ENRICHMENT OF STUDENT LEARNING AND HOMEWORK MANAGEMENT WITH USE OF GITHUB® IN AN INTRODUCTORY CROSS-DISCIPLINARY ENGINEERING COURSE SERIES ON SOFTWARE ENGINEERING AND DATA SCIENCE

Chad D. Curtis, Caitlyn M. Wolf, and David A.C. Beck
University of Washington • Seattle, WA 98195-1750

INTRODUCTION

Git™ and GitHub® have found a growing niche in academia as both software development tools to encourage reproducible research and as educational tools to provide course content, receive student submissions, and foster a collaborative environment.[1-4] At their core Git and GitHub are tools to manage changes to files and documents, especially among teams and groups.[5] Git refers to the version-control software itself, while GitHub is a cloud-based implementation of Git. In this work we will use the term GitHub to refer to the synchronous use of both Git and GitHub.

GitHub's initial and predominant user base was, and is, software engineers. The application of this versatile platform in the classroom has also largely been limited to computer science and software engineering courses. Of 15 educators interviewed in a 2015 study who use or have used GitHub to support teaching and learning, 11 of them were in computer science courses, 2 in humanities courses, 1 in the natural sciences, and 1 in statistics.[6] In a more recent work, authors similarly note that the majority of GitHub implementations in the classroom are related to computer science, but they emphasize the usefulness of GitHub's collaborative and version-tracking attributes in other disciplines such as law.[7]

Despite many other available platforms for online course delivery, GitHub is seeing an increased utilization for this purpose, either alone or paired with traditional learning management systems.[7,8] It can simultaneously serve as a collaborative interface, a feedback mechanism, a version control system, and a community base.[9] Angulo et al. notes advantages of using GitHub for homework and team projects, including easy modification of code by the students, direct code review and feedback by instructors or teammates, and simple instructor access to and management of assignments.[7] In a 2019 survey students and instructors report improved teamwork skills, more project management experience, and an increased "sense of belonging" after the implementation of GitHub in the classroom.[3]

Chad Curtis *(0000-0001-6312-392X) is a lecturer in the Department of Chemical Engineering at the University of Washington. He teaches primarily computational-based courses and design courses. His teaching interests include the incorporation of computational tools in engineering education and an emphasis on statistics and scientific writing. His graduate research focused on the use of data science tools to improve the analysis of nanoparticle trajectory datasets for nanotherapeutic applications.*

Caitlyn M. Wolf *(0000-0002-2956-7049) is a PhD Candidate in the Department of Chemical Engineering at the University of Washington. As a member of the campus data science community, she has completed the Advanced Graduate Data Science Option and is working as a teaching assistant for the DIRECT program. Her research utilizes neutron and x-ray scattering, molecular simulations, and data science tools to explore molecular conformation and dynamics and improve molecular modeling methods of semiconducting polymers.*

David A.C. Beck *(0000-0002-5371-7035) is a Research Associate Professor in the Department of Chemical Engineering at the University of Washington and Director of Research in the eScience Institute, which is the University of Washington's data science institute. He is also Associate Director of the NSF funded NRT-DESE: Data Intensive Research Enabling Clean Technologies (DIRECT, award # 1633216). His work is at the intersections of molecular data science and energy, environment and health.*

Beyond its importance in computer science coursework as a lesson in version control,[8] GitHub has demonstrated its utility as an educational tool for a broader audience.

We have implemented GitHub as a key component of a course series on software engineering and data science methods geared towards engineering students in the general chemical sciences. While there are alternative version control platforms available (e.g. BitBucket, SourceForge), we chose to use GitHub for both its excellent classroom implementation (GitHub Classroom) as well as its open-source orientation. The Software Engineering for Molecular Data Scientists (SEMDS) and Data Science Methods for Clean Energy Research (DSMCER) dual course structure was originally developed as a part of the NSF funded NRT-DESE: Data Intensive Research Enabling Clean Technologies (DIRECT, award # 1633216). It has since expanded to include students across multiple disciplines with home departments of chemistry (26%, current cohort), chemical engineering (34%), materials science and engineering (14%), and molecular engineering and sciences (16%) with the remaining 10% distributed across other departments such as electrical engineering. Many of the students have little to no prior experience in coding, software engineering, or data science. In addition to introducing students to valuable tools that are applicable to their own research, we are also interested in using GitHub as a tool to foster collaboration and reproducibility. This is in part a response to the increased emphasis of open science and open data in research.[10-12] In this paper we will demonstrate how GitHub can be used to monitor student activity and collaboration, measure student perspectives on the use of GitHub and other aspects of the course before and after, and measure class performance metrics. We will also highlight other key components of the course that contributed to student learning. The glossary of terms found in Table 1 will be helpful to readers not familiar with GitHub and other programming tools.

EXPERIENCES AND ASSESSMENT

The SEMDS and DSMCER graduate courses are a requirement for graduate students in the NSF-funded DIRECT program and part of the transcriptable data science degree options in Chemical Engineering at UW, and are also available to other students as electives. There are no prerequisites for these courses, and the courses are taught in conjunction with one another over a ten-week period. Both classes take place twice a week for a total in-class time of six hours. The SEMDS course material focuses primarily on software engineering skillsets: Bash shell scripting, package management, Python®, formal software design, unit testing, continuous integration, reproducibility, etc. The DSMCER course focuses on statistics and machine learning analyses, hypothesis testing, bootstrapping, data visualization, classification, regression, etc. The class schedules are interchangeable, allowing for some SEMDS material-heavy weeks and some DSMCER material-heavy weeks.

TABLE 1
Glossary of software-related terms

Term	Definition
version control	A system for managing and recording any changes to files over time.
Repository (repo)	Similar to the concept of a computer folder — a location to store all project files, documentation, and each file's version history.
commit	An individual change or revision to a file or set of files recorded by Git.
issue	GitHub messaging tool for discussion around suggested improvements, tasks, or questions related to the repository.
branch	A parallel version of a repository.
fork	A personal copy of another user's repository that can be changed without affecting the original.
shell	A special user program that allows the user to send commands directly to the operating system; a command-line interface.
Jupyter notebook	A web application that allows users to create documents composed of live code, visualizations, and narrative text.
clone	A command to create a copy of a repository.
NumPy	A widely used Python package for scientific computing making use of multi-dimensional arrays.[13,14]
Pandas	Short for "panel data," a popular Python package for working with tabular data.[15]
use case	In software engineering, a description of how a person will use a process or system to accomplish a task.
unit test	A software testing method in which blocks or units of code are tested to ensure they are working as intended.
Travis	A continuous integration service that tests software projects and runs unit tests as they are being built.

The SEMDS and DSMCER courses are primarily project-based, but each course includes five homework assignments. All homework is managed using GitHub Classroom. Fiksel et al. comment on the utility of GitHub Classroom as an educational tool that enables private student repositories in compliance with the United States Family Educational Rights and Privacy Act of 1974 (FERPA).[16] This ensures that student privacy is maintained during the homework grading process. GitHub Classroom can also be linked to a learning management system (LMS) such as Canvas® to aid in assignment and grade management. For each homework, feedback is given via GitHub issues, and the assignment can be modified and resubmitted up to one week after grades are posted for a chance to earn full points.

One project is given both a SEMDS grade based on good software engineering practices and a DSMCER grade based on data science tool implementation. Teams of 4-5 members are formed in the fifth week, giving students roughly five weeks to develop projects. Students are encouraged to use data from their own research for the projects, and many projects continue on past the end of the course. All projects are managed by the students and hosted on public or private GitHub repositories. While there can be unique circumstances requiring project repositories to remain private (e.g. data usage rights), we encourage students to develop their projects in a public repository. This promotes an open science approach and gives the students experience with open-source software development. All assessments of student projects are, however, performed privately among the instructors rather than over GitHub issues to ensure student privacy.

There are no prerequisites for these courses. That is, we assume no prior computational or programming experience; however, some knowledge of the molecular sciences helps to contextualize the computation and data science education. For example, some homework assignments refer to energy landscapes or basic molecular orbital concepts such as LUMO and HOMO. As the courses run concurrently, we are able to utilize the in-class time for both courses in the first two and a half weeks to teach introductory to intermediate Python programming to all students. The course materials are all open source, and the syllabus is available at https://uwdirect.github.io.[17]

Course material is presented primarily with Jupyter® notebook after some initial background material is presented using traditional slides. Jupyter notebooks are a convenient way to interweave code, text, and images in an interactive format.[18-21] Class communication is managed using Slack®, a collaboration platform with both general channels (similar to a group discussion board) for class-wide conversation and private messaging for student-student and student-instructor discussions. Students are also provided access to weekly office hours when in-person communication is preferred.

A course survey was sent out at the end of Winter quarter 2019 (n = 51 students, n = 44 responses). Students were asked questions evaluating their experiences in both the SEMDS and DSMCER courses related to their experiences with both course content (e.g. software development, machine learning) and course structure (e.g. homework feedback via GitHub issues, communication via Slack). We were unable to administer a similar pre-course survey and instead had to rely on student recollections of their experience at the beginning of the course from the post-course survey. A retrospective survey was also collected from students of the previous two cohorts in 2017 and 2018 (n = 12 responses) evaluating their current use of skills acquired in SEMDS and DSMCER as well as whether those skills have proved useful in their current occupations.

COURSE PERFORMANCE

Homework

A key feature of the SEMDS/DSMCER course sequence is the use of GitHub for both homework submission and instructor feedback. Each homework is created from a template GitHub repository in which instructors can load any necessary files, such as instructions, datasets, and other supporting material. Each student creates their own personal copy of the homework template from which they can work on their local machine. As students work on their homework assignments, they can commit and push their progress from their local machine to their GitHub repository stored in the cloud. Instructors can then clone a copy of each student homework repository on the assigned due date. Github Classroom helps streamline this process by allowing instructors to batch clone all student submissions for a given homework assignment.[16]

Instructor feedback for our course was handled via another key GitHub feature: GitHub issues. When instructors are ready to comment on a student's assignment, they can open a new GitHub issue. After students have edited their documents accordingly, they can commit and push their changes to the remote version of their homework and use the issue to reply directly to the instructor comments. Instructors can then review the edited documents and submit a new grade via the same GitHub issue, creating a recorded dialogue with the student so that all instructor and student comments are stored together with the version-controlled homework documents in one easily accessible and timestamped location. We have further automated this process by writing a script for batch submission of instructor feedback via GitHub issues.[22]

Each SEMDS/DSMCER homework was graded on a scale of 0 to 5. A README.md file in each homework repository defines how the points are distributed for each portion of the assignment and the tasks required for full credit. Students were given one week after each homework grade was returned to address instructor comments and resubmit. Homework assignments were regraded, and then only this updated score was counted in the final student grade for the course.

Homework Resubmissions

Mastery and resubmission approaches to engineering and programming homework assignments have already been shown to be advantageous for student learning.[23, 24] By allowing students to revise their homework, it has been reported that students spend more time focused on learning the concepts and responding to instructor feedback rather than the homework grade.[23] It was also found that students were more engaged in these courses and had more positive perspectives of the homework process.[24] Similarly, we have found that the two-fold grading scheme enhanced student success in the classroom in our SEMDS and DSMCER classes.

Compiled grades before and after addressing instructor comments are shown in Figure 1. All homework assignments had a median grade of 5 after addressing revisions. The hardest assignments were SEMDS-HW4, with an initial median score of 3.0 (interquartile range [IQR] 2.0-4.0) and 18% receiving a grade of 5, and DSMCER-HW4, with an initial median score of 2.0 (IQR 1.0-3.0) and 12% receiving a grade of 5. Both of these assignments were SEMDS-HW4 focused on development of a k-Nearest Neighbor (KNN) classifier[25] while DSMCER-HW4 focused on a multiple linear regressor.[26] Both of these assignments were cumulative, asking students to incorporate good programming documentation practices and advanced machine learning concepts they had been learning over the duration of the course. The easiest homework was DSMCER-HW1, with an initial median score of 5.0 (IQR 5.0-5.0) and 94% receiving a grade of 5, which required students to summarize a paper about data science in their own words.

Most of the students who did not get a perfect score upon first submission chose to address instructor comments. For SEMDS homework assignments 1 through 4, we found that 77, 93, 93 and 88% of students without perfect scores revised their homework based on instructor feedback, respectively. Median improvements in grades were 1.0 (IQR 1.0-1.0), 1.0 (IQR 1.0-1.0), 1.0 (IQR 1.0-2.0) and 2.0 (IQR 1.0-3.0), respectively. For DSMCER homework assignments 1 through 4, we found that 100, 85, 87 and 95% of students without perfect scores revised their homework based on instructor feedback, resulting in median improvements in grades of 1.0 (IQR 1.0-1.0), 1.0 (IQR 1.0-2.0), 1.5 (IQR 1.0-2.75) and 3.0 (IQR 2.0-4.0), respectively. Prior to regrading, only 2% of students had perfect scores on all SEMDS assignments (mean score of 3.86), and only 2% of students had perfect scores on all DSMCER assignments (mean score of 3.44). No students had perfect scores on all assignments for both courses before regrading. After regrading, 66% of students had perfect scores on all SEMDS assignments (mean score of 4.79), and 62% of students had perfect scores on all DSMCER assignments (mean score of 4.72). Additionally, 48% of students had perfect scores on all assignments for both courses after regrading.

The two-tiered grading structure allowed students to internalize material and address errors or misunderstandings. Small mistakes were not held against students in an arbitrary fashion since they were able to address them in the second round of grading. This also gave students the opportunity to address errors that resulted from an incomplete understanding of the concepts and to learn from their mistakes. As a result,

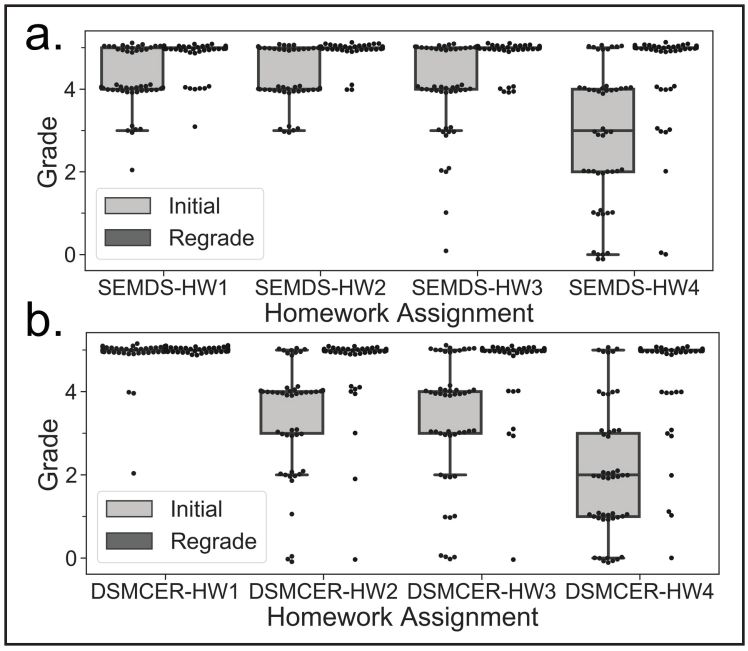

Figure 1. *Homework grades for (a) SEMDS and (b) DSMCER. 'Initial' homework grades were based on each student's initial submission of their assignment. 'Regrade' scores were based on each student's second submission of their assignment after addressing instructor comments from the first version. Grades for the fifth homework assignments in each course were excluded as they were built into the final projects and did not have a similar initial/regrade scheme. Box plots for each homework assignment are shown overlaid on top of a scatter plot of the raw data to more easily view the distribution of grades. The bottom and top edges of the box reference the 25% and 75% percentiles, and the middle line of the box represents the median. The bottom and top whiskers reference the minimum and maximum of the data, respectively, not including outliers. For some assignments, narrow grade distributions result in overlapping 25% and 75% percentiles, and thus have no box present. For these assignments, they instead appear as horizontal lines at the median. The plots were created using the Python packages Matplotlib and Seaborn.*[24]

all students were able to feel more in control of their homework grades and obtain a deeper understanding of the material.

Homework Commit Histories

GitHub commit histories allow instructors to gauge student success not only by their grades, but also by progression on their homework assignments. Instructors can observe trends in this data to learn more about how students are approaching the work. For example, many students continued to work on their homework assignments after the initial deadline. There were often many late submissions, especially for DSMCER-HW3, but the chance for homework regrades gave students this buffer. As an instructional team, we chose to give feedback to assignments submitted after the first due date. Final submissions had on average between 500-1500 lines of code, but there was a wide range, with some students submitting upwards of 10,000 lines of code for some assignments.

A good practice during software development is to commit early and often, and we tried to reinforce this concept to students throughout the quarter. Especially in team projects and collaborative efforts, committing early and often prevents overly complex merge conflicts and improves communication. We explored how well this message came through by tracking the number of commits in each homework assignment. These are shown in Figure 2. Even at the beginning of the course, students were submitting assignments with more than one commit. SEMDS-HW1 had a median commit count of 4.0 (IQR 2.0-5.0) with 89% having more than one commit. DSMCER-HW2 had a median commit count of 2.0 (IQR 2.0-4.0) with 91% having more than one commit. This was also fairly consistent throughout the courses. SEMDS-HW4 had a median commit count of 3.0 (IQR 2.0-6.8) with 94% having more than one commit. DSMCER-HW4 had a median commit count of 3.0 (IQR 2.0-4.0) with 100% having more than one commit. Resubmissions, on the other hand, were often addressed using a single commit, e.g. DSMCER-HW2 resubmission had a median commit count of 1.0 (IQR 1.0-1.0) with only 18% having more than one commit. However, these additions were often not as substantial as completing an entire assignment.

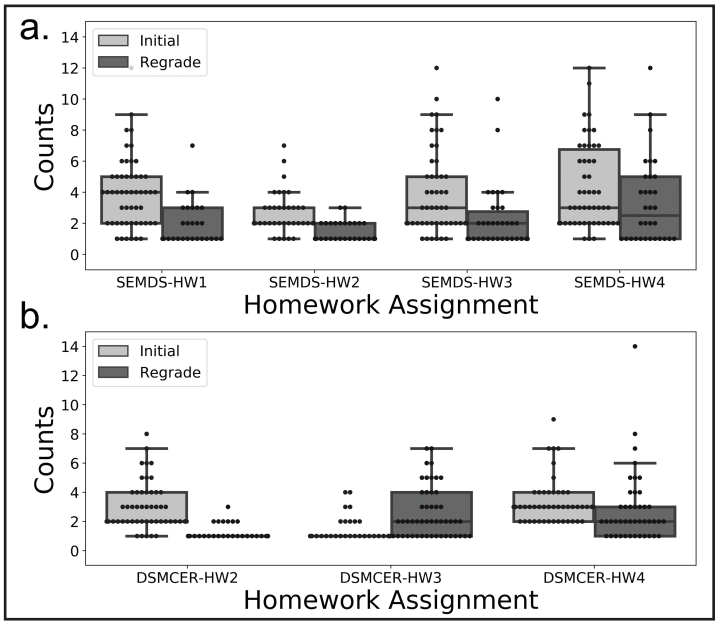

Figure 2. *Number of GitHub commits per homework assignment for (a) SEMDS and (b) DSMCER. 'Initial' homework grades were based on each student's initial submission of their assignment. 'Regrade' scores were based on each student's second submission of their assignment after addressing instructor comments from the first version. Grades for the fifth homework assignments were excluded as they were built into the final projects and did not have a similar initial/regrade scheme. Grades for DSMCER-HW1 were excluded because they were not submitted via GitHub.*

Projects

The course projects make up most of the final grade for both the SEMDS and DSMCER courses. Each student is part of a team having 4 to 5 members. Projects ideas are developed by the students themselves, with one member often bringing a dataset from their research as a foundation for the project. There is only one project for both the SEMDS and DSMCER courses, but each project is evaluated for SEMDS and DSMCER content separately.

During the academic cycle when the data were collected, there were a wide array of project topics:

P0. Prediction of battery degradation using a few cycles of operation data.

P1. Prediction of enzymes with promiscuous activity specific to a compound of interest based on activity to chemically similar compounds.

P2. Analysis of particle characteristics from SEM images.

P3. Prediction of the bandgap of organic semiconductors using data extracted from SMILES strings.

P4. Extraction of synthesis and performance metrics from academic articles on perovskite solar cells.

P5. Prediction of the power conversion efficiency of organic materials from data extracted from SMILES strings.

P6. Using Raman spectroscopy measurements to identify decomposition and formation products in a water gasification reactor.

P7. Feature extraction from fluorescent protein images using Fourier transforms.

P8. Extraction of elevation profiles from bus routes and comparison of route difficulty for battery use.

P9. Prediction of fluorescence emission and absorption spectra using photochemical datasets.

P10. Peak finding using cyclic voltammetry data.

P11. Prediction of the density of state of new materials using X-ray diffraction data.

Similar to tracking homework progress individually, GitHub can be used as a tool to measure team performance. Commit histories of the project repositories not only allow instructors to observe how projects develop over time but also to gauge individual contributions to each project. Contributions to the code can be an objective measure of individual inputs that either balance or confirm more subjective measures like peer evaluations. The cumulative repository history in lines of code is shown in Figure 3. Some teams like P1 and P3 got an early start on the project with gradual additions throughout the quarter, while other teams like P2 and P4 made the biggest additions to their code the day before the due date.

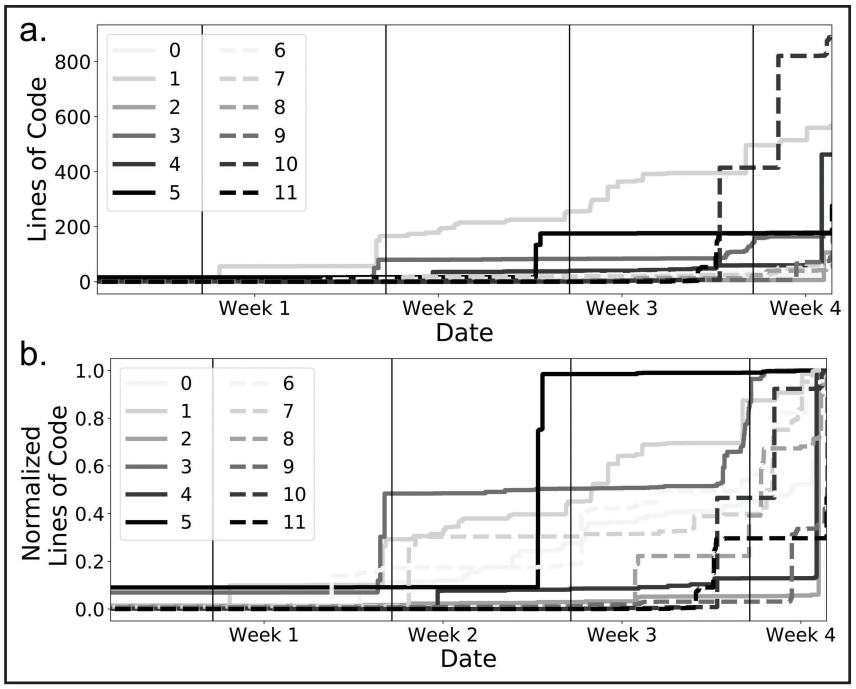

Figure 3. *Final project commit histories in terms of (a) lines of code (in thousands) and (b) lines of code normalized to final number of lines of code.*

Team Performance

The amount of cooperation within teams was evaluated in terms of both individual contributions to project repositories, including number of commits and lines of code contributed as shown in Figure 4, and peer evaluations. In this evaluation each team member was asked to divide 100 points among the team members, and then the results were tallied up for each team

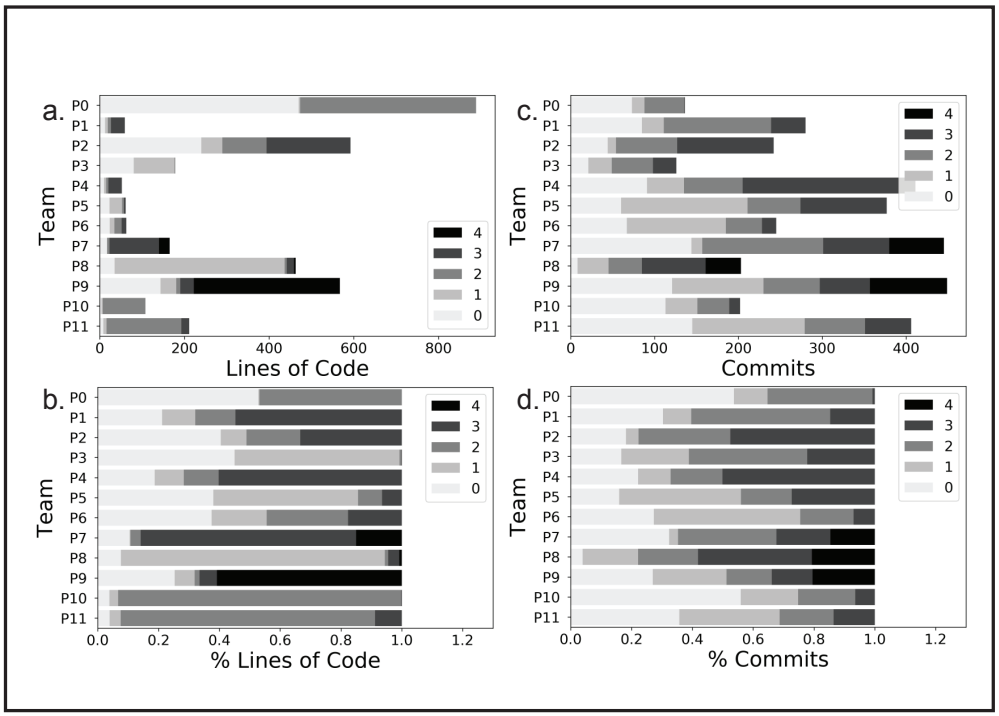

Figure 4. *Project contributions from each team member by (a-b) lines of code (in thousands) and (c-d) number of commits. In (b) and (d), the data in plots (a) and (c) have been normalized to total lines of code and total number of commits per team, respectively*

member. If a team member felt that each team member contributed equally in a team of four, then he or she would assign 25 points to each team member, including himself or herself.

In order to quantify the distribution of work among team members, the standard deviation of each normalized measure (commits, lines of code, and peer evaluations) was reported for each team (see Table 2). Teams with scores close to 0 had high measures of "evenness" with team members contributing equally. In terms of commits, the team that contributed the most equally to their project was P9 (0.06), while in terms of lines of code, it was P6 (0.09). When both scores are averaged into a composite score, the team that achieved the most even distribution among team members was also P6 (0.14).

Not all teamwork can be entirely captured in terms of lines of code or commits. Some students may have contributed more to the conceptualization of the project or to aspects of the project that were not coded, such as documentation. Therefore, we also included peer evaluation results in final scores. Team members were usually generous to their teammates in these peer evaluations. Peer evaluation evenness scores were in all cases lower than the composite coding evenness scores. However, it was found that the two scores were correlated with each other (Pearson correlation coefficient of 0.61).

Course Survey

During the last week of the course, we sent a survey to all students in the class to gauge student confidence in their learning of course material and the usefulness of certain aspects of the course. In order to ensure a high response rate, we bundled the survey with the non-mandatory project peer evaluation. Out of a class of 51 students, we received 44 responses for an 86% response rate.

TABLE 2
Standard deviations of normalized team member contributions to final projects in terms of commits, lines of code, and peer evaluations. Low standard deviations indicate equal contributions of all team members. Extreme high and low values are bolded.

	Commits	Lines of code	Composite	Peer Evaluations
P0	**0.24**	0.29	0.27	0.09
P1	0.16	0.20	0.18	0.02
P2	0.18	0.15	0.17	**0.00**
P3	0.10	0.29	0.20	0.01
P4	0.17	0.24	0.21	0.03
P5	0.11	0.21	0.16	**0.13**
P6	0.18	**0.09**	**0.14**	**0.00**
P7	0.13	0.29	0.21	0.03
P8	0.12	0.37	0.25	0.08
P9	**0.06**	0.25	0.16	0.03
P10	0.21	**0.45**	**0.33**	0.11
P11	0.11	0.39	0.25	0.12

Overall Course Material

First, we sought to measure student confidence in a few key general areas in both the SEMDS and DSMCER material both before and after the course series, including shell scripting, version control, Python, software development, and machine learning. Results from the questions *"Before taking the SEMDS/DSMCER course series, how confident were you with the following?"* and *"After taking the SEMDS/DSMCER course series, how confident were you with the following?"* on a scale of 0 to 10 are shown in Figure 5a.

Coming into the course, students were somewhat familiar with Python (median 2.0, IQR 0.0-5.2) but were unfamiliar with version control (median 0.0, IQR 0.0-2.0) and machine learning (median 0.0, IQR 0.0-2.0). It is common for engineering students to have an introductory knowledge of coding concepts, often taught in MATLAB. At the end of the course, students expressed the most confidence in programming with Python (median 8.0, IQR 7.0-8.2). Given that the majority of homework and project components are done in Python, this is encouraging and shows that students gained a solid grounding in coding concepts by the end of the course. Substantial gains in confidence were made in version control (median 6.0, IQR 4.0-7.0), Bash shell scripting (median 5.0, IQR 3.0-6.2), and software development (median 5.0, IQR 3.0-6.2).

Students remained least confident in machine learning concepts by the end of the course (median 5.0, IQR 4.0-7.0). It is likely that this is partially due to the newness of the concepts since most students had no prior experience with machine learning. However, it should also be noted that students did comment in the feedback section that there was too little time allotted in the course schedule to machine learning concepts and that it felt much too fast-paced. In future iterations of the course, we will consider how to rebalance the course material to include more time for the advanced data science topics.

Similar trends were seen in the retrospective survey results of previous cohorts shown in Figure 5b. Coming into the course, students reported some familiarity with Python (median 2.5, IQR 0.0-5.0) but little to no familiarity with Bash shell scripting (median 0.0, IQR 0.0-1.8), version control (median 0.0, IQR 0.0-1.0), software development (median 0.0, IQR 0.0-3.0), and machine learning (median 0.0, IQR 0.0-2.0). At the end of the course, previous cohorts

expressed the most confidence in Python (median 8.0, IQR 7.0-10.0). The biggest gain in confidence was made in version control (median 6.0, IQR 4.5-8.0), while students were least confident in Bash shell scripting after the course (median 6.0, IQR 4.0-8.0). It appears that confidence in machine learning concepts at the end of the course has decreased with respect to previous cohorts (from median 6.5, IQR 5.0-7.2 to median 5.0, IQR 4.0-7.0) while confidence in Bash shell scripting after the course has increased (from median 6.0, IQR 4.0-8.0 to median 7.0, IQR 5.8-8.0). Since taking the course, students report a slight decline in their Bash shell scripting ability (from median 6.0, IQR 4.0-8.0 to median 5.5, IQR 4.0-8.0) and an increase in their Python (from median 8.0, IQR 7.0-10.0 to median 9.0, IQR 7.8-10.0), software development (from median 6.5, IQR 5.8-8.2 to median 7.5, IQR 5.8-9.2), and machine learning (from median 6.5, IQR 5.0-7.2 to median 7.5, IQR 5.5-9.0) abilities. It is encouraging that students continue to build on the skills covered in these courses.

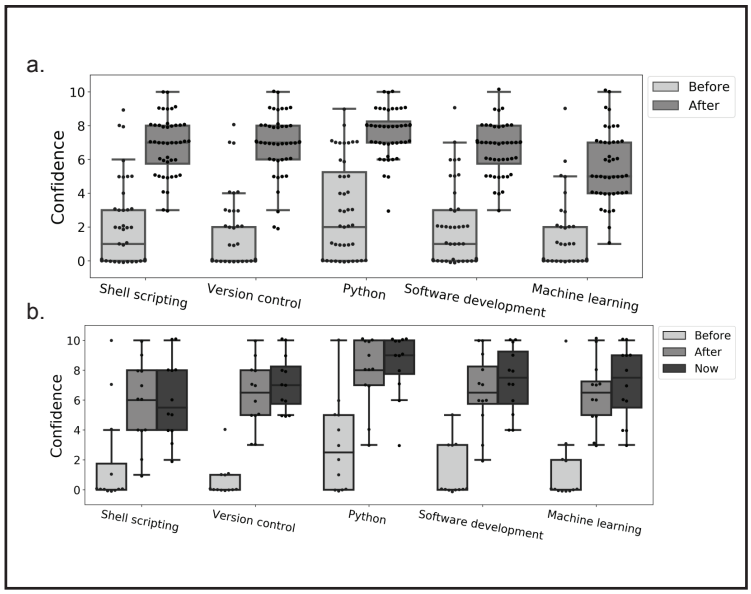

Figure 5. *Confidence expressed by students on a scale of 0 to 10 in key topics from the SEMDS/DSMCER course before and after the course series for (a) the current cohort (n = 44) and (b) previous cohorts (n = 12). Question prompts given were "Before taking the SEMDS/DSMCER course series, how confident were you with the following?" and "After taking the SEMDS/DSMCER course series, how confident were you with the following?"*

SEMDS

As shown in Figure 6a, by the end of the course students were most confident in navigation with the Bash shell (median 9.5, IQR 7.0-10.0), NumPy and Pandas (median 9.0, IQR 7.0-10.0), creating a repo (median 9.0, IQR 7.8-10.0), creating a README (median 9.0, IQR 8.0-10.0), writing a function (median 9.0, IQR 8.0-10.0), and writing a loop (median 9.0, IQR 7.8-10.0). Substantial gains in confidence were made in creating a repo (median 7.0, IQR 3.0-9.0), creating a README (median 7.0, IQR 4.0-9.0), unit testing (median 7.0, IQR 5.0-9.0), and creating an issue on GitHub (median 7.0, IQR 5.0-9.0). Some of this confidence was found to be related to previous experience. Prior to the course, students already had some confidence writing a loop (median 4.0, IQR 1.8-8.0), writing a function (median 2.0, IQR 2.0-8.0), and navigating with the Bash shell (median 2.0, IQR 1.0-8.0). These concepts are also key foundational skills introduced near the beginning of the course that were then

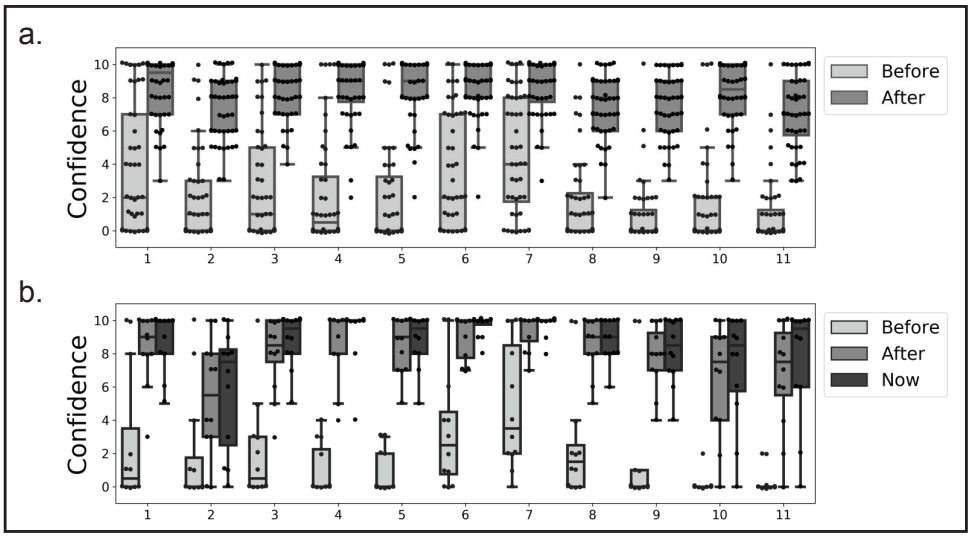

Figure 6. *Confidence expressed by students on a 0-10 scale in a given set of tasks in the SEMDS course (1. navigation with shell, 2. shell scripting, 3. NumPy/Pandas, 4. create a repo, 5. create a README.md, 6. write a function, 7. write a loop, 8. define a use case, 9. unit tests, 10. create an issue, 11. fork a repo) before and after the course series for both (a) current (n = 44) and (b) previous (n = 12) cohorts.*

cumulatively built upon throughout the next ten weeks.

By the end of the course, students were least confident in forking a repo (median 7.0, IQR 5.8-9.0). Forks are a useful way of contributing to a project without affecting the original project. However, branches were much more emphasized throughout the course for project development, likely contributing to the slightly lower confidence using forks. Overall, students were very comfortable with SEMDS course material.

Similar results were obtained in the retrospective survey of previous cohorts as shown in Figure 6b. After the course students were most confident in creating a repo (median 10.0, IQR 8.0-10.0), writing a function (median 10.0, IQR 7.8-10.0), writing a loop (median 10.0, IQR 8.8-10.0), navigation with the Bash shell (median 9.0, IQR 8.0-10.0), creating a README (median 9.0, IQR 7.0-10.0), and defining a use case (median 9.0, IQR 8.0-10.0). Substantial gains in confidence were shown for creating a repo (median 8.5, IQR 5.8-10.0), creating a README (median 7.5, IQR 6.5-9.2), unit tests (median 7.5, IQR 4.5-8.0), and creating an issue (median 7.5, IQR 4.0-9.0). By the end of the course, previous cohorts were least confident in Bash shell scripting (median 5.5, IQR 3.0-8.0). It is interesting that confidence in Bash shell scripting after the course has increased compared to previous years (from median 5.5, IQR 3.0-8.0, to 8.0, IQR 6.0-9.0) indicating that the SEMDS course material is becoming more effective, either through out-of-class support or improved implementation in the course material.

As shown in Figure 7a, most students had already used a for/while loop structure (80%), used a function (73%), and used a Jupyter notebook (57%) before the course. However, for a portion of the class (9%), everything in the SEMDS material was completely new. Similar trends were observed in the retrospective survey of previous cohorts (for/while loops 83%, functions 75%, Jupyter notebooks 25%, none 17%) as shown in Figure 7b. The use of Jupyter notebooks prior to the course has increased over time, likely due to the higher data science emphasis across departments. Most of the class were confident they would continue to use the course material in their research (Bash shell scripting 91%, Jupyter notebooks 95%, GitHub 98%, make a repo 91%, use a for loop 98%, make a function 95%, make a unit test 93%). In terms of outcomes, this is a huge success. Introducing version control and open-source software to students should help improve collaborative workflows and reproducibility of analyses for future research.

Students were less confident they would continue to use continuous integration tools such as Travis (61%) and virtual environments (75%). These were introduced near the end of the course, and their utility may not have been fully internalized by students. However, we did provide students with resources they can use beyond the course, such as the template repositories (e.g. http://github.com/dacb/codebase), so they can continue to play around with these productivity tools. We also seek to increase the number of touchpoints students have with software and data science tools through programs such as Software Carpentry and quarter-long incubator grants sponsored by the eScience Institute at the University of Washington.[28]

Previous cohorts reported that a large number of students continue to use Jupyter notebooks (92%), for loops (92%),

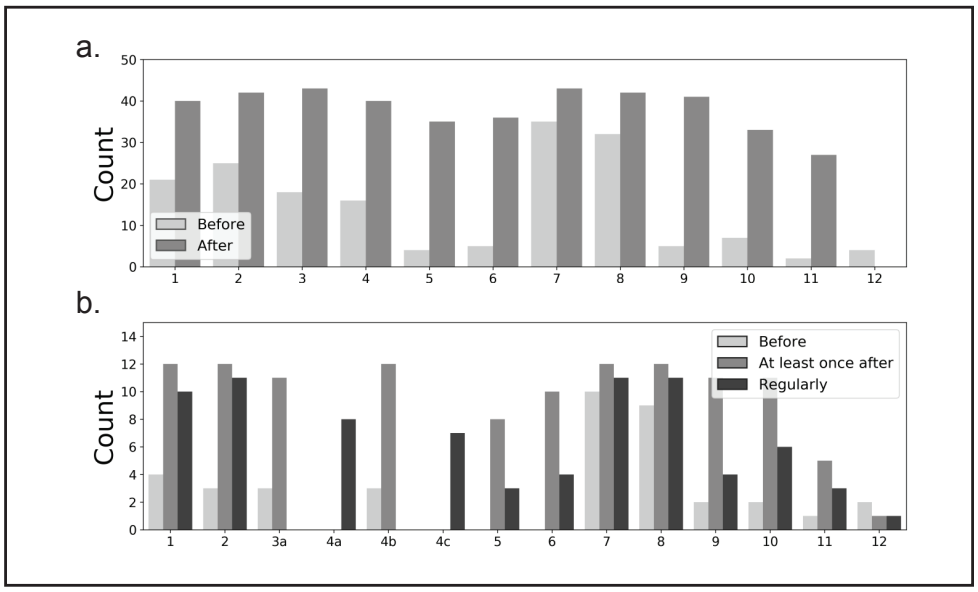

Figure 7. *Before and after experience with key course components (1. shell, 2. Jupyter, 3. GitHub, 4. make a repo, 5. create an issue, 6. create a branch, 7. use a for loop, 8. make a function, 9. make a unit test, 10. create a virtual environment, 11. Travis, 12. none; alternate questions for retrospective survey: 3a. make a GitHub account, 4a. manage a repo, 4b. make a repo, 4c. collaborate via Git for (a) current and (b) previous cohorts. (a) Counts of students who answered in the affirmative to "Which of the following have you done prior to taking the SEMDS course?" and "Which of the following are you likely to do again after taking the SEMDS course?" for the current cohort (n = 44). (b) Counts of students who answered in the affirmative to "Which of the following have you done prior to taking the SEMDS course?," "Which of the following have you performed since taking the SEMDS course?" and "Which of the following do you perform on a semi-regular basis?" for previous cohorts (n = 12).*

functions (92%), and Bash shell scripting (83%) regularly after the course, confirming that these skills become an integral part of student workflows. Some aspects of the courses were not rated as highly by previous cohorts when compared to the reported likely future usage by the current cohort: GitHub 67% compared to 98% and unit tests 33% compared to 93%. We seek to encourage good practices like open science and purposeful use of software design, but these may continue to seem like "extras" to students rather than essential practices if further reinforcement is not implemented after the course. Some students may also enter careers that are not software oriented. The authors hope that by providing additional infrastructure in the department, such as capstone projects and collaborations with the eScience Institute, the number of students regularly implementing these good practices after the course will increase.

DSMCER

As shown in Figure 8a, by the end of the course students were most confident in calculating measures of central tendency (median 9.0, IQR 8.0-10.0) and simple linear regression (median 9.0, IQR 7.8-10.0). These are two concepts that engineering students are likely to encounter even if they have not taken a statistics course as reflected in the reported confidence prior to taking the DSMCER course (median 6.5, IQR 2.0-9.0 and median 2.0, IQR 0.0-6.0, respectively). Substantial gains in confidence were made in using a KNN classifier (median 7.0, IQR 5.0-8.2), visualizations with Python (median 6.0, IQR 2.8-8.0), multiple linear regression (median 6.0, IQR 3.0-8.0), and bootstrapping (median 6.0, IQR 5.0-8.0). It is reassuring that some of the more complicated material in the course was internalized, especially the machine learning component. Some of the later concepts, such as LASSO regression and neural networks, still felt unfamiliar to students by the end of the course (median 6.0, IQR 4.0-8.0 and median 6.0, IQR 3.0-7.0, respectively). Students expressed that they felt the pace of the class was "exponential" with more difficult topics being taught in rapid succession close to the end of the course without sufficient time to work through and internalize the concepts. Students wanting to implement neural networks in their projects were not taught the concepts until weeks 8 and 9 of the course, leaving short turn-around times for implementation.

We found similar results for the DSMCER course in the retrospective survey of previous cohorts. Students expressed most confidence in measures of central tendency (median 10.0, IQR 8.8-10.0, Figure 8b), simple linear regression (median 10.0, IQR 8.8-10.0), multiple linear regression (median 10.0, IQR 7.5-10.0), and KNN classifiers (median 9.0, IQR 6.0-10.0). Confidence in multiple linear regression (from median 10.0, IQR 7.5-10.0 to median 8.0, IQR 6.0-10.0) and KNN classifiers (from median 9.0, IQR 6.0-10.0, to median 8.0, IQR 7.0-9.2) seems to have waned slightly with the current cohort. Previous cohorts similarly struggled with later concepts in the course such as neural networks (median 5.5, IQR 5.0-8.5) and bootstrapping (median 6.0, IQR 3.8-7.5). Previous cohorts also reported higher confidences in visualization (from median 8.5, IQR 6.5-10.0, to median 10.0, IQR 8.0-10.0), visualization with Python (from median 7.5, IQR 5.0-9.2 to median 9.5, IQR 8.8-10.0), LASSO regression (from median 7.0, IQR 5.0-10.0 to median 8.0, IQR 6.8-10.0) and neural networks (from median 5.5, IQR 4.0-7.8 to median 7.5, IQR 5.0-8.5) since taking the course. These aspects of the DSMCER course seem to have become integral to student workflows, and confidence has increased with further

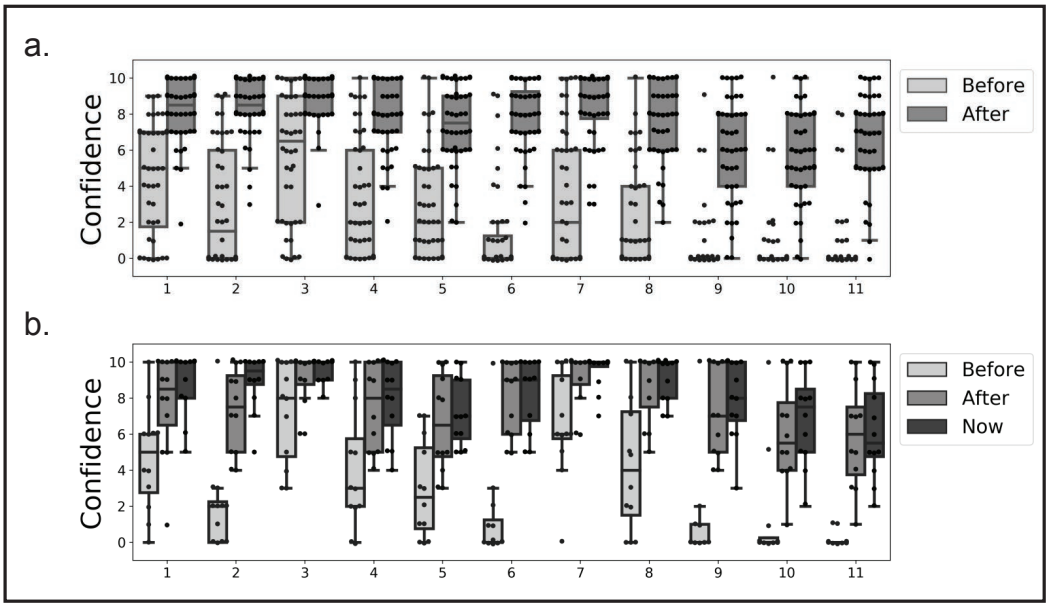

Figure 8. *Confidence expressed by students on a 0-10 scale in a given set of tasks in the DSMCER course (1. visualizations, 2. visualizations with Python, 3. central tendencies, 4. normal distribution, 5. hypothesis testing, 6. KNN classifier, 7. simple linear regression, 8. multiple linear regression, 9. LASSO regression, 10. neural networks, 11. bootstrap) before and after the course series for both (a) current (n = 44) and (b) previous (n = 12) cohorts.*

exposure. The only skillset that seems to have weakened and seen less application since taking the course is bootstrapping (from median 6.0 IQR 3.8-7.5 to median 5.5, IQR 4.8-8.2).

Contribution to Student Learning

We also wanted to gauge what students found most useful for their learning throughout the course. We included a survey outlining key components of content delivery throughout the course and posed the question, *"On a scale of 1 to 10, how helpful were the following to your learning?"* Results are shown in Figure 9. We found that students ranked components of GitHub in the top five contributors to their learning experience (homework submitted via GitHub, median 9.0, IQR 9.0-10.0 and homework feedback via GitHub issues, median 9.0, IQR 7.0-10.0).

Post-Course Accomplishments

Students of previous cohorts reported significant accomplishments implementing software engineering and machine learning methods since taking the dual course SEMDS/DSMCER program: 42% reported accepting a software engineering or data science-related job, 33% reported publishing a paper or presenting a poster or talk implementing DSMCER/SEMDS concepts, 42% further developed their projects from the course, 58% reported starting new projects related to data science, 25% took additional course work related to data science, and 100% of students reported using Slack as a means of collaboration. When students were asked, *"SEMDS/DSMCER helped me with my graduate work,"* students responded with a mean (± standard deviation) score of 0.67 ± 0.59 on a scale of -1 (strongly disagree) to 1 (strongly agree) as shown in Figure 10. Students were even more confident that *"SEMDS/DSMCER helped with [their] job prospects,"* responding with a mean score of 0.83 ± 0.42. Students reported strongly positive results to the statements *"SEMDS/DSMCER skillsets are now an integral part of my workflow"* (mean 0.71 ± 0.56) and *"I have built on my SEMDS/DSMCER skillsets"* (mean 0.79 ± 0.38).

CONCLUSIONS

We have examined how GitHub provides instructors with an additional quantitative tool to monitor and assess homework, project progress, and team contributions. As all metrics are inevitably vulnerable to manipulation (i.e. "gaming the system"), it is important that additional factors are taken into account when assessing team performance, such as peer review and manual checks. We have also assessed student confidence in key software engineering and machine learning concepts before and after the course, identifying areas of strength (creating repositories, writing for loops and functions, linear regression) and areas that could be improved (Bash shell scripting, bootstrapping, neural networks). We have demonstrated with data from previous cohorts that these software engineering and machine learning skills have been useful to students after the course for both their research and job prospects. Finally, we have demonstrated positive student feedback in using Slack as a primary in-class communication tool forum.

ACKNOWLEDGMENTS

The SEMDS and DSMCER courses were developed as a part of the NSF funded NRT-DESE: Data Intensive Research Enabling Clean Technologies (DIRECT, award # 1633216). We would like to thank the SEMDS and DSMCER course teaching assistants, Caitlyn Wolf, Torin Stetina, and Theodore Cohen for their help in course administration and for their personal investment in the students' education. We would also like to thank Dr. Jim Pfaendtner for his advice while writing the paper.

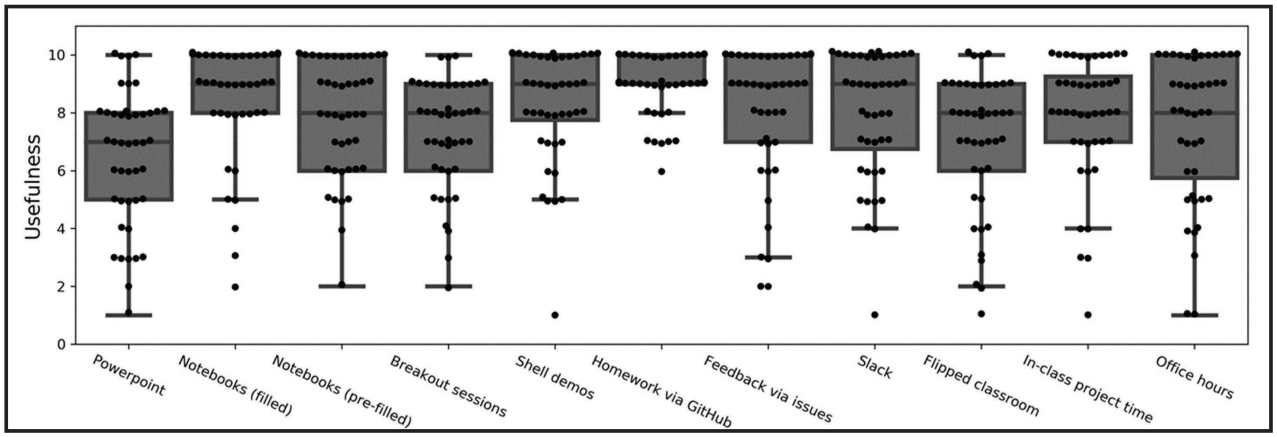

Figure 9. *The usefulness of key course features expressed by students on a 1-10 in the SEMDS/DSMCER course sequence. Survey results were collected in an online survey at end of quarter. Results represent n = 44 students from n(total) = 51.*

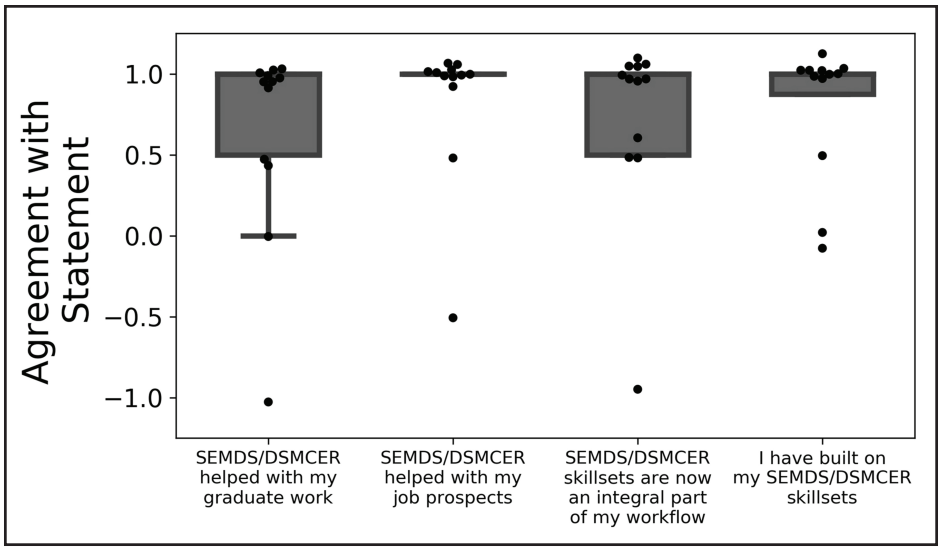

Figure 10. *Student responses to the statements: SEMDS/DSMCER helped with my graduate work, SEMDS/DSMCER helped with my job prospects, SEMDS/DSMCER skillsets are not an integral part of my workflow and I have built on my SEMDS/DSMCER skillsets. Answers coded as "Strongly disagree": -1.0, "Somewhat disagree": -0.5, " Neither agree nor disagree": 0.0, "Somewhat agree": 0.5, "Strongly agree": 1.0.*

REFERENCES

1. Groen D, Guo X, Grogan JA, Schiller UD and Osborne JM (17 June 2015) Software development practices in academia: A case study comparison. *arXiv:1506.05272*.
2. Geiger RS (8 June 2017) Summary analysis of the 2017 Github open source survey. *arXiv:1706.02777*.
3. Hsing C and Gennarelli V (2019) Using GitHub in the Classroom Predicts Student Learning Outcomes and Classroom Experiences: Findings from a Survey of Students and Teachers. *Proceedings ACM Technical Symposium on Computer Science Education*. 672-678.
4. Griffin T and Seals S (2013) Github in the classroom: Not just for group projects. *Journal of Computing Sciences in Colleges*. 28(4):74-74.
5. Gunnarsson S, Larsson P, Månsson S, Mårtensson E and Sönnerup J (2017) Enhancing student engagement using GitHub as an educational tool, (Genombrottet, Lunds tekniska högskola).
6. Zagalsky A, Feliciano J, Storey M-A, Zhao Y and Wang W (2015) The emergence of Github as a collaborative platform for education. *Proceedings ACM Conference on Computer Supported Cooperative Work & Social Computing*. 1906-1917.
7. Angulo MA and Aktunc O (2018) Using GitHub as a Teaching Tool for Programming Courses. *Proceedings ASEE Gulf-Southwest Section Annual Meeting*.
8. Sprint G and Conci J (2019) Mining GitHub Classroom Commit Behavior in Elective and Introductory Computer Science Courses. *The Journal of Computing Sciences in Colleges*. 35(1).
9. Lewis LM, Edwards MC, Meyers ZR, Talbot Jr CC, Hao H and Blum D (2018) Replication Study: Transcriptional amplification in tumor cells with elevated c-Myc. *Elife*. 7:e30274.
10. Sandve GK, Nekrutenko A, Taylor J and Hovig E (2013) Ten Simple Rules for Reproducible Computational Research. *Plos Comput Biol*. 9(10).
11. Peng RD (2011) Reproducible Research in Computational Science. Science. 334(6060):1226-1227.
12. Wilson G et al. (2014) Best Practices for Scientific Computing. *Plos Biol*. 12(1):e1001745.
13. Oliphant TE (2006) A guide to NumPy, (Trelgol Publishing USA).
14. Walt Svd, Colbert SC & Varoquaux G (2011) The NumPy array: a structure for efficient numerical computation. Comput Sci Eng. 13(2):22-30.
15. McKinney W (2010) Data structures for statistical computing in python. Proceedings of the 9th Python in Science Conference. 44551-56.
16. Fiksel J, Jager LR, Hardin JS and Taub MA (2019) Using GitHub Classroom To Teach Statistics. *J Stat Educ*. 27(2):110-119.
17. Beck DA (2020) UWDIRECT/uwdirect.github.io. *Zenodo*.
18. O'Hara K, Blank D and Marshall J (2015) Computational notebooks for AI education. *Proceedings International Flairs Conference*.
19. Wright AM, Schwartz RS, Oaks JR, Newman CE and Flanagan SP (2019) The why, when, and how of computing in biology classrooms. *F1000Research*. 8:1854.
20. Guerra H, Gomes LM and Cardoso A (2019) Agile approach to a CS2-based course using the Jupyter notebook in lab classes. *Experiment International Conference* (exp. at'19). 177-182.
21. DePratti R (2019) Using Jupyter Notebooks in a Big Data Programming Course. *The Journal of Computing Sciences in Colleges*.157.
22. Curtis C and Wolf C (2019) UWDIRECT-2019/batch_issues: batch_issues v1.1. *Zenodo*.
23. Moore JP and Ranalli J (2015) A mastery learning approach to engineering homework assignments. *Proceedings ASEE Annual Conference*. 26.64.21 - 26.64.15.
24. Holland-Minkley AM and Lombardi T (2016) Improving Engagement in Introductory Courses with Homework Resubmission. *ACM Technical Symposium*. 534-539.
25. Peterson LE (2009) K-nearest neighbor. *Scholarpedia*. 4(2):1883.
26. Myers RH and Myers RH (1990) *Classical and modern regression with applications*. Duxbury Press, Pacific Grove, CA.
27. Hunter JD (2007) Matplotlib: A 2D graphics environment. *Comput Sci Eng*. 9(3):90-95.
28. Wilson G (2006) Software carpentry: getting scientists to write better code by making them more productive. Comput Sci Eng. 8(6):66-❒