Experience Paper: A First Offering of Software Engineering

James C. Davis
Purdue University
West Lafayette, Indiana, USA
davisjam@purdue.edu

Paschal Amusuo
Purdue University
West Lafayette, Indiana, USA
pamusuo@purdue.edu

Joseph R. Bushagour
Purdue University
West Lafayette, Indiana, USA
jbushago@purdue.edu

ABSTRACT

This paper describes our first offering of a project-based software engineering course for undergraduate seniors. The course was given to 72 undergraduates, mostly seniors majoring in computer engineering. Our project taught the full engineering cycle with a narrative based on supporting the re-use of software. In two parts spanning 13 weeks, successful teams deployed a web service. We identify lessons learned and opportunities for improvement.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Software and its engineering**;

KEYWORDS

Computing Education, Software engineering

ACM Reference Format:
James C. Davis, Paschal Amusuo, and Joseph R. Bushagour. 2022. Experience Paper: A First Offering of Software Engineering. In *Proceedings of The 1st International Workshop on Designing and Running Project-Based Courses in Software Engineering Education (DREE) (Q-SE 2022).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Course Context

This report describes an offering of the course *"ECE 461: Software Engineering"*.[1] This course is offered as a senior-level course in the Elmore Family School of Electrical & Computer Engineering (ECE) of Purdue University. This course had not been taught since 2011. In the Fall 2021 offering, the content was redone from scratch.

The 72 enrolled students had relatively homogeneous backgrounds. All had fulfilled the prerequisites: two introductory programming courses and one course in data structures, all taught in the C programming language. Many had also taken a course in data science (Python), and some had taken a course in object-oriented programming (Java or C++). Most were seniors majoring in computer engineering, with some juniors. The primary relevant difference between the students was whether they had previously had an internship in software engineering. Based on a sample of 21 students, 16 students (76%) had previously held a software engineering internship.

The course was staffed by one professor, one graduate TA (20 hours/week), and one undergraduate TA (5 hours/week).

1.2 Syllabus and Structure

The course had three learning outcomes. Abridged, they are:

(1) Understanding models of the software engineering process.
(2) Conducting the software engineering process.
(3) Understanding the social aspects of software engineering.

The course followed a typical structure: weekly lectures and homework; four guest speakers; and a project. The project accounted for 60% of a student's grade, with the remainder divided among class participation (25%) and a final exam (15%). Some of this class participation was designed to support the teamwork aspect of the project, *e.g.,* a module on intercultural collaboration [5] and a guest lecture on "*How to Run a Meeting*" by Dr. Greg Wilson.

1.3 Infrastructure

The teams used a GitHub Classroom instance. Each team had $150 in Cloud credits through the Google Cloud Education program. Teams also had access to the department's teaching-focused Linux cluster, which some students used in Part 1 of the project.

2 THE PROJECT

The course project was designed to assess a student's mastery of all of the course learning outcomes. We used a narrative to contextualize the project in a business setting, connecting students' work to software engineering practice to increase their motivation [6].

The theme of the project was to develop tools that support engineers in re-using software. Part 1 focused on social and technical considerations when considering a re-use candidate. Part 2 focused on a system design that would support re-use at scale. Both aspects are important to software engineering work, in which re-use plays a major role [8, 9]. In both parts, student teams had to demonstrate software engineering skills in developing the specified product.

2.1 Timeline

The project was divided into two parts. In each part, most students worked on a team of three, with some teams slightly larger or smaller based on changes in student enrollment. The timeline is given in Table 1. Purdue University has a 16-week semester, and the students were engaged with the project for 15 of those weeks.

2.2 General constraints

Since our students were typically seniors, we provided few constraints (and, consequently, little scaffolding). In particular, they

[1] At time of writing, the course catalog entry is: https://engineering.purdue.edu/ECE/Academics/Undergraduates/UGO/CourseInfo/courseInfo?courseid=402.

Q-SE 2022, May ??-??, 2022, Pittsburgh, PA, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Project timeline. Minimal homework was assigned in the first two weeks. We instructed students to spend this time improving their engineering toolkits (*e.g.,* mastering version control).

Part	Semester week	Event
1	3	Part 1 published
	4	Plan
	5	Milestone
	6	Milestone
	7	Delivery
2	8	Part 1 postmortem; Part 2 published
	9	Plan
	10	Milestone
	11	Milestone
	12	Milestone
	13	Delivery #1
	14	Milestone
	15	Milestone
	16	Delivery #2
	17 (Finals)	Postmortem

were allowed to use the programming language(s) and tool(s) of their choice. They were permitted to re-use snippets from Stack Overflow with attribution, and to re-use modules and frameworks provided that they included them using the package management tool appropriate to their programming language(s).[2] We did require them to use GitHub for configuration management, and to use the Google Cloud Platform for deployment.

2.3 General Structure of Each Part

As indicated in Table 1, the teams prepared a plan in the week after the publication of each project part. In this plan, they were required to: summarize the project requirements and refine ambiguities into an informal specification; identify the tools they planned to use; sketch their designs in UML [4]; outline their validation plan; and propose a timeline. This timeline was divided into (self-defined) weekly milestones, based on Shaffer & Kazerouni's finding that intermediate milestones improve student outcomes [13].

After preparing their plans, the student teams proceeded at their chosen pace, providing updates via weekly milestone reports. If a team departed from its timeline, they discussed corresponding changes in their plan. On each milestone, team members also indicated their contributions and the number of hours they spent. This measurement was intended to promote an equitable distribution of labor by making the efforts of each team member visible.

While most milestones were self-defined, due to the length of Part 2 we specified the details of one intermediate delivery after five weeks. Here, students had to demonstrate (1) some end-to-end working functionality according to their plan; and (2) the use of continuous integration (via GitHub Actions) as well as an update on their use of Continuous Deployment.

One week after delivery, teams submitted a postmortem. These documents asked students to reflect [14] on their projects through the three lenses of planning, process, and product.

[2]In previous courses, students are forbidden to use external resources (*e.g.,* Stack Overflow). This course is their first instructor-approved opportunity to re-use software.

2.4 Project Part 1

This part ran for five weeks, plus one week for the postmortem.

2.4.1 Narrative. Each team was informed that their client, ACME Corporation, plans to adopt Node.js in their web teams. Part of the client's value proposition for using Node.js is the opportunities for re-use afforded by npm. However, ACME Corporation is concerned about several aspects of re-use, including documentation, standards of correctness, and ability to apply security patches in a timely manner. Each team was responsible for developing a tool to help ACME Corporation choose dependencies wisely.

2.4.2 Specification. **Module Metrics**. The project specification describes 6 attributes that ACME Corporation wants to measure. These attributes are ambiguous, *e.g.,* "*ensure that maintainers will be responsive to fix any bugs*". Teams had to operationalize these attributes, *e.g.,* with reference to Munaiah *et al.* 's definitions [10].

Non-functional requirements. The client indicated that the tool should not be slow, it should support multiple levels of log verbosity, and that both the GitHub API and the source code repository must be used as part of the module measurement process. Teams were required to provide evidence of a test suite containing at least 20 tests and achieving at least 80% line coverage.

Interface. A team's software needed to expose a command-line interface, with separate commands for installing dependencies, running a test suite, and measuring a module. Configuration such as the GitHub API token and logging verbosity was done using environment variables.

2.5 Project Part 2

This part ran for nine weeks, plus one week for the postmortem.

2.5.1 Narrative. Each team was informed that their company had lost its contract with ACME Corporation and had gone out of business, and that they had been hired on as engineers to the competitor who won the contract for the next phase of the project. In this part, ACME Corporation has requested a custom module registry to replace npm. This registry should include the ability to score modules using the metrics from Part 1, as well as a web API to support operations such as module upload, enumeration, and download.

2.5.2 Specification. The project specification states that it contains "*far more work than I think you can reasonably do in the time allowed.*" The specification distinguished between baseline features that all teams had to deliver, and extended features from which the teams could pick and choose. In light of the substantial size of this project for 3-person teams, each team had to enumerate and organize the requirements, estimate the cost of each feature, and identify the subset they planned to deliver. The desired cost was roughly 80 hours per person spread over the 10 weeks for this part.

API. The web service needed to provide endpoints for Create-Read-Update-Delete interactions with packages. It could optionally support: a notion of package groups and transactions; a debloat operation to reduce storage costs; and a security audit mechanism.

Each team's web API had to follow an OpenAPI specification to facilitate automated grading. We provided an OpenAPI specification

for the baseline requirements. Teams that implemented additional features were required to update this specification appropriately.

Teams could optionally implement a web browser interface. Such an interface needed to be compliant with the *Americans with Disabilities Act* (ADA), namely via WCAG 2.1 at level AA [3].

Non-functional requirements. In the spirit of the project narrative, teams were required to build on *another* team's Project 1 implementation (randomized exchange).

Teams were required to conduct a security analysis. This analysis used Microsoft's STRIDE framework [2],[3] which applies threat analysis to dataflow models.

Teams could optionally support: *performance*—concurrent interactions with up to 10,000 clients, and report the mean, median, and 99$^{\text{th}}$ percentile latency for a specific usage scenario; *traceability*—tracking all interactions, and supporting a custom monitor-style check on accesses to "sensitive" packages; and *permissions*—users, groups, distinct access permissions, and an authentication endpoint that provides tokens for service access.

Deployment requirements. Teams were required to assess, select, and use appropriate components from Google Cloud Platform (GCP). This was most students' first interaction with a cloud platform. Each team delivered their system as a public GCP endpoint.

2.6 Connection to Learning Objectives

Project Part 1 connected to learning objectives 2 and 3. For *objective 2*, the teams had to negotiate ambiguity in the project specification, design and implement a solution, and validate it based on their plan. As noted earlier, this was also their first foray into implementation-level re-use within our department's software sequence. For both re-use and security, the project specification described the benefits of re-use as well as security concerns that the customer has concerned in the context of re-use. For *objective 3*, the teams had to develop a project plan, identify weekly milestones, and collaborate to succeed.

Project Part 2 connected to learning objectives 1, 2, and 3. For *objective 1*, the project was long enough (9 weeks of implementation) that teams could employ various engineering processes. Some teams opted for a more plan-based approach, with a detailed timeline, up-front component interface specifications, and division of labor. Other teams chose an incremental process, sketching planned features, deferring most to a backlog, and working on 1-2 features per milestone. The connection to *objective 2* was similar to that of Part 1, adding requirements of re-using and extending another team's implementation from Part 1, and of security through both practical issues (configuring authentication in CI/CD and GCP) and systematic analysis (STRIDE). The connection to *objective 3* was similar to that of Part 1.

3 ASSESSMENT

We assessed three aspects of each part: process (~30%), product (~50%), and learning from mistakes (~20%). In most cases, all team members received the same grade on each assessment. To grade teams with substantial disparity in effort, *e.g.,* a teammate who did not attend meetings or contribute code, we used our discretion.

[3]STRIDE is a mnemonic for six classes of security threats: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege.

3.1 Assessing the Process

We evaluated each team's process using two artifacts: their planning and milestone *documents*, and their source code *repositories*. In our assessment of their documents, we checked for teamwork aspects including a team contract and milestones; design aspects such as diagrams (*e.g.,* structural models and dataflow); and the alignment of their validation approach with the project requirements. In our assessment of their repositories, we checked for evidence that they followed their planned process, specifically in terms of code quality standards and in relatively equal contributions across the team (by commits and lines of code per person).

3.2 Assessing the Product

We assessed each team's product using automated tests and manual inspection. We automatically tested for adherence to the functional specification, as well as randomized tests for crashes. We manually inspected a random sample of files from each project for stylistic consistency and implementation-level documentation. To assess the non-baseline (and deliberately ambiguous) elements of Part 2, we examined written documentation in the deliverable report. A team's report described their API, design, limitations of the implementation, validation approach, and provided records of testing. We graded on the completeness of this documentation.

3.3 Assessing Learning from Mistakes

To assess learning from mistakes, we examined each team's post-mortem document. This assessment was generous to account for the range of learning that can be expressed through reflection. If a team identified positive and negative practices, covering their plan, process, and product, then its members received full credit.

4 EXPERIENCES AND LESSONS LEARNED

This section reports on our experiences and lessons learned. Quotes are taken from student responses to our end-of-course survey, which was completed by 21 of the 72 enrolled students.

4.1 Are Weekly Milestones Helpful?

On the whole, we think the weekly milestones were helpful [13]. Six of survey respondents said that the milestones were "*Very relevant: They helped us make considerable progress each week*"; another six were ambivalent ("*So-so: We would have made pretty good progress anyway, so it just felt like paperwork*"); and two said that despite the milestones they ended up doing most of the work close to the submission date. Since almost half of the respondents said they strongly benefited, and since paperwork is part of engineering work, we plan to repeat this aspect of the course.

4.2 The Project Swap

One perennial difficulty in software engineering education is exposing students to the "legacy system" experience [7]. To this end, we had teams build Part 2 on top of another team's Part 1 (cf. §2.5.2). Many students appreciated this, and said things like: "*Seeing other people's code was interesting*", "*The code we got was pretty poorly designed however that just means it would take a bit more time to refactor*", and "*There were some problems getting another team's project to*

run...it was a little bit painful but we will probably experience it in the future". However, the project exchange was mildly controversial, due to variation in the quality of the different Part 1 implementations. One student observed, "*In order to make our project work, we...completely overhaul[ed] the other team's project...groups [who received] poor Project 1s got more work than [other] groups.*" We fear this student was correct — although the project specification allowed teams to reduce the scope of their project based on the state of the project they received, no teams took us up on the offer.

In the next offering of the course, we plan to repeat the project exchange. However, to address the issues that students experienced, we will make two changes. First, in this offering we did not tell the students in advance about the swap, and this may have negatively affected the quality of the code and documentation available to the inheriting teams. Next time we will inform them of this aspect in advance. Second, we will make the project hand-off a structured event, and ensure that the implementations are accompanied by appropriate documentation. We will use one lecture period to provide teams with a synchronous meeting to conduct the hand-off.

4.3 GCP Token Theft and Bitcoin Mining

In Part 2, project requirements caused inadvertent GCP token theft. This certainly met the aspect of learning outcome 2 related to cybersecurity, but we see opportunities to improve.

GCP Token Storage. We required students to implement continuous deployment (CD) to their GCP services. Some students chose to store GCP access tokens directly in their repositories, enabling a fully automated CD process. While we advised them not to do so, this decision was not immediately problematic because we required teams to use private GitHub repositories to discourage plagiarism.

GitHub Action Threshold. We required teams to use GitHub Actions to implement their CI/CD pipeline. Unfortunately, GitHub Classroom limits the number of GitHub Actions that can be made by private repositories, and students began to experience errors. We therefore asked the teams to convert their repositories to public, which removed the Actions cap.

Result: Theft. Upon converting their repositories to public, the teams that stored GCP access tokens insecurely had now leaked them publicly. Three teams informed us that they had been affected. Fortunately, GCP automatically detects anomalous resource usage consistent with Bitcoin mining, and disables accounts before too many resources are exhausted. We are working with one student team to seek reimbursement of a resulting bill.

Discussion. The students got real-world experience in cybersecurity, but we are unsure whether to celebrate this experience. Following the "Swiss Cheese" failure model [12], students saw how a fault (unsound GCP token storage) led to an error (public disclosure of tokens) and subsequent failure (break-in and Bitcoin mining). This is a desirable lesson for students to learn. However, we would prefer to design the lesson to educate students without emotional distress or temporary financial impact, *e.g.,* as a game [11].

In the next offering, we will explore a homework assignment with a "honeypot" repository. This assignment will demonstrate token theft in a controlled manner.

4.4 Assessing Teamwork

The main area for improvement in our assessment is in the assessment of teamwork. Many teams felt that some member was not doing their fair share of the work.[4] We struggled to identify scalable metrics that would allow us to fairly grade team members differently, and eventually opted for a generous interpretation of traditional software contribution metrics (commits, lines of code). However, these metrics do not capture non-coding contributions, *e.g.,* in coordination, design, and research into components to support re-use. We only acted on them in exceptional circumstances.

In the next offering of the course, we plan to make a more structured assessment of teamwork aspects using the Comprehensive Assessment of Team Member Effectiveness (CATME) tool [1].

4.5 Cost-Effectiveness and Sustainability

The course staff did not find it overly costly to supervise this project. We chose to derive most of a student's grade from the project. To help the students (and staff) focus on the project, we set only seven homework assignments, no quizzes, and one final examination. This reduced our grading responsibilities and allowed us to spend most of our out-of-classroom time supporting the project.

We plan to increase the cost of the project in one regard — in-class time. In this offering, we omitted lectures and homework connected to the specific services of GCP, both to preserve the existing lecture sequence and to help students develop self-learning skills. Based on student feedback, we think this may have unfairly benefited the students with prior exposure to Cloud technologies, *e.g.,* through internships. In the next offering we plan to add one lecture on cloud computing and one homework related to GCP, concurrent with the release of Part 2 of the project.

4.6 Transferability

This project is transferable to other instructors or institutions. The project uses only publicly-available resources (*e.g.,* GitHub), with reliance on infrastructure accessible to most educators (*e.g.,* GitHub Classroom; Google Cloud Education).

In designing the project, we did assume that this was our students' first many-week software engineering project, and scoped the project to accommodate teamwork issues. This scope might be too narrow for classes with broader prerequisite coursework.

ACKNOWLEDGMENTS

We are grateful to the students enrolled in Purdue University's ECE 461 in Fall 2021 for their participation in the course. We thank Google Cloud Education for providing cloud credits. We thank GitHub for providing free access to GitHub Education.

DATA AVAILABILITY

A Zenodo artifact is available, with the project specification, our templates for project milestones, submissions, postmortems, and rubrics. See https://doi.org/10.5281/zenodo.5828087.

RESEARCH ETHICS

Purdue University's IRB approved our use of student data.

[4] In fairness, many students were honest in reporting lackluster contributions. On 13 of the 24 teams, there was notable inequity in the hours claimed by the team members.

REFERENCES

[1] 2005. About CATME (Comprehensive Assessment of Team Member Effectiveness). https://info.catme.org/features/overview/

[2] 2006. Uncover Security Design Flaws Using The STRIDE Approach. https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach

[3] 2018. Web Content Accessibility Guidelines (WCAG) 2.1. https://www.w3.org/TR/WCAG21/

[4] Sebastian Baltes and Stephan Diehl. 2014. Sketches and Diagrams in Practice. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Hong Kong China). ACM, 530–541. https://doi.org/10.1145/2635868.2635891

[5] Nicole Hornbrook and James C Davis. 2021. An Intercultural Engineering Module for Software Engineers. In *24th Annual Colloquium on International Engineering Education (ACIEE)*.

[6] Brett D Jones. 2009. Motivating Students to Engage in Learning: The MUSIC Model of Academic Motivation. *International Journal of Teaching and Learning in Higher Education* 21, 2 (2009), 272–285.

[7] Capers Jones. 2006. The Economics of Software Maintenance in the Twenty First Century.

[8] Charles W Krueger. 1992. Software Reuse. *ACM Computing Surveys (CSUR)* 24, 2 (1992), 131–183.

[9] Mike Loukides. 2021. *Thinking About Glue – O'Reilly*. https://www.oreilly.com/radar/thinking-about-glue/

[10] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating GitHub for Engineered Software Projects. *Empirical Software Engineering (EMSE)* 22, 6 (2017). https://doi.org/10.1007/s10664-017-9512-6

[11] James Parker, Michael Hicks, Andrew Ruef, Michelle L. Mazurek, Dave Levin, Daniel Votipka, Piotr Mardziel, and Kelsey R. Fulton. 2020. Build It, Break It, Fix It: Contesting Secure Development. *ACM Transactions on Privacy and Security* 23, 2 (2020), 1–36. https://doi.org/10.1145/3383773

[12] J Reason, E Hollnagel, and J Paries. 2006. Revisiting the Swiss Cheese Model of Accidents. *Journal of Clinical Engineering* 27, 4 (2006), 110–115.

[13] Clifford A. Shaffer and Ayaan M. Kazerouni. 2021. The Impact of Programming Project Milestones on Procrastination, Project Outcomes, and Course Outcomes: A Quasi-Experimental Study in a Third-Year Data Structures Course. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (Virtual Event USA). ACM, 907–913. https://doi.org/10.1145/3408877.3432356

[14] Jennifer Turns, Brook Sattler, Ken Yasuhara, Jim Borgford-Parnell, and Cynthia Atman. 2014. Integrating Reflection into Engineering Education. In *2014 ASEE Annual Conference & Exposition Proceedings*. 24.776.1–24.776.16. https://doi.org/10.18260/1-2--20668