Incorporating Practical Computing Skills Into a Supplemental CS2 Problem Solving Course

Margaret Ellis
Virginia Tech

Stephen H. Edwards
Virginia Tech

Clifford A. Shaffer
Virginia Tech

Catherine T. Amelink
Virginia Tech

Computer scientists are often expected to obtain practical technical skills on their own. We have updated our supplemental CS2 problem-solving course to introduce students to technical skills across a variety of computer science topics in order to expand their incidental skills that often fall through the cracks. The goal of the course is for students to feel comfortable taking their problem-solving skills to unfamiliar computing situations. This course helps bridge the gap for students with less computing experience by introducing them to current computer science tools and demystifying potentially intimidating topics such as version control, security, command line tools, web development, and machine learning. This experience report details our motivation and approaches for this contemporary problem-solving course. We discuss outcomes regarding student perceptions of the course, and our innovative approach of measuring student comfort in situations that require solving problems with computer science such as undergraduate research, hackathons, and personal programming projects.

Keywords: problem-solving, skills, diversity, co-curricular, CS2

INTRODUCTION

Upper-level CS courses, internships, and research often require students to employ practical technical skills. At our institution students may not be taught these topics in their courses and they are often expected to learn such skills on their own. Students with less pre-college or extracurricular computing experiences may have fewer of these practical skills and less experience learning on their own. Such students may be at an academic disadvantage or feel intimidated (Amelink et al., 2018; Alvarado et al., 2018). Realizing this gap motivated us to revise a key introductory course in our curriculum. CS2104 Problem Solving in Computer Science teaches students problem-solving skills in conjunction with an introduction to various

practical computer science topics such as using version control, developing websites, navigating networks, and interacting with databases. The intention of the revision is to:

1. provide practical technical skills needed in a variety of computer science settings,
2. improve problem-solving skills by providing practice within meaningful context, and
3. increase student comfort in situations that require solving problems with computer science

In this experience report we detail how our teaching experiences and related research motivated our course evolution. We consider problem solving, student experience, and technical skills with an emphasis on the relationship between practical skill acquisition and participation in computing activities outside of students' coursework. The course modules give students a sampling of computer science topics with common problem-solving heuristics connecting them. Our instructional strategies involve a hybrid of teacher-directed and active learning approaches. Our intention is that this intervention will improve students' problem-solving skills, give them an opportunity to acquire contemporary technical skills early in their education, and prepare them to apply these skills in future courses and beyond the classroom.

To help us determine students' perceptions of the course, their learning of technical and problem-solving skills, and their comfort in applying these skills outside of their coursework, we administered pre and post surveys with multiple choice, Likert scale, and open-ended questions. In our open-ended student responses, many students mention a positive impact on their CS experience including acquisition of new problem-solving and technical skills. Overall, students find the course useful and interesting and report becoming more comfortable participating in computing activities that they encounter outside of the course.

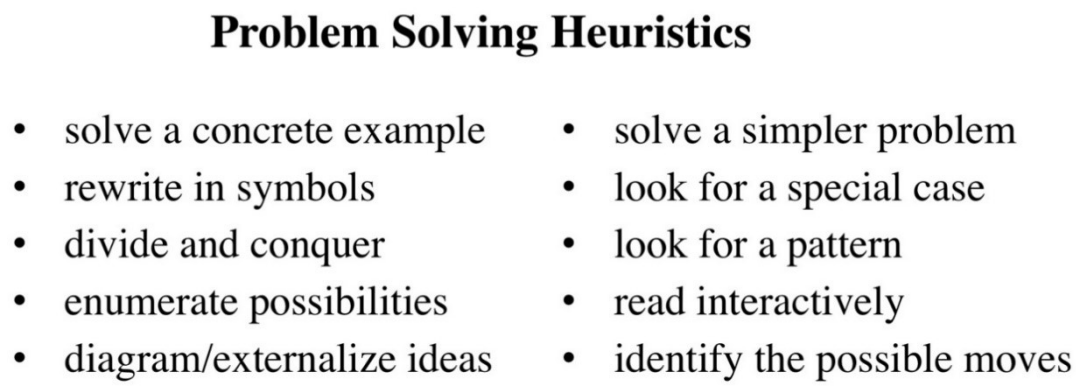
BACKGROUND

Computer science educators have long been interested in improving students' problem-solving skills (Hasni & Lodi, 2011; McCartney et al., 2007). There is recent interest in providing scaffolding to improve students' metacognition and self-awareness around problem-solving steps and strategies in computer science (Loksa et al., 2016; Loksa & Ko, 2016; Prather et al., 2019). For more than a decade our CS Department has been teaching a first-year experience course intended to help build students' problem-solving and metacognitive skills. From senior exit interviews our department found some students did not feel the course was useful or relevant and that it overlapped with various other courses they had. When students find course material useful and interesting it improves their motivation (Jones et al., 2016). Earlier iterations of the course emphasized problem-solving heuristics in the context of classic, but more abstract, logic puzzles and math problems. We redesigned our problem-solving course to emphasize the application of the problem-solving heuristics shown in Figure 1 across contemporary topics using practical skills. While Loksa et al. use an explicit strategy of coaching students through six problem-solving strategies for programming problems, we recommend similar strategies to our students, but more broadly and across various types of problems encountered in Computer Science (Loksa & Ko, 2016; Prather et al., 2019). Our goal is to improve problem-solving skills by applying them in the meaningful context of solving technical problems in general, not just in programming.

We are interested in promoting a welcoming culture in our department. Previously in 2016, in focus groups with students about the factors impacting career interest in computer science, some of our students reported perceptions that could negatively impact their experience in our program. Students reported feeling intimidated, needing to teach themselves material required for courses, and that there was distance between students who had computing as a hobby and those who did not (Amelink et al., 2018). Students identified tinkering and previous experience as an important part of feeling successful in computer science. We are motivated to build students' confidence and help them persist in the field. We are inspired by previous work demonstrating that students' sense of CS identity, belonging, and self-efficacy is correlated with success, and that tinkering and skill-building can improve these feelings (Lewis, 2017; Lewis et al., 2011; Rittmayer & Beier, 2009; Roick & Ringeisen, 2017). We consider students' comfort in a situation to be an amalgamation of feelings and former experiences. Our institution and others offer various courses that emphasize the use of practical skills such as an elective capstone and software engineering courses, undergraduate research programs, and course credit for participation in the programming competition team

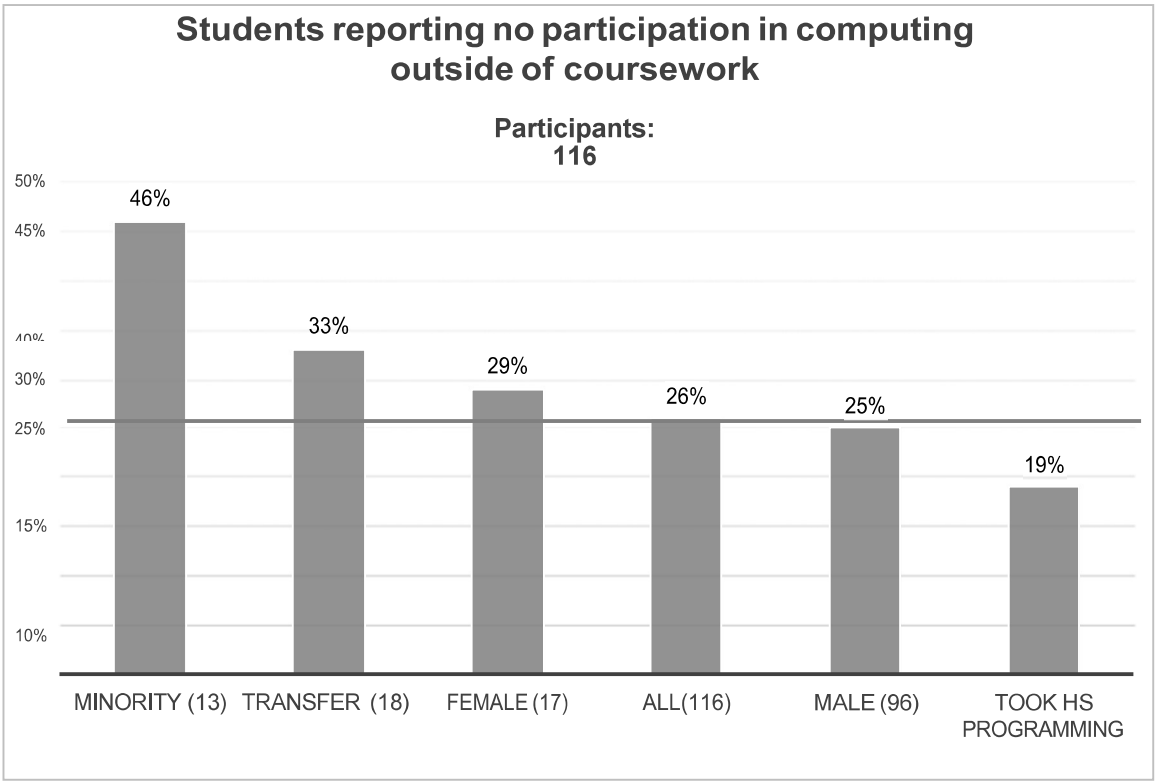
(Davis & Rebelsky, 2019; Iberman, 2017; Koppelman et al., 2011; Linhoff & Settle, 2009; Mohan et al., 2012). This course is designed to help prepare students for such courses and similar co-curricular activities. We aim to provide early course experiences that help students feel comfortable in the field of computer science.

FIGURE 1
LIST OF COMMON PROBLEM-SOLVING HEURISTICS REFERENCED IN PROBLEM SOLVING IN COMPUTER SCIENCE COURSE



Some students have more technical experience, especially with mechanical tools of the trade. Undergraduates often encounter topics such as version control, web development, regular expressions, or databases for the first time during an internship or as an incidental requirement in one class. In the first semester of the new version of our course we witnessed student enthusiasm for using Chrome Developer tools (Google, 2019). The context was jeopardy-style Capture-the-Flag (CTF) which is a cybersecurity competition where students find hidden flags online, or in provided files, for varying point values in categories such as reconnaissance, web, cryptography, and networking. Later we discovered that a group of students, predominantly females, subsequently traveled to another university to compete in their first co-curricular CTF. We also cooperated with the student GitHub organization, who held workshops in conjunction with our class. Two students asked to make a class announcement about a Web Development club meeting they intentionally scheduled to be in sync with the course. At this point it was clear that there was cross-pollination between this course and student co-curriculars. We began wondering more specifically about student co-curricular activities and the potential relationship with this course, so we began collecting data related to co-curriculars. As shown in Figure 2, 74% of these students had participated in some form of external computing, but this leaves a notable subset of 26% who have not. In our small sample, females and students from underrepresented racial and ethnic minorities such as Black/African American and Hispanics had less participation in, and more apprehension about, participating in certain external computing experiences. Previous work has demonstrated the benefits of co-curriculars and there is evidence that students who participate in them are higher achieving (Frieze & Blum, 2002; Gersting & Young, 1998; Iberman, 2011; Mandernach, 2016; Nandi & Mandernach, 2016; Thompson, 2006). We would like to bring these benefits into our course to provide more access to all students. This reinforces our desire to provide practical technical skills needed in a variety of computer science settings early within the curriculum.

FIGURE 2
STUDENT RATE OF PARTICIPATION IN EXTERNAL COMPUTING ACTIVITIES IN PRE-SURVEY 2019



COURSE DESIGN

As shown in Table 1, we redesigned our Problem Solving course. It was formerly puzzle-based. The redesign had more emphasis on topics in Computer Science, and hands-on use of technologies. Each semester three sections of this course are offered with total enrollment varying between 115 to 255, with this number expected to rise. This course has CS1 as a prerequisite so we expect students to have some programming experience. We chose to use python in this course, as it is easy-to-use across many topics. It is a bonus that this allows us to introduce a language that is otherwise not a standard part of our lower-division curriculum.

Our experience working with undergraduate research students helped inform our decisions about technologies and approaches for this course. Mentoring undergraduate research students as they learned python, git, and SQL helped reinforce the need for these topics in our curriculum and provided an opportunity to try various tutorials, IDEs, and instructional approaches. In the summer of 2018, we hired two Undergraduate Teaching Assistants to specifically help with course development. These UTAs vetted tutorials, tools, and activities, and provided valuable feedback and insight. Working with students as TAs, in research, and in other co-curricular settings, further informs our desire to provide hands-on collaborative learning experiences in the classroom. We strive to create a supportive classroom environment where students can learn from each other and tinker together to get exposed to new technologies, tools, and tricks of the trade as is often the case in co-curriculars (Frieze & Blum, 2002; Thompson, 2006).

TABLE 1
COMPUTER SCIENCE TOPIC MODULES

MODULES	Problem Solving Introduction	Algorithms	Software Engineering Process	Networking and Web Development	Databases and Machine Learning
TUTORIALS	Runestone Python	Codecademy (bash, git, HTML, CSS)			SQL Bolt
TOOLS	PyDev	hand trace, leetcode	bash, git, balsamiq	Wireshark, HTML/CSS, Google Cloud Platform	SQL, sklearn
RESOURCES	Virginia Cyber Range, CS Field Guide, You Tube				

Topic Selection

When considering which topics to add to the course, we thought about:

1. skills that are useful during internships, undergraduate research, and upper-level courses
2. topics meaningful to students relative to career aspirations and personal use of technology, and
3. areas of computing that students likely find attractive from mainstream media.

Familiarity with regular expressions, SQL, web development, and network fundamentals are useful in industry and undergraduate projects and so were key skills we wanted to include. Students are interested in algorithms, software engineering practices, and interface design because they perceive them as potential career paths. Students are excited to build a website because they use them so regularly and they can easily share their creation with family and friends. Students are motivated to learn about popular topics from media attention, such as cybersecurity and machine learning. When students recognize the usefulness of course material it provides a natural motivating framework for learning and practicing problem-solving skills (Jones et al., 2016). Students are mature enough with a CS1 background to get exposure to the many areas of computer science before they have the opportunity and background to take upper-level courses dedicated to a specific topic.

The topics and approach in this course could be adopted in a variety of course settings. A subset of the topics could be introduced in a first-year experience course, as part of a traditional programming course, or in a supplemental course designed to introduce students to python. The material in most of the modules can be adapted to not require CS1 as a prerequisite. In the spirit of the motivation for this course, if a module is used outside of the sequence of this course it is important that an instructor ensure students have prerequisite skills as needed. Most of these modules also map easily to the first few weeks of an upper-level course on the corresponding topic.

Modules

As described above we selected topics expected to be perceived as useful and interesting to students. We sequenced the topics to best address perquisites. Below we provide further details on the technologies and content for each of these topics.

Problem Solving Introduction Module

In this first module the students learn basic python with the Runestone Python tutorial and we introduce the problem-solving heuristics shown in Figure 1. We also highlight mindset and metacognitive skills in order to improve student awareness of their problem-solving approaches. Students have problem-solving assignments that align with the CS Field Guide chapters on encoding and encryption. In these first weeks,

students participate in two jeopardy-style Capture-the-Flag in-class competitions. We use the Web and Reconnaissance style questions which require only a quick introduction to Chrome Developer Tools to view HTML source and a few tips about finding flags. These CTF exercises are good practice in persistence and teamwork. Once students have completed the python tutorial they set up and install the PyDev plugin for eclipse. We chose Eclipse because it is used in our traditional CS2 course and so would reduce the learning curve for the students (The Eclipse Foundation, 2019; Google, 2019; pydev, 2019; Runestone Interactive, 2019; University of Canterbury Computer Science Education Research Group, 2020).

Algorithms Module

The Algorithms Module is the most traditional portion of the course and is similar to the material used previously. We review the concept of algorithm and the various ways to represent an algorithm. The algorithms content that was formerly in pseudocode is now taught with python. During this portion of the course students have programming homework assignments to practice figuring out list algorithms, learn to strive for efficiency, and obtain more experience with recursion. For divide and conquer algorithms and dynamic programming the students learn to hand trace. Many students recognize these problems as interview preparation and are engaged. We spend one day using LeetCode in small groups to help students realize online coding challenges are available and that they have the skills to be successful at them. During the Algorithms Module students are doing Codecademy tutorials on bash and git to prepare for the Software Engineering Module (Codeacademy, 2019; Free Software Foundation, 2019; LeetCode, 2019).

Software Engineering Process Module

Students extend from programming in the Algorithms Module to software engineering concepts and using version control for the algorithms they have programmed. During the Software Engineering Module we address the classic phases of requirements gathering, testing, design, and coding, and touch on agile development. We discuss the problems involved with project management and strategies for coping with team programming. Students then learn about the interdisciplinary nature of Human Computer Interaction. They work in teams using balsamiq to build a prototype for a presented scenario (balsamiq, 2019).

Networking and Web Development Module

After designing interfaces, we take a sharp turn from a high-level to a low-level computing topic, networking. Students have homework problems to solve in conjunction with reading about networking in the CS Field guide. They are also expected to watch several videos about networking and the internet. There is a natural connection back to the first weeks of the course when we studied encoding and cryptography. We spend several days in class using the command line for networking and getting introductory exposure to utilities like traceroute, scripting, and secure shell remote login. Command line use helps bridge the skills gap for students with no external computing experience as it helps prepare them for using Linux in subsequent courses. We also spend a day learning about packet sniffing and using Wireshark. As the students acquire more skills we are able to do additional CTF class activities, which are rich in problem solving. We then go deeper into the HTTP protocol and sever-client relationships to learn the foundations of web development. After completing tutorials on HTML and CSS, the students put these pieces into practice using a set of skeleton files to incrementally learn web development concepts on Google Cloud Platform (Computer Science Education Research Group at the University of Canterbury, New Zealand, 2019; Google Cloud Platform, 2019; Wireshark, 2019).

Data Science and Machine Learning Module

Students have a basic idea about networking and servers which helps them understand the various possible configurations for database servers. After lectures on database concepts and working through a SQL Bolt tutorial, students can write basic SQL statements in python using a local sqlite3 database. SQLBrowser gives students a usable GUI to help them grasp database concepts. We follow up by working with data in pandas and matplotlib so students get brief exposure to programming with statistics and graphs. Next, we incorporate sklearn so students can execute and explore the results of machine learning

algorithms. To prepare for machine learning content students watch bots videos and they are also assigned some ethics reflection prompts in response to Cathy O'Neil's TED Talk (Auth0, 2020; Grey, 2019; NumFOCUS, 2019; O'Neil, 2017; sklearn, 2019; SQLBolt, 2019).

The common thread across topics is the problem-solving heuristics shown in Figure 1. We introduce these early on and revisit them with each topic and explicitly point out when we are using a strategy, or trying several of them, to solve a problem. For example, we point out the use of concrete examples for solving encoding problems, developing algorithms, and initially using hard-coded values in incremental web development. Another example is how students are exposed to diagramming when creating a flow chart for software and systems engineering, designing a user interface, and modeling entity relationships for database design. They repeatedly see divide and conquer. It is first introduced with binary search and reappears repeatedly such as with the concepts of parallel computing, network layers, and the distribution of front end and server-side responsibilities in web development. These applications of problem-solving strategies across topics in computer science are interesting and engaging for the students.

Instructional Approaches

We incorporate scaffolding to guide students through discovering solutions by working concrete examples and generalizing to find an algorithm, and we fade prompts throughout the semester. We also provide many smaller-sized problems to solve, with practice both in and out of class, with and without partners. Students are commonly assigned videos to watch and tutorials to guide them in self-directed learning in preparation for course topics. We have announced and unannounced quizzes on such material. Students feel a sense of urgency to complete work during class but the stakes are relatively low as they usually earn only participation credit in class.

Experience with, and coaching about, self-directed learning helps prepare students for likely future situations when they will need to learn an unfamiliar technology (McCartney et al., 2007; McCartney et al., 2016). Undergraduate Teaching Assistants attend classes and hold office hours. They act as a role models by building relationships with the students in the hands-on environments. A graduate teaching assistant helps with grading and also engages with the students in office hours. The instructional team regularly meets to be well prepared for assignments and to improve clarity and consistency across the course. The teaching style varies between lecturer-style and facilitator-style so that students gain some teacher-directed knowledge as a foundation but also experience guided practice with peers in class (Grow, 1991). We regularly tell the students they are not expected to have previous experience other than CS1 and that there will be a wide range of backgrounds in the room. We also let them know that they will likely enjoy certain topics more than others. Many of these approaches intersect with the aforementioned implications for teaching determined by studying the benefits of co-curriculars in computing (Frieze & Blum, 2002; Gersting & Young, 1998; Iberman, 2011; Nandi & Mandernach, 2016; Thompson, 2006).

Our day-by-day course schedule and sample activities can be found at http://people.cs.vt.edu/maellis1/teaching resources/.

RESULTS

Anecdotally, through student assessment, and via targeted survey questions, we have found that we accomplished our initial goals for the new version of this problem-solving course. We know the students obtained practical technical skills and that they improved their problem-solving skills. They are motivated to engage in problem solving when it is hands-on with useful technology. Overall students increased their comfort in situations that require solving problems with computer science.

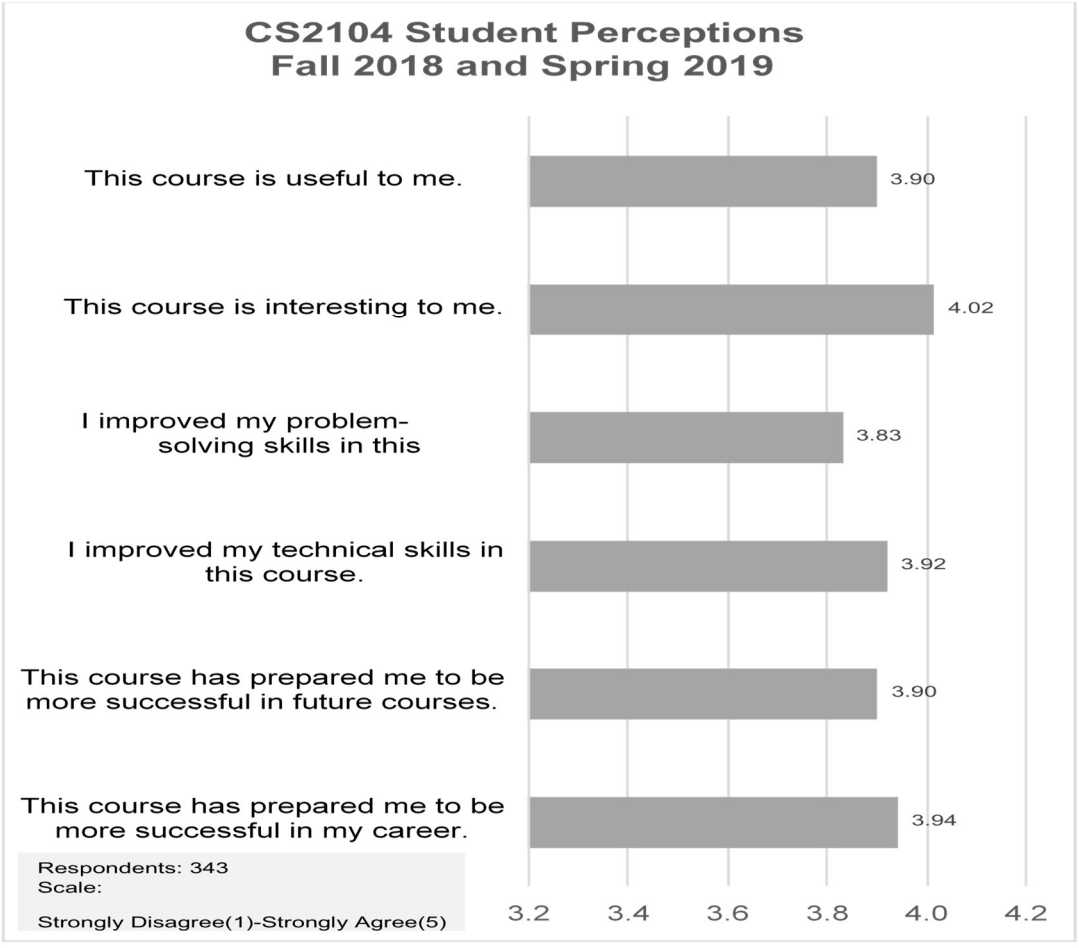
In the first two semesters we collected post-survey data from 324 students (220 in Fall 2018 and 104 in Spring 2019) about their perceptions of the course with regard to usefulness, interest, and acquisition of both problem-solving and technical skills, the averages are shown in Figure 3. In contrast to former feedback about the previous version of the course, students found the course useful and interesting which helped with their engagement and motivation. They reported improving their problem-solving skills and learning new technical skills. As shown in Figure 3, students also reported improved preparedness for

pursuing future courses and a career in computer science. We would like to follow up with students in their senior year to learn more about whether they have the impression that the course positively impacted their subsequent computing experiences. We observed that the response averages from underrepresented minorities appeared to consistently be slightly higher than the overall class average, and the response averages from females appeared to consistently be slightly lower than the overall class averages. We would like to determine if the pattern continues and if so, attempt to understand any causes we could possibly affect.

Students consistently report that their eyes have been opened to the breadth of computer science and they have acquired more skills. For example, these responses reflect students' interest in the course and its perceived usefulness:

> *Focuses on connecting different topics throughout the semester which I really like (they don't all feel disconnected and random topics). Not to mention, the topics are all very interesting and helpful information to learn for future careers/classes.*
>
> *This class was really useful to me simply because it showed off all the different areas of CS that I didn't even knew existed. I feel like this course has given my academic career here at tech a much more defined direction than my previous goal of "take random classes and don't fail."*

FIGURE 3
STUDENT PERCEPTIONS OF PROBLEM SOLVING IN COMPUTER SCIENCE COURSE



Different students emphasize the benefits of different portions of the course. In each section there seems to be a few students who wish we would dive deeper into the topics, although we do try to be very clear that we are introducing concepts that they can study more deeply in the future.

Students provided detailed positive descriptions of the course in response to the prompt "You indicated that this course has or will influence your computer science activities outside of your coursework, please explain.":

> *I learned a lot of cool stuff in this class, especially regarding git and web development. I was able to use the google cloud platform project to my advantage during an interview I had with salesforce.*
>
> *I know more about computer science as a field because of what I've learned in this class, and I am able to have more informed discussions about issues in CS*

We were pleased to see that even with seemingly more exciting technical content in the course, students reflected on the problem-solving skills they acquired:

> *In other cs courses, I faced many situation where I was not sure how to come up with a solution. Every time I faced that, I take a break, and break the problem into several pieces, so that the simple case passes, and keep did it until the problem was solved.*
>
> *I have learned how to tackle problems that are difficult to me. When I am stuck, I feel as though from taking this class that I know what steps to take to solve the problem*

To evaluate in more detail the extent to which the interventions used in the course helped build confidence in key areas among students we conducted a pre/post assessment. The assessment included an online survey that was administered to students. We asked students to rate their apprehension or comfort level on a four point scale (1=Apprehensive, 2=Somewhat Apprehensive, 3=Somewhat Comfortable, 4=Comfortable) on a variety of activities. The activities included personal programming projects, programming projects with friends, hackathons, CTFs, and undergraduate research. The pre-survey was administered during the first week of class and the post-survey was administered during the last week of class in the Spring 2019 semester. Additional questions on the survey asked demographic data including race and gender. These questions were used to parse the data to see whether the rate of change in apprehension was different for underrepresented groups in computer science. We looked at the trends in female apprehension. In addition, we examined the trends in apprehension among underrepresented students by grouping students that reported themselves as African Americans/Black, Hispanic, Native American or Two or More Races. In total 104 students responded to the survey for a 90% response rate.

In order to analyze the change in levels of apprehension, the Apprehensive/Somewhat Apprehensive responses were combined into one category "Apprehensive" while the Comfortable/Somewhat Comfortable response items were combined into one category "Comfortable." The percentage change was then calculated between responses in the students that reported Apprehensive versus Comfortable on the pre/post survey, see Figure 4.

Students' open-ended responses supported the apprehension survey data. Overall students became more comfortable participating in computing activities beyond the classroom. The technical practice was immediately useful for some students as described below:

> *It has shown me how to use python and wireshark and even do some web dev. I plan to make my own website using some of this knowledge. I also appreciate learning so much material that I never even knew, now CS is not as intimidating*

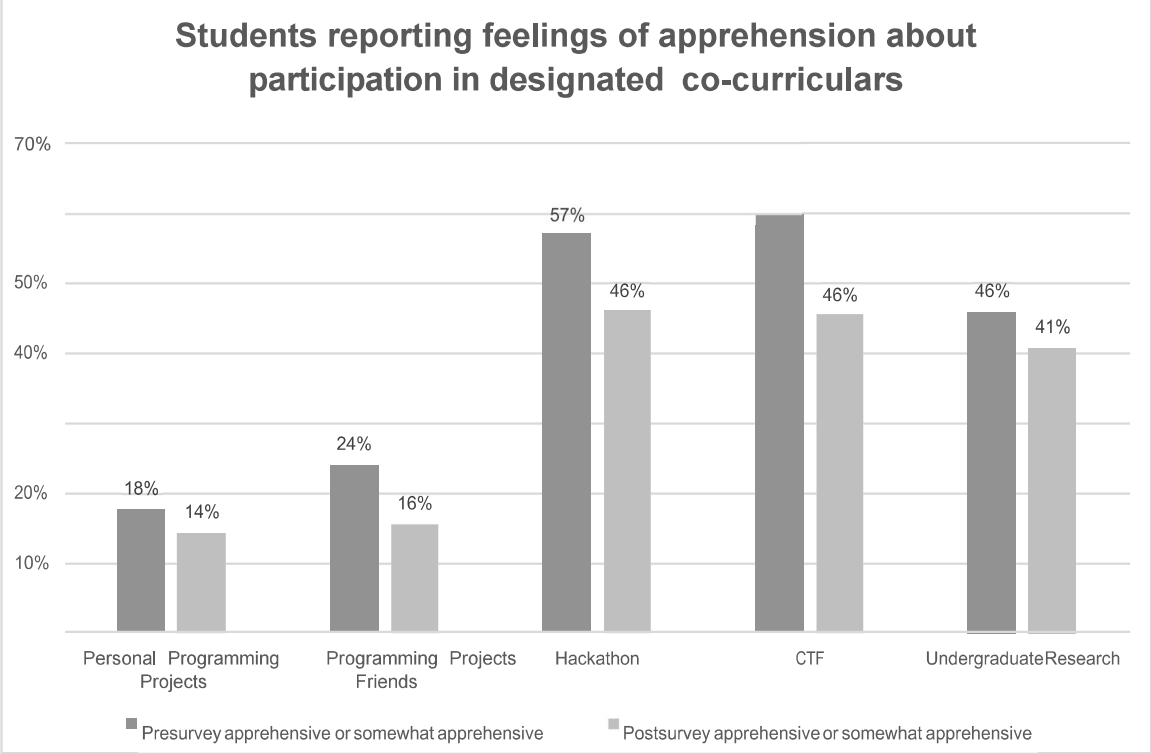
Learned python and got experience using libraries that are crazy, and it helped refamiliarize me with git & git bash. I actually contributed to a GitHub repository because I felt somewhat confident. Been a few weeks now and there aren't any comments bashing it! :D

I have joined more CS activities like the Web Development Club and AutoDrive [an autonomous vehicle club] which got me more involved in CS outside of school

The organization was great and the course content was awesome. I loved how we got to learn about seemingly distant CS topics such as networking, machine learning, web dev, and more! I now feel confident and ready to pursue any CS related task on my own in my personal projects

There were some notable differences in the percent change in apprehension among Underrepresented Minorities (URMs) including Black and Hispanic students. In some cases their apprehension declined over the course of the semester and at a greater rate than trends seen overall among students, but in some activities apprehension increased. Likewise, female students showed variation from the overall class trends. Both of these populations are small (see Figure 2) and we plan to continue collecting data to see if these variations continue in future semesters and across a larger dataset.

FIGURE 4
STUDENT FEELIGNS OF APPREHENSION ABOUT PARTICIPATION IN CO_CURRICULAR ACTIVITIES AT THE BEGINNING AND END OF THE SEMESTER



We are pleased that students became less apprehensive about the co-curriculars we questioned them about. We are concerned and curious about why some students became more apprehensive. We plan to collect more data across multiple semesters to see if there is a consistent pattern, and to also perform a

qualitative analysis regarding these feelings in order to help us improve our instructional approaches. For example, the in-class CTF mimicked a co-curricular activity, and students were in groups and graded only on participation. This situation may be more encouraging to students than the outside individual networking programming project which possibly increased some students' apprehension about personal projects.

FUTURE PLANS

Looking ahead we expect to gather larger volumes of data so we can perform a more complete analysis, particularly with regard to student feelings about and participation in external computing activities. We expect to ask more open-ended questions to further understand student perceptions. We would like to determine whether some subsets of students became more apprehensive about some co-curricular activities and try to understand why and how to adjust our instructional approaches to affect this.

We will also plan to further investigate the relationships between participation in co-curricular activities and student success, especially across demographics, possibly using a variation of the Center for Evaluation the Research Pipeline (Center for Evaluating Research Pipeline (CERP) surveys (Computing Research Association Evaluation, 2020). We are also interested in various distinctions in co-curricular activities such as those that are competitive vs. noncompetitive. We would like to explore additional approaches for integrating the benefits of co-curricular activities directly into the curriculum with hopes to reach, attract and retain a broader population of students.

We are also interested in supplemental CS2 courses in general. We plan to continue to investigate methods to provide early exposure to current topics (Bressoud & Thomas, 2019). We are also curious about potential approaches to integrate various cross-cutting areas of computer science such as design, ethics, security, and data science throughout the curriculum.

CONCLUSION

The course uses a variety of problem-solving strategies and encourages practice, exploration, and tinkering to increase students' comfort not only with their technical skills but also with their ability to acquire new skills. Students can thus approach future coursework, research, and internships with some introductory experience with version control, security, command line tools, web development, and user-centered design. The course is significantly composed of classwork activities so that students are highly engaged and regularly expected to work with their peers. Furthermore, this course could be a gateway for students to engage in more computing experience outside of their college coursework: hackathons, online coding challenges, computing clubs, personal projects, or Capture-the-Flag cybersecurity challenges. As expected, the combination of practical technical and problem-solving skills in a supplemental CS2 course helped improve students' overall reported comfort with computing in a variety of settings. Other CS educators may want to consider this course redesign model to engage and facilitate problem-solving skills among students. The results of the assessment tied to the project demonstrate that the pedagogical model can facilitate student confidence as it relates to problem-solving across various student levels.

ACKNOWLEDGEMENT

©2020 American Society for Engineering Education. ASEE Annual Conference Proceedings, June 2020, Virtual Conference

REFERENCES

Alvarado, C., Umbelino, G., & Minnes, M. (2018). The persistent effect of pre-college computing experience on college CS course grades. *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3159450.3159508

Amelink, C., Davis, K., Ryder, B., & Ellis, M. (n.d.). Exploring factors influencing the continued interest in a Computer Science major. *2018 ASEE Annual Conference & Exposition Proceedings*. https://doi.org/10.18260/1-2--30488

Auth0. (2020, August 1). *DB Browser for SQLite*. Retrieved from https://sqlitebrowser.org/

balsamiq. (2019*)*. Retrieved from https://balsamiq.com/learn/

Bressoud, T.C., & Thomas, G. (2019). A novel course in data systems with minimal prerequisites. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3287324.3287425

Codecademy. (2019). Retrieved from https://www.codecademy.com/catalog/subject/all

Computing Research Association Evaluation. (2020). *Data Buddies*. CERP. Retrieved from https://cra.org/cerp/data-buddies/

Davis, J., & Rebelsky, S.A. (2019). Developing soft and technical skills through multi-semester, remotely mentored, community-service projects. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3287324.32875

Free Software Foundation. (2019). *GNU Operating System*. Retrieved from https://www.gnu.org/software/bash/

Frieze, C., & Blum, L. (2002). Building an effective computer science student organization. *ACM SIGCSE Bulletin*, *34*(2), 74–78. https://doi.org/10.1145/543812.543835

Gersting, J.L., & Young, F.H. (1998). Service learning via the computer science club. *ACM SIGCSE Bulletin*, *30*(4), 25–26. https://doi.org/10.1145/306286.306304

Grow, G. (1991). Teaching learners to be self-directed. *Adult Education Quarterly*, *41*(3), 125–149.

Google Cloud Platform. (2019). *Google Cloud Platform.* Retrieved from https://cloud.google.com/

Google Inc. (2019). *Chrome DevTools Tools for Web Developers Google Developers*. Retrieved from https://developers.google.com/web/tools/chrome-devtools/

Grey, C.G.P. (2019*). How Machines *Really* Learn*. Retrieved from https://www.youtube.com/watch?v=wvWpdrfoEv0

Hasni, T.F., & Lodhi, F. (2011). Teaching problem solving effectively. *ACM Inroads*, *2*(3), 58–62. https://doi.org/10.1145/2003616.2003636

icpc.foundation. (2019). *ICPC*. Retrieved from https://icpc.baylor.edu/

Iberman, S.J. (2011). The Computer Science Club: Building a student community with projects and activities that extend beyond the classroom: Faculty poster. *J. Comput. Sci. Coll.*, *26*(6), 178–179.

Jones, B.D., Tendhar, C., & Paretti, M.C. (2016). The effects of students' course perceptions on their domain identification, motivational beliefs, and goals. *Journal of Career Development*, *43*(5), 383–397. https://doi.org/10.1177/0894845315603821

Koppelman, H., Dijk, B. V., & Hoeven, G.V.D. (2011). Undergraduate research. *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education - ITiCSE '11*. https://doi.org/10.1145/1999747.1999828

LeetCode. (2019). Retrieved from https://leetcode.com/explore/

Lewis, C.M. (2017). ACM RETENTION COMMITTEE Twelve tips for creating a culture that supports all students in computing. *ACM Inroads*, *8*(4), 17–20. https://doi.org/10.1145/3148524

Lewis, C.M., Yasuhara, K., & Anderson, R.E. (2011). Deciding to major in computer science. *Proceedings of the Seventh International Workshop on Computing Education Research - ICER '11*. https://doi.org/10.1145/2016911.2016915

Linhoff, J., & Settle, A. (2009). Motivating and evaluating game development capstone projects. *Proceedings of the 4th International Conference on Foundations of Digital Games - FDG '09*. https://doi.org/10.1145/1536513.1536541

Loksa, D., & Ko, A.J. (2016). The Role of Self-Regulation in Programming Problem Solving Process and Success. *Proceedings of the 2016 ACM Conference on International Computing Education Research*. https://doi.org/10.1145/2960310.2960334

Runestone Interactive LLC. (2019). *How to Think Like a Computer Scientist: Interactive Edition*. Retrieved from https://runestone.academy/runestone/static/thinkcspy/index.html.

Loksa, D., Ko, A.J., Jernigan, W., Oleson, A., Mendez, C.J., & Burnett, M.M. (2016). Programming, problem solving, and self-awareness. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/2858036.2858252

McCartney, R., Boustedt, J., Eckerdal, A., Sanders, K., Thomas, L., & Zander, C. (2016). Why computing students learn on their own. *ACM Transactions on Computing Education*, *16*(1), 1–18. https://doi.org/10.1145/2747008

McCartney, R., Eckerdal, A., Mostrom, J.E., Sanders, K., & Zander, C. (2007). Successful students' strategies for getting unstuck. *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education - ITiCSE '07*. https://doi.org/10.1145/1268784.1268831

Mohan, S., Chenoweth, S., & Bohner, S. (2012). Towards a better capstone experience. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE '12*. https://doi.org/10.1145/2157136.2157173

Nandi, A., & Mandernach, M. (2016). Hackathons as an informal learning platform. *Proceedings of the47th ACM Technical Symposium on Computing Science Education - SIGCSE '16*. https://doi.org/10.1145/2839509.2844590

NumFOCUS. (2019). *matplotlib*. Retrieved from https://matplotlib.org/

NumFOCUS. (2019). *Python Data Analysis Library*. Retrieved from https://pandas.pydata.org/

O'Neil, C. (2017). *The Era of Blind Faith in Big Data Must End.* Ted Talks. Retrieved from https://www.ted.com/talks/cathy_o_neil_the_era_of_blind_faith_in_big_data_must_end

Prather, J., Pettit, R., Becker, B.A., Denny, P., Loksa, D., Peters, A., … Masci, K. (2019). First things first. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. https://doi.org/10.1145/3287324.3287374

pydev. (2019). Retrieved from https://www.pydev.org/

Rittmayer, A.D., & Beier, M.E. (2008). Overview: Self-efficacy in STEM. *SWE-AWE CASEE Overviews*, *1*, 12.

Roick, J., & Ringeisen, T. (2017). Self-efficacy, test anxiety, and academic success: A longitudinal validation. *International Journal of Educational Research*, *83*, 84-93.

Software Freedom Conservancy. (2019). *Git—Distributed is the New Centralized*. Retrieved from https://git-scm.com/

SQLBolt. (2019). Retrieved from https://sqlbolt.com/

The Eclipse Foundation. (2019). *ECLIPSE Foundation*. Retrieved from https://www.eclipse.org/downloads/

Thompson, A.A. (2006). Approaches to recruiting and retaining in computer-science based student organizations. *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education - ITiCSE-WGR '06*. https://doi.org/10.1145/1189215.1189178

University of Canterbury Computer Science Education Research Group. (2020). *Chapters*. Chapters - Computer Science Field Guide. Retrieved from http://csfieldguide.org.nz/en/chapters/index.html

Virginia Cyber Range. (2019). Retrieved from https://virginiacyberrange.org/

Wireshark. (2019). Retrieved from https://www.wireshark.org/

You Tube. (2019). Retrieved from https://www.youtube.com