computer science concepts [14].

C. Survey Administration

The survey is administered during the first or second week of classes in the fall semester. Prior to the COVID-19 pandemic, instructors allowed us to use class time to administer the survey (2018 and 2019). In Fall 2020, the survey was administered outside of class time.

Each semester, the online software, Qualtrics, was used to collect survey responses. The typical amount of time to complete the survey was 20-25 minutes. After the survey window closed (approximately two weeks into the semester), the survey results were downloaded as a secure file, coded to protect students' identities, and the results were then deleted from the Qualtrics site.

The survey has been administered three times, which covers all cohorts of students in the software engineering program except the first cohort. Although omitting the first software engineering cohort was not intentional, it provided us with an opportunity to pilot our first-year SE-First curriculum and then make adjustments and improvements accordingly prior to carrying out the study reported in this paper. This is reasonable because, to the best of our knowledge, no other institutions or organizations have attempted to implement the SE-First model for an undergraduate program in software engineering. Hence, our design and development of the core courses was performed without the benefit of models or learning materials to guide what and how to teach first-year SE-First courses.

IV. Data Collected

A. Survey Statistics

We base our analysis of the first-year curriculum on the three years' worth of data collected to date. In total, 448 survey results were obtained. Of these data, 251 (56%) were completed by students who are computer science majors (eight of whom were double majors), 28 (6%) were completed by computer engineering majors (one was a double major), and 128 (29%) were completed by software engineering majors (none of whom were double majors). The remaining 41 surveys (9%) were completed by students from non-computing majors, with actuarial science (14 survey results) being the most common major in that group. Table I provides an overview of the completed surveys organized by student major. Note that the number of software engineering majors is relatively small in comparison to the number of computer science majors because the software engineering program is still relatively new (currently in its fifth year).

TABLE I. Number of Completed Surveys by Major

Major	*Fall 2018*	*Fall 2019*	*Fall 2020*	*Total*
Computer science	86	75	90	**251 (56%)**
Software engineering	50	50	28	**128 (29%)**
Computer engineering	13	8	7	**28 (6%)**
Actuarial science	8	3	3	**14 (3%)**
Other	14	6	7	**27 (6%)**
Totals	**171 (38%)**	**142 (32%)**	**135 (30%)**	**448 (100%)**

Because our programs include both honors and non-honors versions of the course used in our study, we also view the data organized by course, as shown in Table II. The traditional second-year computer science course, CSCE 310 and the corresponding honors course, CSCE 310H, are attended by computer science and computer engineering students. Software engineering majors attend SOFT 260. It should be noted that the software engineering course, SOFT 260, is only open to software engineering students since it expects students to have a foundation in software engineering concepts in addition to a foundation in computing concepts. Honors students who take SOFT 260 have the option to contract for honors credit since an honors version of the course is not currently offered.

TABLE II. Number of Completed Surveys by Enrolled Course

Enrolled course	*Fall 2018*	*Fall 2019*	*Fall 2020*	*Total*
CSCE 310	87	57	63	**207 (46%)**
SOFT 260	46	37	24	**107 (24%)**
CSCE 310H	7	16	14	**37 (8%)**
Other	31	32	34	**97 (22%)**
Totals	**171 (38%)**	**142 (32%)**	**135 (30%)**	**448 (100%)**

Although our primary interest is in a comparison of the first-year SE-First curriculum with our traditional first-year computing curriculum, we also analyze the data collected based on the students' self-reported gender identification to observe student performance, perceptions and attitudes on a more granular level as well as to provide a more complete context when discussing our analysis. Table III shows the response rate based on gender. Here we see 80% of the surveys across the three yeas of the study were completed by male students, 19% were completed by female students, and <1% of the surveys were completed by students of other/unspecified genders.

Finally, of the 448 surveys identified in Tables I—III, we note that although the students completed the demographics section, a number of these surveys were incomplete in other sections of the survey. The extent to which these surveys were incomplete ranged from missing one or more responses to leaving one or more sections unanswered. For the purpose of analysis in later sections of the paper, only survey results that were complete in the self-efficacy sections (412 of 448, or 92%) or the computing concepts knowledge section (398 of 448, or 89%) were included in the analysis.

TABLE III. NUMBER OF COMPLETED SURVEYS BY GENDER

Gender	*Fall 2018*	*Fall 2019*	*Fall 2020*	*Total*
Male	140	117	104	**361 (80%)**
Female	30	25	31	**86 (19%)**
Unspecified	1	0	0	**1 (<1%)**
Totals	**171 (38%)**	**142 (32%)**	**135 (30%)**	**448 (100%)**

B. Course Retention Rates

Data related to course retention rates are critical in determining if our curriculum provides opportunities for all students to succeed and receive supplemental instruction when needed. While higher-than-average DFW rates may be anticipated in computing programs given the challenging nature of the program, patterns of high DFW rates over time suggest the need to evaluate teaching methods, curriculum design, and the availability and accessibility of supplemental instruction for students. The analysis of these data is especially relevant in the early years of delivering our novel SE-First curriculum, in order to assess the effectiveness of learning materials and guide improvements for future cohorts. To compare student retention and success rates in our first-year SE-First courses with our traditional first-year computing curriculum, we collected data starting in Fall 2016, when the first cohort of software engineering majors enrolled in SOFT 160. DFW rates represent the number of students who received a course grade letter of D or F, or who withdrew from the course. Note that students in the software engineering program are required to earn a course letter grade of C+ or higher to proceed to the next course in the sequence, whereas students in our traditional computing course are required to earn a course letter grade of C or higher. The specific DFW rates for each semester and course are shown in Table IV and are examined more closely in Section V.

TABLE IV. ENROLLMENT AND DFW RATES OF FIRST-YEAR COMPUTING COURSES (PRE-COVID 19 PANDEMIC)

Course	*Academic Year 2016-17*		*Academic Year 2017-18*		*Academic Year 2018-19*		*Academic Year 2019-20*	
	# Students Enrolled	*DFW Rate (%)*	*# Students Enrolled*	*DFW Rate (%)*	*# Students Enrolled*	*DFW Rate (%)*	*# Students Enrolled*	*DFW Rate (%)*
SOFT 160	50	22.0	60	25.0	71	31.0	49	22.5
CSCE 155 [a]	483	36.2	425	34.0	403	40.0	424	44.0
CSCE 155H	23	8.7	24	4.2	27	18.5	30	10.0
SOFT 161	36	25.0	48	4.2	45	4.4	34	2.9
CSCE 156 [b]	233	24.5	240	23.0	200	18.0	212	25.0
CSCE 156H	26	0	16	0	17	5.9	31	6.5

[a] CSCE 155 is the traditional Computer Science I course offered at UNL.
[b] CSCE 156 is the traditional Computer Science II course offered at UNL.

V. RESULTS

To evaluate the effectiveness of the first-year software engineering curriculum at UNL we considered three research questions:

- RQ1: How does student knowledge of fundamental computing concepts vary between students who complete the first-year SE-First curriculum and students who complete a traditional first-year CS curriculum?
- RQ2: How do students who complete the first-year SE-First core courses compare with students who complete a traditional first-year CS curriculum in terms of attitudes and perceptions towards their computing courses and their ability to apply their computing skills?
- RQ3: How do student retention rates differ in the first-year courses between the SE-First core courses and the traditional CS first-year courses?

Note that although our survey was completed by students in both the honors and the non-honors sections of our courses and we report the survey response and DFW rates in the previous sections, we only compare the results of students in the non-honors courses since our software engineering curriculum does not yet have honors sections of our first-year courses.

A. RQ1: Knowledge of Computer Science Concepts

RQ1 is a critical factor in assessing the success of our first-year SE-First curriculum. This is because our first-year courses were designed without the benefit of an existing SE-First model showing an effective approach for teaching fundamental computing topics from a software engineering perspective to students without a computing background. Furthermore, all of the course materials for the first-year courses were developed from scratch, including all lessons, labs, assignments, capstone project, and textbook. If students successfully complete the SE-First courses but do not master fundamental computing concepts, then they will not be adequately prepared to complete subsequent or advanced computer science courses.

For this question, we compared the performance of computer science and computer engineering students (i.e., students who completed the traditional first-year computing courses) with software engineering majors (i.e., students who completed the first-year SE-First courses). Surveys from students who completed the traditional first-year computing courses but who indicated they are not a computing major at the time of the survey were omitted for this analysis. This helps to ensure the two populations are similar in terms of general motivation or goals for enrolling in these courses.

In Table V, we show the students' scores and standard deviations on the computing concepts portion of the survey for each year of our study. These questions are from a validated test of 10 questions designed to assess student understanding of CS1 concepts [14]. We also show the average score and standard deviation across the three years of our study, and the average ACT and SAT scores for each group. Overall, software engineering students scored significantly higher ($p < .001$, $\alpha = 0.05$) on the computing concepts portion of the survey (with an average score of 7.6 versus 6.81) than students who completed the traditional first-year courses. They also had a lower standard deviation (1.67 versus 2.09), compared with students who completed the traditional first-year computing courses. Scores shown are the average number of correct answers on ten equally-weighted questions. However, we also note there is a statistically significant difference between the two groups in terms of ACT and SAT scores ($p < .001$ and $p = .002$, respectively). This means that student performance on the concepts test may be influenced by other factors (e.g., prior knowledge) in addition to the first-year curriculum they completed.

Nevertheless, we draw an important conclusion from the concepts test results: the SE-First curriculum does *not* appear to have a negative impact on the software engineering students in terms of computing knowledge even though students who completed the SE-First curriculum received instruction in fundamental computing topics from a software engineering perspective. Furthermore, we also observe that the standard deviation on the concepts test is consistently lower for students who completed the first-year SE-First curriculum versus students who completed the traditional first-year curriculum.

TABLE V. STUDENT PERFORMANCE ON COMPUTING CONCEPTS KNOWLEDGE QUESTIONS FROM SURVEY * P<.001

Group	*Number of correct questions in Fall 2018 / stdev*	*Number of correct questions in Fall 2019 / stdev*	*Number of correct questions in Fall 2020 / stdev*	*Overall number of correct questions / stdev*	*ACT score / stdev*	*SAT score / stdev*
SE students	7.36 / 1.81	7.69 / 1.40	7.85 / 1.90	**7.60*/ 1.67**	30.21* / 3.42	1388^ / 106
CS & CE students in CSCE 310	7.10 / 2.05	6.77 / 2.09	6.47 / 2.08	**6.81 / 2.09**	28.28 / 3.97	1263 / 114

^ $p = 0.002$ * $p < 0.001$

TABLE VI. CHANGE IN PERFORMANCE ON COMPUTING CONCEPTS KNOWLEDGE QUESTIONS

Major	*Number of correct questions in Fall 2018*	*Number of correct questions in Fall 2019*	*Number of correct questions in Fall 2020*	*Change from 2018 to 2020*
Computer science	7.28	7.31	7.38	**+0.10**
Computer engineering	7.64	8.50	6.57	**-1.07**
Software engineering	7.36	7.69	7.85	**+0.49**

In Table VI, we show student performance on the knowledge test by major, dividing the population of students who completed the traditional first-year curriculum between computer science and computing engineering majors. Here we see that students majoring in computer science answered 7.3 of the computing concepts knowledge questions correctly on average in 2018 and 2019, and 7.4 on average in 2020, improving by 0.1 questions over the course of our study. On the other hand, the trend for students majoring in computer engineering fluctuated from year to year; however, we also note this population is much smaller than either of the other two majors so it is difficult to draw any meaningful conclusions from these results. Finally, students who completed the first-year SE-First courses consistently scored higher than the computer science majors, answering 7.4 of the computing concepts knowledge questions correctly on average in 2018, 7.7 on average in 2019, and 7.9 on average in 2020; thus, over the course of this study, this group had a +0.5 question change in the number of computing concepts knowledge questions answered correctly on average.

We also note that performance on the concepts test by computer science and software engineering students in Fall 2020 improved despite the shift to remote learning after spring break in Spring 2020 due to the pandemic. Additional analysis of these results is necessary to identify factors contributing to SE-First student improved performance each year. One hypothesis is that software engineering student performance is increasing each year due to the deployment of additional course materials and the evolution of the new first-year SE-First curriculum, as the faculty gain more experience teaching computing topics from a software engineering perspective.

B. RQ2: Confidence in Computing Abilities

In Table VII, we report student responses to confidence questions, comparing students who completed the SE-First curriculum with students who completed the traditional computing courses. Overall, students in the software engineering program are more confident in their computing abilities as compared to students from the traditional first-year computing program. For the self-efficacy section of the survey, which allowed students to rate their confidence from 0 (unconfident) to 100 (completely confident) in completing 12 computing tasks, software engineering students rated their confidence at 72.8 on average, computer engineering students rated their confidence at 72.0 on average, and computer science students rated their confidence at 69.6 on average. We again calculated p values ($\alpha = 0.05$) in order to obtain an idea of the statistical significance of these data; when comparing software engineering confidence ratings to computer engineering confidence ratings ($p = .82$) as well as to computer science confidence ratings ($p = .07$), the data are compatible with the null hypothesis stating that there is no significant difference between the datasets. Though the data shows no significant differences among students in the three majors, we do see that, across the three years, and despite the pandemic environment, a slight increase in confidence for software engineering majors and computer science majors, while computer engineering students again had fluctuating results, but an overall decline in confidence from 2018 to 2020.

TABLE VII. CONFIDENCE IN COMPUTING ABILITIES

Major	*Confidence rating in Fall 2018 / standard deviation*	*Confidence rating in Fall 2019 / standard deviation*	*Confidence rating in Fall 2020 / standard deviation*	*Overall confidence rating / standard deviation*
Software engineering	71.87 / 18.24	73.16 / 16.16	73.87 / 17.90	**72.78 / 17.36**
Computer science	67.37 / 19.30	70.17 / 18.83	71.39 / 18.26	**69.58 / 18.93**
Computer engineering	73.00 / 15.72	73.25 / 13.82	69.64 / 18.12	**72.04 / 16.66**

Half of the self-efficacy questions (six questions) is about applying concepts, or creating novel solutions in the field of computer science. The other six questions are more specifically related to a student's performance in the classroom. It is interesting to note that students reported feeling less confident on the application and creation questions than they were on the confidence questions overall. However, software engineering students rated their confidence at 71.2 on average for these six questions, while computer science and computer engineering students rated their confidence at 66.4 on average for the same questions. This difference in confidence levels (+4.8 points) is even greater than the difference in confidence levels between the two groups for all the confidence questions (+4.4 points on average). Although we observe the difference between the two groups in terms of their confidence on these six questions is not statistically significant ($p = .09$, $\alpha = 0.05$), we are again encouraged by the fact that the SE-First curriculum does not appear to penalize students either in terms of performance or self-efficacy.

C. RQ1 & RQ2 By Gender

Gender-based comparisons of performance and confidence within populations or across majors is not possible due to the small number of female students in many of these categories. For instance, between 2018 and 2020, only two female computing engineering students completed the survey. Consequently, we compare all male students with all female students in terms of achievement and confidence as shown in Table VIII. Here, we see that male students performed slightly better than female students on the computing concepts knowledge questions, on average. Over the course of three years, male students answered 7.5 of these computing concepts knowledge questions correctly on average (out of 10 questions), as compared to 7.2 questions answered correctly on average by female students. Where we see a bigger gap is in student confidence. In this category, males score more than six points higher than females. As a whole, male students who completed the self-efficacy questions rated their confidence at 71.6 on average (on a scale from 0 to 100), while female students rated their confidence at 65.4. We see that while the difference in terms of test scores is not statistically significant ($p = .16$, $\alpha = 0.05$), the difference in terms average confidence rating between males and females is statistically significant ($p = .002$, $\alpha = 0.05$).

TABLE VIII. COMPARISON OF GENDERS IN PERFORMANCE ON COMPUTING CONCEPTS QUESTIONS AND CONFIDENCE IN COMPUTING ABILITIES

Group	*Number of correct questions (out of 10) / standard deviation*	*Confidence rating (between 0 and 100) / standard deviation*
Male students	7.51 / 1.94	71.64^ / 18.27
Female students	7.17 / 1.81	65.40 / 20.58

^ $p = 0.002$

D. RQ3: Course Retention Rates

Student retention in first-year computing courses is an important metric of success at many universities, including UNL. When the software engineering major was created, special attention was given to the goal of broadening participation in the computing field. From the data shown in Table IX, we can see that the DFW rate (percentage of students who receive a course grade of D, F, or they withdraw from the course) for the first semester software engineering course (SOFT 160), which ranges from 22% to 31%, is consistently lower than the DFW rate for the non-honors CS I course (CSCE 155), which ranges from 34% to 40%. We see a similar story when comparing the DFW rates for the second semester software engineering course (SOFT 161) which started higher than the CS II course (CSCE 156) DFW rate, but has remained considerably lower (<5%) while the CS II DFW rate remains much higher. In Fall 2020, the DFW rate for SOFT 160 increased to 42% and the CS I DFW rate was also 42%. Our hypothesis for this dramatic increase in DFW rate is the delivery format change to a hybrid model due to the pandemic which did not lend itself to the aspects of the course design to engage students in an in-person environment and was especially difficult for many first-year college students.

TABLE IX. DFW RATES (%) OF FIRST-YEAR COMPUTING COURSES

Course	*2016-17 Year*	*2017-18 Year*	*2018-19 Year*	*2019-20 Year*
SOFT 160	22.0	25.0	31.0	22.5
CSCE 155	36.2	34.0	40.0	44.0
SOFT 161	25.0	4.2	4.4	2.9
CSCE 156	24.5	23.0	18.0	25.0

VI. Conclusions and Future Work

A. Conclusions

In this work, we present the results of a study evaluating our novel first-year software engineering program based on a Software Engineering-First (SE-First) model. The first-year courses in our program teach students fundamental engineering principles at the same time that they learn to code. This provides students with a more complete picture of software development beginning in their first semester. To evaluate the effectiveness of our novel first-year SE-First curriculum, we designed a student survey to assess student knowledge of fundamental computing concepts and their self-efficacy. We have collected three years of survey data (Fall 2018, Fall 2019, and Fall 2020), resulting in a total of 448 survey responses.

Our initial analysis of the survey results shows interesting trends, especially when comparing the data across our three computing majors. First, students in the software engineering major consistently performed better than students who completed our traditional computer science first-year courses on a computing concepts test at the beginning of their second year. Software engineering students were also more confident in their own computing abilities as compared to computer science and computer engineering students. And, the average confidence of software engineering students has increased each year in our study.

The data obtained from this research study have enabled us to see that an SE-First core curriculum is an effective model for an undergraduate program in software engineering. Although we have only just begun to analyze the data, the initial results are promising, showing that students who complete the SE-First first-year are not disadvantaged in terms of learning fundamental computing concepts nor in gaining confidence in their ability to apply their computing skills. Furthermore, we see that as the program continues to evolve and improve, we become more practiced at teaching computing fundamentals from this perspective, and we develop more learning materials, student achievement and confidence continues to grow. This study not only provides data to support this groundbreaking approach to teaching computing concepts in an SE-First model, but it also provides a groundwork for measuring student performance in introductory computing courses around the world.

One limitation of the reported findings involves the fact that software engineering students had, on average, higher ACT and SAT scores than computer science and computer engineering students, and thus one might argue that this difference led to the higher performance in the knowledge test. Further analysis is needed in order to examine if student performance on the ACT or SAT is linked to trends in course performance and confidence. Similarly, another threat to the validity of our results may be that DFW rates do not capture the reason(s) why students perform better or worse in a class. For instance, non-course-related factors such as personal events may impact a student's success in a course, as opposed to course-related factors such as implementing the SE-First curriculum. Even in courses that implemented the SE-First curriculum, we were not able to control for certain factors in first-year courses such as class size, instructional techniques, and teaching approach (individual instructor versus team teaching), which may explain differences in student performance in different semesters. Finally, we note that not all students in the examined courses were required to participate in the survey, meaning that results may not be fully representative of all students in the first-year computing courses.

B. Future Work

Although the initial results of our study are promising, much work remains to analyze the data we have collected to help us understand the impact of our novel SE-First curriculum on student learning, attitudes, and perceptions. To begin with, we plan to continue our analysis to identify factors that may impact student performance and confidence in their first-year courses (e.g., gender, race/ethnicity, ACT/SAT score, first-generation status, native written/spoken language). An intersectional approach can be utilized in order to identify how students' multifaceted identities may be associated with higher or lower performance and confidence levels. We also plan to analyze student performance at a more granular level (e.g., on individual questions on the concepts test, goals, and attitudes). Finally, it is important to continue to develop tools such as surveys or instructor evaluation that adequately measure student progress in acquiring non-technical skills, in order to determine whether or not students are becoming more well-rounded software engineers in the 21st century.

In this work, we focus solely on student achievement and confidence at the beginning of the second year, which can be a critical time for students to decide whether to stay in a major or change to another program. However, student success in future years of their program is another way to measure the success of the program. Analyzing computing course grades in subsequent years in the program and graduation rates may highlight differences in programs that need to be addressed to ensure all students are adequately prepared for advanced courses. Establishing a study to track and observe changes in student attitudes and perceptions in their second, third, and fourth years may also provide valuable insights as students take additional computer science and software engineering courses.

Comparing students in the SE-First software engineering program with other computing majors in our department provides a useful comparison for ensuring students who complete the first-year SE-First courses are adequately prepared for future computing courses. However, it would be interesting to compare students in our SE-First software engineering program with students who take a traditional CS-First software engineering program of study. Does one model provide advantages in terms of student performance? Are students' attitudes, perceptions or confidence significantly different in one model versus the other? Do students persist better under one model or the other? By answering these, and other questions that address the changing landscape of software engineering in practice, educators can design the best program for delivering an undergraduate program in software engineering education that attracts a diverse body of students who will build the next generation of software systems.

Acknowledgment

The authors would like to acknowledge Dr. Brady Garvin for his significant contributions to the development of the first-year SE-First curriculum at UNL.

REFERENCES

[1] V. Garousi, G. Giray, E. Tuzun, C. Catal, and M. Felderer, "Closing the gap between software engineering education and industrial needs," IEEE Software, March-April 2020, vol. 37, no. 2, pp. 68-77, doi: 10.1109/MS.2018.2880823.

[2] T. C. Lethbridge, "A survey of the relevance of computer science and software engineering education," Proceedings 11th Conference on Software Engineering Education, Atlanta, GA, USA, 1998, pp. 56-66, doi: 10.1109/CSEE.1998.658300.

[3] N. M. Devadiga, "Software engineering education: converging with the startup industry," 2017 IEEE 30th Conference on Software Engineering Education and Training, Savannah, GA, USA, 2017, pp. 192-196, doi: 10.1109/CSEET.2017.38.

[4] S. C. Dos Santos, "PBL-SEE: an authentic assessment model for PBL-based software engineering education," IEEE Transactions on Education, May 2017, vol. 60, no. 2, pp. 120-126, doi: 10.1109/TE.2016.2604227.

[5] A. Fontão, B. Gadelha, and A.C. Júnior, "Balancing theory and practice in software engineering education—a PBL, toolset based approach," 2019 IEEE Frontiers in Education Conference, 2019, pp. 1-8, doi: 10.1109/FIE43999.2019.9028606.

[6] B. Pati, S. Misra, and A. Mohanty, "A model for evaluating the effectiveness of software engineering virtual labs," IEEE International Conference on Technology Enhanced Education (ICTEE), Amritapuri, India, 2012, pp. 1-5, doi: 10.1109/ICTEE.2012.6208602.

[7] Y. Lin, "Impacts of a flipped classroom with a smart learning diagnosis system on students' learning performance, perception, and problem solving ability in a software engineering course," Computers in Human Behavior, 2019, vol. 95, pp. 187-196, doi: 10.1016/j.chb.2018.11.036.

[8] K. Garg and V. Varma, "A study of the effectiveness of case study approach in software engineering education," 20th Conference on Software Engineering Education & Training (CSEET'07), Dublin, Ireland, 2007, pp. 309-316, doi: 10.1109/CSEET.2007.8.

[9] J. Chen, G. Qiu, L. Yuan, L. Zhang, and G. Lu, "Assessing teamwork performance in software engineering education: a case in a software engineering undergraduate course," 2011 18th Asia-Pacific Software Engineering Conference, Ho Chi Minh City, Vietnam, 2011, pp. 17-24, doi: 10.1109/APSEC.2011.50.

[10] E. M. Choi, "Applying inverted classroom to software engineering education," International Journal of e-Education, e-Business, e-Management and e-Learning, 2013, vol. 3, no. 2, pp. 121-125, doi: 10.7763/IJEEEE.2013.V3.205.

[11] L. A. Williams and R. R. Kessler, "The effects of 'pair-pressure' and 'pair-learning' on software engineering education," Thirteenth Conference on Software Engineering Education and Training, Austin, TX, USA, 2000, pp. 59-65, doi: 10.1109/CSEE.2000.827023.

[12] R. Chanin, J. Melegati, A. Sales, M. Detoni, X. Wang, and R. Prikladnicki, "Incorporating real projects into a software engineering undergraduate curriculum," 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2019, pp. 250-251, doi: 10.1109/ICSE-Companion.2019.00099.

[13] O. Cico, L. Jaccheri, A. Nguyen-Duc, and H. Zhang, "Exploring the intersection between software industry and software engineering education – a systematic mapping of software engineering trends," Journal of Systems and Software, 2021, vol. 172, pp. 1-28, doi: 10.1016/J.JSS.2020.110736.

[14] M. S. Peteranetz and A. D. Albano, "Development and evaluation of the Nebraska Assessment of Computing Knowledge," Frontiers in Computer Science, 2020, vol. 2, article 11, doi: 10.3389/fcomp.2020.00011.