2021 ASEE ANNUAL CONFERENCE
Virtual Meeting | July 26–29, 2021 | Pacific Daylight Time
Paper ID #34240

Learning a Second Language and Learning a Programming Language: An Exploration

Ms. Jutshi Agarwal, University of Cincinnati

I am a PhD candidate in Engineering Education with a research focus on professional development for future faculty. Currently, I am the Lead Graduate Teaching Assistant for the first year engineering design course with an enrollment of 1300 students across all engineering majors.

Dr. Gregory Warren Bucks, University of Cincinnati

Gregory Bucks joined the Department of Engineering Education at the University of Cincinnati in 2012. He received his BSEE from the Pennsylvania State University in 2004, his MSECE from Purdue University in 2006, and his PhD in Engineering Education in 2010, also from Purdue University. After completing his PhD, he taught for two years at Ohio Northern University in the Electrical and Computer Engineering and Computer Science department, before making the transition to the University of Cincinnati. He has taught a variety of classes ranging introductory programming and first-year engineering design courses to introductory and advanced courses in electronic circuits. He is a member of ASEE and IEEE.

Dr. Kathleen A. Ossman, University of Cincinnati

Dr. Kathleen A. Ossman is an Associate Professor in the Department of Engineering Education at the University of Cincinnati. She teaches primarily freshmen with a focus on programming and problem solving. Dr. Ossman is interested in active learning, flipped classrooms, and other strategies that help students become self-directed learners.

Prof. Teri J. Murphy, University of Cincinnati

Dr. TJ Murphy is a professor in the Department of Engineering Education at the University of Cincinnati.

Dr. Cijy Elizabeth Sunny, Baylor University

Dr. Cijy Elizabeth Sunny is a PD Research Associate in the Department of Information Systems and Business Analytics, Hankamer School of Business at Baylor University. She is a research methodologist and psychometrician who has applied her skills in quantitative and mixed methods research methodology in the substantive areas of STEM education research, medical education, and more recently in engineering education. Additionally, she has been an educator and has taught primarily physics and also research methodology on three different continents. In addition to research, she has also conducted workshops on using concept mapping methodology for scale development, mixed methods research methodology for standardized patient educators, and standard-setting for physician educators. Dr. Sunny continues to invest her skills in engineering education research through her collaborations. As part of her new undertaking at Baylor University, she is investing her skills as a research methodologist and data analyst to fight human trafficking through the use of Information Technology working alongside the research team there in collaboration with a diverse group of stakeholders.

Learning a Second Language and Learning a Programming Language: An Exploration

Abstract

Computing has become a foundational subject across the engineering disciplines with many first-year engineering curricula either including a course on computing or integrating computing within a broader introductory course. However, there is significant evidence that students have difficulty both learning and applying the computing concepts traditionally covered. Many different theories have been proposed for why students find computing concepts so difficult to understand. Some theories ascribe the difficulty in learning computing concepts to the abstract nature of the concepts and the lack of prior experience on which to build understanding. Other theories relate the difficulty to students incorrectly applying natural language structure to computing structures.

It is this latter idea that is of interest in this paper. Learning multiple natural languages has often been associated with increased mathematical and scientific aptitude. Research also suggests that learning a new computing language may be similar to learning a natural language. However, little research has been done exploring the relationship between natural language acquisition and computing language acquisition, with even less on whether strategies employed for learning a natural language transfer to learning a computing language, particularly in an engineering setting.

This full research paper aims to explore the first idea of whether a relationship exists between learning a second natural language and learning a computing language. All first-year students in the College of Engineering and Applied Science at the University of Cincinnati are exposed to computing concepts through a common first-year engineering curriculum. To explore if a relationship exists between learning a second natural language and learning a computing language, student performance in the first-year engineering courses was evaluated based on previous or concurrent experience with second natural languages acquisition and controlled for by prior experience with computing languages. For purposes of this study, experience with a second natural language was defined as having successfully taken at least one university-level foreign language course (e.g., Spanish, French, German, Italian) either with AP credit (from high school) or in the first year of college. Experience with a computing language was defined as having taken at least one university-level computing course either in high school or in the first year of college. Non-parametric tests were conducted to investigate whether there were significant differences between students who had experience with a second natural language and those who didn’t. Statistical analysis showed answers to the research question was inconclusive. Future work will involve further investigation into the reasons behind inconclusive results.

Background

Computers have become an essential part of the design process across all fields of engineering. They are used to model potential solutions through simulation, collect and analyze data, and create new parts through computer aided design packages and rapid/additive manufacturing techniques. Additionally, they are also becoming primary components of the products of design themselves. Computing devices are being integrated into everything from clothing that can monitor vitals to building materials that can report on the stresses and strains they are experiencing. Because of this, computational thinking has been identified as one of the key skills that future engineers will be required to possess[1]. As a result, many first-year engineering curricula include either a course devoted entirely to computing concepts or incorporate those concepts into more general introductory engineering courses.

Despite the need for engineers with strong computational thinking skills, there is little opportunity for engineering students to develop those skills. Learning to program is widely viewed as one of the more difficult subjects experienced by first-year students[2]. One reason for this is that learning to program requires a student to develop not only an understanding of specific concepts, but also a way of thinking not encountered in other disciplines. On top of the already difficult concepts and a completely new way of thinking, students are also required to learn a new tool in the form of the programming environment utilized in the class[3]. As a result, many students fail to develop a sufficient level of proficiency in programming, even after progressing through the traditional two or three course introductory programming sequence followed by many computer science programs[4, 5]. This makes it even more difficult for engineering students to gain proficiency when they often only have exposure to this content in a single course, or potentially only as part of a course. Thus, an important area of research for engineering education is to design ways to help students learn computational thinking skills and concepts more efficiently in an effort to better prepare students for the demands of their future careers in the limited time allotted in the curriculum.

The primary frameworks used to investigate the ways students learn computational thinking concepts is that of information processing and mental model theory. Information process theory treats the student as processor of information, similar in structure to that of a traditional computer system[6]. Students receive information through their senses (receptors), process that information utilizing either working or long-term memory, and enact changes in their environment through effectors. This model is represented in Figure 1 below.

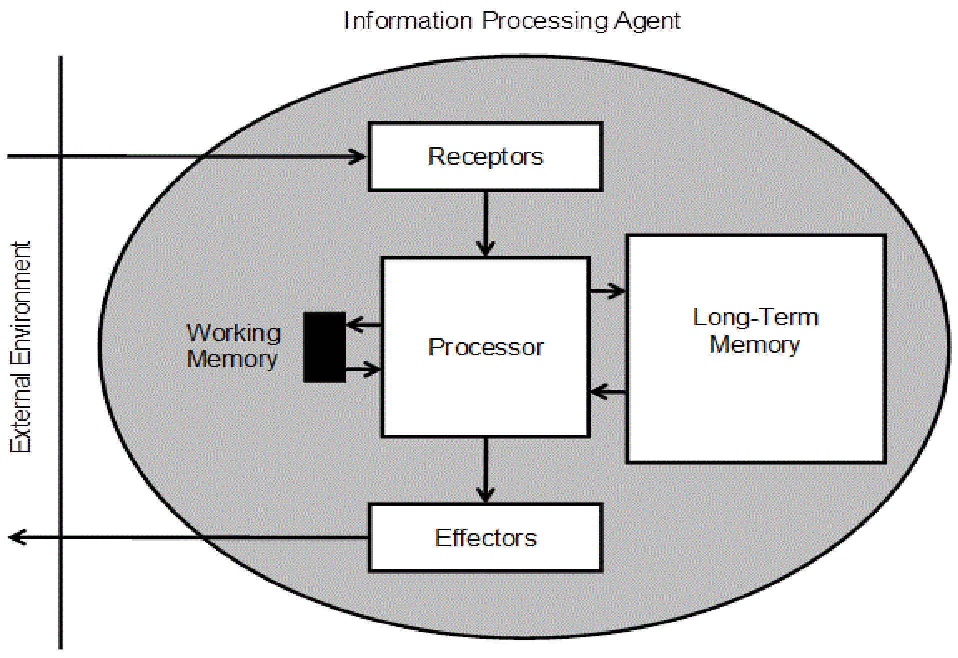

Figure 1: Information Process System Model[7]

Mental model theory is then used to describe how information is stored in long-term memory. In mental model theory, information is stored as a network of interconnected nodes (concepts) which emulates the individual's understanding of the structure and relationship of the material[8]. The individual can retrieve information about a particular subject by either accessing the node associated with that specific information or by accessing nodes connected to it within the individual's mental model structure. As individuals gain expertise within a given area, they develop both additional connections between nodes as well as a hierarchy (known as chunking) that allows for more efficient retrieval of information[9].

Applying this framework results in several different reasons for the discrepancy between the learning outcomes desired by instructors and employers and student performance. One is that the nature of the material does not match well with the learning styles of most engineering students, leading to inefficiencies in the information acquisition process. Numerous studies have looked at the learning style preferences of engineering students[10-13], and those preferences have been shown to be consistent across different populations[14]. These studies have found that engineering students tend to prefer visual and sensing learning styles. However, since most programming languages are text-based, a mismatch arises between the content and the ways students prefer to learn. It has been shown that interpretation of the written word is processed in the same way as spoken words[15], meaning that even though a student may be seeing (visual) a program, the brain is processing the information as if it were spoken (verbal). As a result, students who prefer to learn in a visual manner may have difficulty assimilating programming. Additionally, interpreting written words favors individuals who prefer to learn intuitively rather than from a sensing perspective, again putting engineering students at a disadvantage.

Many students lack appropriate mental models on which to build conceptions of the important programming concepts required to gain proficiency in computational thinking[16]. In conjunction with this, many text-based languages use syntax that incorporates many English language words and phrases, resulting in students resorting to models previously developed in a natural language context. However, this poses a significant problem for certain terms because the model for how the word is used in a natural language context may differ from how it is used in a programming context.

For example, in natural language, the term "while" has a somewhat different meaning than it does in programming usage. In natural language, "while" implies that as soon as the condition associated with the statement is no longer satisfied, the activity will cease. In a programming context, the conditional statement associated with the "while" is only checked once during each iteration of the repetition. This can result in misconceptions among students if they believe that as soon as the condition for termination is met, the loop will exit[17].

Because of this similarity in the way students understand programming languages and natural languages, one avenue to explore to help students better learn programming languages is to understand how students learn natural languages. Learning a second natural language in young adults is different than learning the first language as an infant. Second language acquisition research suggests that learning occurs in three steps[18]:

- Comprehensive input: acquiring and understanding something new in the second language, by means of listening or reading,
- Comprehensive output: producing something in the second language, by speaking or writing,
- Review or feedback: identifying errors and making changes to correct responses.

Both natural language learning and learning programming languages require the understanding of set of rules that define the language (grammar or syntax) and learning a body of words (vocabulary). Natural languages however can be ambiguous and imprecise, while programming language are required to be direct and precise in order to accomplish a specific task[19,20]. Significant literature is dedicated to developing new tools to assist novice programmers translate natural language into programming language[19-21]. There is also research on using programming languages to assist in learning second natural languages[22].

Historically, scholars have posed that learning programming languages can be predicted by one's mathematical aptitude[23,24]. However, recent studies suggest that neurocognitively, individual differences in learning programming more closely associated with language learning aptitude [25,26]. This implies that learning a new programming language may be very similar to learning a new natural language[26,27]. In fact, Chomsky's theory of formal languages that defines valid sentence building in natural languages has been a central tool in mathematics theory and programming languages[28].

Bilingualism and multilingualism have also been associated with brain functions that improve mathematical and scientific performance in classes[29]. However, it is unclear whether the neurocognitive effects of bilingualism that originates from bilingual parenting during childhood differs from intentional second language acquisition in adults.

If learning a programming language is truly like learning a second natural language, much can be drawn from natural language pedagogy to teach programming to young adults. Several studies have recommended and used natural language learning paradigms in programming classes[27,30,31], finding lower mental workload on students and fewer failing grades in student performance.

In this study, we explore the latter part in the context of learning of programming languages. The central purpose of the study is to explore whether young adults participating in a first-year engineering course focused on developing computational thinking skills perform better if they are learning or have learned a second natural language when compared to those who have not. More specifically the research question guiding this study is: Was there a significant difference in student performance in an introductory programming course between students who are learning or have learned a second natural language when compared to those who have not, when participation in a second or prior programming course is controlled for?

Methods

To answer the above research question, performance in a high enrollment first-year engineering course offered in the College of Engineering and Applied Science (CEAS) at the University of Cincinnati (UC) was considered. Three consecutive offerings of the course (Fall 2014, Fall 2015, and Fall 2016) were examined in relation to the students' prior or parallel participation in language courses other than the English courses taken as part of a typical first-year curriculum. Students' participation in other programming courses was also considered during the analysis. Detailed description of the methods follows in this section.

Context:

Engineering Models I was the first in a two-semester sequence of courses that were required for all first-year students in the CEAS from fall 2012 through spring 2018. This sequence of courses served two purposes: to introduce students to the computer as a tool for solving engineering problems and to provide context and applications for the mathematics and science material covered in other introductory STEM courses. In the Engineering Models I course, students were introduced to the computational package MATLAB® and shown how MATLAB® can be used as a tool when solving engineering problems and modeling physical processes. For example, when plotting various functions including exponentials, sinusoids, and damped sinusoidal functions, the emphasis is on how these functions can be used to model certain physical processes such as the charging or discharging of a capacitor, chemical reaction rates, and damped harmonic motion. Several weeks were spent developing the logical thinking and computing knowledge required to make full use of MATLAB®. Even while developing students' programming skills, the labs and homework assignments were closely tied to mathematics, science, and engineering applications. For example, students used loops to program iterative algorithms based on the Newtown-Raphson algorithm or Taylor Series to determine square roots and cube roots of numbers and the sine or cosine of angles, respectively.

Data Collection:

The data analyzed for this study originated from two primary sources. Data specifically related to performance in the Engineering Models I course were collected from the records of individual instructors that were submitted to the course coordinator at the end of the fall 2014, 2015, and 2016 semesters (the assignments were common across all sections each semester). These semesters were selected because there were only minor variations to the course structure and implementation over this time span. Additionally, the data all originated from the same four instructors, who worked closely together, which helps to minimize the impact of instructor variation in student performance. The data includes homework averages, worth 25%, lab assignment averages, worth 25%, quizzes, worth 10%, midterm, worth 20%, and final exam, worth 20% along with the final course average and course grade.

Demographic data and the data related to performance in natural language and other programming language courses were all collected from university records. Performance scores in natural and programming language courses were limited to only those courses taken prior to or concurrently with the first time a student took the Engineering Models I course. Once all of the data was collected, one member of the research team not involved in the analysis consolidated all of the data sources into a single set and deidentified the data.

The deidentified data contained information on student letter grades in second language courses in Arabic, American Sign Language, Chinese, English as a Second Language, French, German, Italian, Japanese, Latin, Russian, Spanish. In addition, information on student letter grades in six other programming courses were also available. For the purposes of this study, performances in these courses were categorized into participation/non-participation. If a student had AP credits or a grade for any of the language courses, they had a participant entry in natural language while those who did not were deemed a non-participant. Similar categorizations were made for the programming courses. For the dependent variables, performance in the Engineering Models I course was considered. Weighted average scores for the entire course were calculated, which included scores from homework assignments, quizzes, labs, midterm exams, and final exams. The same procedures were applied for all three cohorts.

In order to understand the population investigated in this study, a set of demographic data is shown in Tables 1, 2, and 3 below. The three cohorts were broken out by gender, race, and major. Additionally, there were student records that showed participation in Engineering Models I in two consecutive years. These entries were removed with the notion that scores of students who attended the course twice would be skewed because of the multiplicity of participation. In total 5 student entries were removed from the analysis.

Table 1: Demographics

	2014	**2015**	**2016**
Total	551	677	697
Gender			
Male	460	543	558
Female	91	134	139
Race			
White	447	545	541
Black or African American	10	24	26
Hispanic/Latino	9	19	26
Asian	20	19	31
Two or more races	10	17	28
Non-resident Alien	25	20	37
Other or Unknown	30	32	8

Table 2 summarizes the enrollment for each of the majors offered in CEAS who took the Engineering Models I course along with the DFW rate over the three semesters being considered. As can be seen, enrollment among engineering students was fairly stable across the three semesters, as was the DFW rate. Note that this data is for the overall enrollment, not just for the cohorts investigated in this study. Table 3 shows the distribution of majors for only those students included in the three cohorts used in this study.

Table 2: Overall Distribution of Majors

	Fall 2014		**Fall 2015**		**Fall 2016**	
Major	**Enrollment**	**DFW Rate**	**Enrollment**	**DFW Rate**	**Enrollment**	**DFW Rate**
AE	41	7.3%	26	3.8%	43	11.6%
ASE	95	5.3%	60	6.7%	81	2.5%
BME	66	0.0%	77	2.6%	62	1.6%
CE	81	8.6%	74	1.4%	87	8.0%
CHE	147	4.8%	140	3.6%	125	4.0%
CM	33	12.1%	32	18.8%	33	15.2%
CMPE	80	6.3%	79	5.1%	106	7.5%
CS	76	1.3%	89	3.4%	65	7.7%
EASE	73	19.2%	60	8.3%	74	18.9%
EE	74	6.8%	74	8.1%	88	10.2%
EET	22	13.6%	32	18.8%	39	10.3%
ENVE	24	4.2%	31	0.0%	42	7.1%
FEP	101	4.0%	109	4.6%	161	6.2%
ME	147	4.1%	123	4.1%	152	2.6%
MET	64	17.2%	91	11.0%	93	16.1%
Total	**1124**	**6.8%**	**1097**	**5.7%**	**1251**	**7.8%**

Table 3: Cohort Breakdown of Majors

	2014	**2015**	**2016**
AE	8	15	19
ASE	54	11	28
BME	17	35	37
CE	22	28	22
CHE	28	42	58
CM	15	22	10
CMPE	13	31	32
CS	44	45	63
EASE	10	7	7
EE	21	42	45
EET	8	13	18
ENVE	5	12	12
FEP	1	2	5
ME	123	112	83
MET	26	47	53

Data Analysis:
The demographic data (Table 1 above) indicate that the sample has characteristics of a typical large midwestern, R-1 institution. To begin the analysis, differences between each cohort were investigated to determine whether the cohorts could be combined or needed to be analyzed as separate data sets. One-way ANOVA was conducted on the data grouped by each cohort. However, the parametric assumptions were violated, so the decision was made on the basis of Kruskal-Wallis test. The test showed significant differences between the cohorts using a significance level of $\alpha = 0.05$ ($\chi^2(2) = 11.534$, $p = 0.003$). Thus, further analysis continued for each cohort separately. Descriptive statistics for each cohort are provided in Table 4 and Figure 2 below.

Table 4: Descriptive Statistics for each cohort

	Fall 2014	Fall 2015	Fall 2016
Mean	86.888	86.063	85.879
Std. Deviation	13.645	14.233	12.520
Minimum	0.607	0.000	2.500
Maximum	99.233	99.644	100.000

For each cohort, observations were divided into 4 groups:
- students who participated in a course to learn a second natural language at the university level;
- students who participated in a course to learn a different programming language at the university level;
- students who participated in courses to learn both a second natural language and a different programming language at the university level; and
- students who did not participate either type of course at the university level.

One-way ANOVA was conducted to investigate differences in the course average between the four groups.

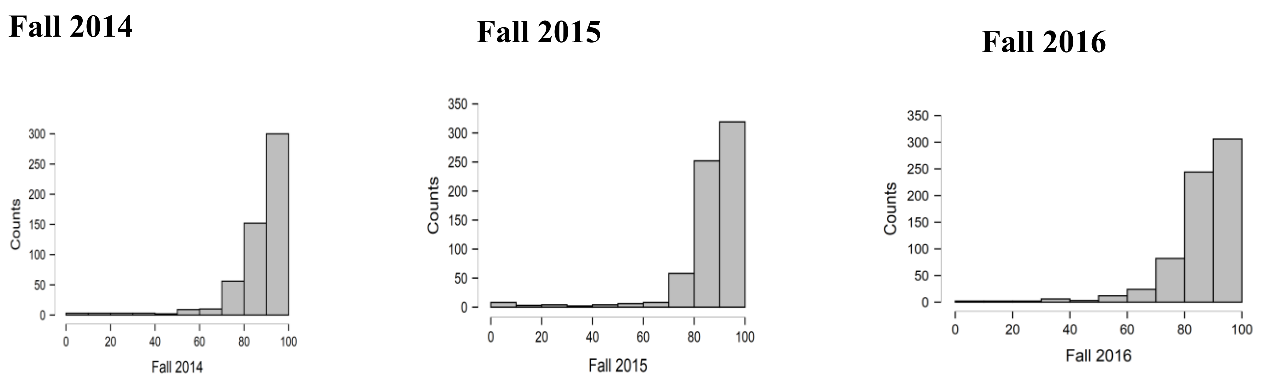

Figure 2: Distribution of course averages

Results

2014:

Since parametric assumptions were violated for the 2014 cohort, Kruskal-Wallis test was conducted to identify differences in course averages between each of the 4 groups. Table 5 below gives the course averages of the cohort by group of interest. Non-parametric Kruskal-Wallis test suggested no significant differences between the four groups (χ^2 (3) = 7.212, p = 0.065).

Table 5: Course average for different groups in 2014

Experience Group	Mean	SD	N
Natural Language only	89.934	8.773	64
No Programming or Natural Language	86.558	13.434	451
Programming & Natural Language	78.461	34.317	5
Programming only	86.708	20.953	21

2015:

Similar to the 2014 cohort, assumption violation prohibited a one-way ANOVA test and a Kruskal-Wallis test was conducted (see Table 6 below), which suggested significant differences between the groups (χ^2 (3) = 28.113, p < 0.001). Post hoc Dunn's test with Bonferroni corrections suggested that there were significant differences between the group pairs of *programming only* and *natural language only* (p = 0.003), and *programming only* and *no participation* in a second programming or natural language course (p < 0.001).

Table 6: Course average for different groups in 2015

Experience Group	Mean	SD	N
Natural Language Only	83.425	19.784	57
No Programming or Natural Language	85.981	13.615	571
Programming and Natural Language	92.430	5.468	4
Programming Only	91.440	12.894	32

2016:

Lastly, for the 2016 cohort, Kruskal-Wallis test revealed significant differences between the groups (χ^2 (3) = 31.218, p < 0.001). Post hoc Dunn's test with Bonferroni correction suggested significant differences between three group pairs: *programming only* and *natural language only* (p=0.023); *programming only* and *no participation* (programming or natural language) (p<0.001); *natural language* and *no participation* (p = 0.039). Table 7 below summarizes these results.

Table 7: Course average for different groups in 2016

Experience Groups	Mean	SD	N
Natural Language Only	87.999	11.605	79
No Programming or Natural Language	85.131	12.614	555
Programming & Natural Language	90.938	8.163	8
Programming Only	90.928	12.106	41

Discussion

Based on the results presented above, the answer to our original research questions is inconclusive. While there are significant differences for the 2015 and 2016 cohorts, they do not support a claim that learning a second natural language is linked to better performance in an initial programming course. In only one cohort (2016) was there a significant different between the *natural language only* group and the *no participation* group. While this one cohort does indicate that the natural language learners performed better in the Engineering Models I course, it is not enough to support a claim that learning a second natural language improves performance in learning a programming language.

The results do support a claim that experience learning a prior programming language does contribute towards performance in learning a second programming language. This can be seen in both the 2015 and 2016 cohorts, where the programming only group significantly outperformed the *no participation* group. Interestingly, the *programming only* group also outperformed the *natural language only* group in both 2015 and 2016. While learning a second natural language may provide some benefit, it appears the best way to develop proficiency with computing knowledge and skills is by actually programming.

It is also interesting to consider the reasons for the variation in performance and results among the different cohorts. As was stated previously, the semesters were selected specifically because they were the most stable across the six years the course sequence was offered. The four instructors who taught the sections of the course investigated were also common to all three semesters. One potential factor that may have contributed to the differences in the cohorts was the variation in the number of sections taught by each instructor. In 2014, instructors 1 and 4 taught two sections while instructors 2 and 3 taught three sections each. In 2015 and 2016, instructors 1 and 2 taught two sections each while instructor 3 taught three sections and instructor 4 taught four sections. Given that the 2015 and 2016 cohorts showed some similarity in results, this similarity in the number and distribution of sections among instructors may have played a role.

The overall performance in the course could also be a contributing factor to why there was little variation seen between the different groups across the cohorts. With the exception of the *programming and natural language* group from the Fall 2014 cohort, the averages for all other groups across the cohorts only ranged from the mid 80s to the lower 90s. Since these averages are fairly close and also relatively high, there may simply not be enough variation in the performance based on final grades to distinguish amongst the groups. The high grades could also indicate that students did not struggle significantly in the course, regardless of prior experience, due to the design and implementation of the course.

Another potential reason for the differences seen between the cohorts is the distribution of majors within the cohorts. For instance, in 2016, there were a larger percentage of computer science students in the sections of the course included than in 2014 or 2015. There was also a larger number of chemical engineering students, which tends to attract some of the higher performing students. A similar difference was also seen between the 2014 and 2015 cohorts, with increased percentages of biomedical, chemical, computer, and electrical engineers. A potential future direction for this study is to look specifically at the differences between the various engineering disciplines.

Conclusion

This study investigated the impact of learning a second natural language on performance in a specific programming language course (Engineering Models I). Results of the data analysis were inconclusive, with different cohorts potentially suggesting different relationships, or even no relationships at all. These differences between cohorts could potentially be a result of differences in the distribution of sections among instructors or the distribution of majors among the students included in each cohort.

There are several directions future work may take. One pathway is to consider whether bilingualism (from childhood) has an impact on performance. Along these same lines, an investigation into the type of natural language learned could also be illuminative. Every language has a different vocabulary and set of grammar rules. There are also differences in the symbology used. While many languages utilize the Latin alphabet common in most wester cultures, other languages use alternative alphabets, such as the Greek alphabet or the Cyrillic alphabet. Other languages utilize symbols to represent entire words or ideas, such as the cuneiform of many Asian countries. These differences may have a significant impact on the way in which someone learns a programming language, which typically utilizes a Latin alphabet and English terms, and warrants investigation.

While many would equate proficiency in programming with computational or mathematical skills, that is only one part of what is needed to be create a functioning program to solve a particular problem. Additionally, a programmer must be able to think through the logic necessary to carry out the various tasks the program must perform and translate the problem description, typically given in a narrative form. Another future avenue to pursue is the link between someone's language abilities and the ability to convert the description of a problem into the logic necessary to solve it.

As was mentioned previously, an investigation into the ways that different majors experience learning to program could also help to better understand the differences seen in the cohorts for this study. Are there differences in performance for students in the same groups (second natural language, prior programming language, no prior experience) based on major? Is a computer science student typically going to perform better than other disciplines, even without prior experience because of a natural inclination to computing or identification with the discipline? Does a biomedical engineering student with experience in a second natural language perform better than an electrical engineer with similar experience, and what is it about the students typically attracted to that major that causes such differences? These questions and others along the same lines could help instructors understand their students better and develop more robust pedagogies for engaging all students in the act of learning to program.

Finally, only participation in a second programming language course and second natural language course was considered for the purposes of this study. Future work may examine whether performance in these courses is related to performance in the Engineering Models I course.

References

1. National Academy of Engineering, *The engineer of 2020 : visions of engineering in the new century*. 2004, Washington, DC: National Academies Press. xv, 101 p.
2. Robins, A., J. Rountree, and N. Rountree, *Learning and teaching programming: a review and discussion.* Computer Science Education, 2003. **13**(2): p. 137-172.
3. Mayer, R.E., *Cognitive aspects of learning and using a programming language*, in *Interfacing Thought*, J.M. Carrol, Editor. 1987, The MIT Press: Cambridge, MA. p. 61-79.
4. McCracken, M., et al., *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students.* ACM SIGCSE Bulletin, 2001. **33**(4): p. 125-180.
5. Thomas, L., et al., *Learning styles and performance in the introductory programming sequence.* ACM SIGCSE Bulletin, 2002. **34**(1): p. 33-37.
6. Newell, A. and H. A. Simon, *Human problem solving*. Englewood Cliffs, NJ, Prentice-Hall, 1972.
7. Bucks, G., *A Phenomenographic study of the ways of understanding conditional and repetition structures in computer programming languages.* PhD Dissertation, Purdue University, 2010.
8. Johnson-Laird, P. N., *Mental models: towards a cognitive science of language, inference, and consciousness*. Cambridge, MA, Harvard University Press, 1983.
9. Bransford, J. D., A. L. Brown, et al., Eds., *How people learn: brain, mind, experience, and school*. Washington, D.C., National Academies Press, 1999.
10. Felder, R.M. and L.K. Silverman, *Learning and teaching styles in engineering education.* Engineering Education, 1988. **78**(7): p. 674-681.
11. Felder, R.M. and J. Spurlin, *Applications, reliability and validity of the index of learning styles.* International Journal of Engineering Education, 2005. **21**(1): p. 103-12.
12. Rosati, P.A. *The learning preferences of engineering students from two perspectives*. in *ASEE/IEEE Frontiers in Education*. 1998. Tempe, Arizona.
13. Litzinger, T.A., et al., *A psychometric study of the index of learning styles.* Journal of Engineering Education, 2007. **96**(4): p. 309-319.
14. Zualkernan, I.A., J. Allert, and G.Z. Qadah, *Learning styles of computer programming students: a middle eastern and American comparison.* IEEE Transactions on Education, 2006. **49**(4): p. 443-50.
15. Felder, R.M., *Learning and teaching styles in foreign and second language education.* Foreign Language Annals, 1995. **28**(1): p. 21-31.
16. Ma, L., et al., *Investigating the viability of mental models held by novice programmers.* ACM SIGCSE Bulletin, 2007. **39**(1): p. 499-503.
17. Bonar, J. and E. Soloway. *Uncovering principles of novice programming*. in *Proceedings of the 10th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 1983. Austin, Texas: ACM.
18. Brown, H. D., & Gonzo, S. T. *Readings on second language acquisition*. Englewood Cliffs, NJ: Prentice Hall Regents, 1995
19. Barmpoutis, A. *Learning programming languages as shortcuts to natural language token replacements*. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research.* 2018. Association for Computing Machinery, New York, NY, USA, Article 1, 1–10. DOI:https://doi.org/10.1145/3279720.3279721

20. Liu, X., and Wu, D. *From Natural Language to Programming Language*. In Goschnick, S. (Ed.*), Innovative Methods, User-Friendly Tools, Coding, and Design Approaches in People-Oriented Programming* (pp. 110-130). IGI Global. 2018. http://doi:10.4018/978-1-5225-5969-6.ch004

21. Good, J. and Howland, K. *Programming language, natural language? Supporting the diverse computational activities of novice programmers*. Journal of Visual Languages & Computing, 2006. **39:** p. 78-92, https://doi.org/10.1016/j.jvlc.2016.10.008.

22. Moreno-León, J. and Robles, G. *Computer programming as an educational tool in the English classroom a preliminary study*. In IEEE Global Engineering Education Conference (EDUCON) 2015. p. 961–966.

23. Sauter, V. *Predicting computer programming skill*. Computers & Education, 1986. **10(2**): p. 299-302.

24. Erdogan, Y., Aydin, E., & Kabaca, T. *Exploring the psychological predictors of programming achievement*. Journal of Instructional Psychology, 2008. ***35*(3)**.

25. Prat, C., Madhyastha, T., Mottarella, M., & Kuo, C.-H. *Relating natural language aptitude to individual differences in learning programming languages*. Science Reports, 2020. ***10*(3817).** doi: https://doi.org/10.1038/s41598-020-60661-8.

26. Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethman, A., Leich, T., . . . Brechmann, A. *Understanding understanding source code with functional magnetic resonance imaging*. ICSE 2014: Proceedings of the 36th International Conference on Software Engineering, 2014 (pp. 378-389). Hyderababad, India.

27. Frederick, C., Sun, L., Liron, C., Verleger, M. A., Cunningham, R. M., & Espejo, P. S. *Implementation and evaluation of a second language acquisition–based programming course*. Proceedings of the ASEE Annual Conference & Exposition, 2016.

28. Ragonis, N., & Shilo. G. *Drawing analogies between logic programming and natural language argumentation texts to scaffold learners' understanding*. Journal of Information Technology Education Research, 2014. **13**: p. 73- 89.

29. Stocco, A., & Prat, C. S. *Bilingualism trains specific brain circuits involved in flexible rule selection and application*. Brain and Language, *2014.* **137**: p. 50-61. doi: https://doi.org/10.1016/j.bandl.2014.07.005

30. Sun, L., Frederick, C., Liron, C., Ding, L., Gu, L., II, A. C., . . . Espejo, P. S. *Motivating students to learn a programming language: Applying a second Language acquisition approach in a blended learning environment*. 2018 ASEE Annual Conference & Exposition. Salt Lake City, UT.

31. Van Roy, P., Armstrong, J., Flatt, M., & Magnusson, B. *The role of language paradigms in teaching programming*. 34th SIGCSE technical symposium on Computer science education, 2003. p. 269-270. Reno, Nevada.