Engineering Educators Bringing the World Together
2025 ASEE Annual Conference & Exposition
Palais des congrès de Montréal, Montréal, QC • June 22–25, 2025 ASEE
Paper ID #48610

DUE: Integrating Performance Engineering in Software Engineering Education: A Multi-Course Project Approach

Dr. Lu Xiao, Stevens Institute of Technology (School of Engineering and Science)

Dr. Lu Xiao is an Associate Professor in the Department of Systems and Enterprises at Stevens Institute of Technology. Her research focuses on software engineering, particularly software architecture, software economics, cost estimation, and software ecosystems. Dr. Xiao has received several National Science Foundation grants, such as the CAREER award for developing an AI-empowered architecture-centric framework for systematic software performance optimization. She actively contributes to the academic community through roles in organizing committees and program committees for conferences like ICSE and ASE. At Stevens, Dr. Xiao teaches courses in software engineering and plays a significant role in academic service, including serving on curriculum committees and initiatives to enhance student-faculty interaction.

Yu Tao, Stevens Institute of Technology (School of Humanities, Arts, and Social Sciences)

Yu Tao, Ph.D., is an Associate Professor of Sociology at Stevens Institute of Technology. Her research analyzes issues related to human resources in science, technology, engineering, and mathematics (STEM) as well as online privacy and fair privacy from the sociological perspective.

Dr. Andre Benjamin Bondi, Stevens Institute of Technology (School of Systems & Enterprises)

André Bondi has worked on performance modeling and engineering problems in many problem domains, including industrial monitoring, railway signaling, and telecommunications. He received the Computer Measurement Group's A. A. Michelson Award in 2016. His book, Foundations of Software and System Performance Engineering, was published by Addison-Wesley in 2014. He is an independent consultant and has served as an adjunct professor of software engineering or senior research scientist at Stevens Institute of Technology since August 2019. Until October 2015, he was a Senior Staff Engineer working in software performance at Siemens Corp., Corporate Technologies in Princeton, New Jersey. He previously held senior performance positions at two startups and was a Principal Technical Staff Member at AT&T Labs and its predecessor AT&T Bell Labs. He was a visiting professor at the University of L'Aquila in 2016. Dr. Bondi holds a Ph.D. in computer science from Purdue University, and an M.Sc. in statistics from University College London.

Eman Abdullah AlOmar, Stevens Institute of Technology (School of Engineering and Science)

Eman Abdullah AlOmar is an Assistant Professor at Stevens Institute of Technology. She completed her Ph.D. in Computing and Information Sciences at Rochester Institute of Technology in 2021. Her research interests lie at the intersection of software engineering and artificial intelligence with a focus on different software engineering areas such as software maintenance, software evolution, software refactoring, technical debt, software quality assurance, code review, and documentation. She has received four Best Paper Awards and Best Presentation Awards at IWoR 2019, MSR 2022, MSR 2024, and SIGCSE 2024. She has also received the Distinguished Doctoral Research Award at MSR 2023, and Best Reviewer Award at JSS 2022 and JSS 2023. Her collaborations with national and international researchers and industry leaders have resulted in ACM and IEEE publications in leading software engineering platforms.

©American Society for Engineering Education, 2025

RUE: Integrating Performance Engineering in Software Engineering Education: A Multi-Course Project Approach

Lu Xiao, Andre Bondi, Eman Abdullah AlOmar, Yu Tao
lxiao6, abondi, ealomar, ytao@stevens.edu
Stevens Institute of Technology

1 Introduction

Performance considerations are often an afterthought in software development. Software engineering and computer science courses frequently emphasize design, implementation, and delivery over quality attributes such as performance, dependability, reliability, and security. This leaves many graduates with limited exposure to performance modeling and analysis[4,1].

This project was conceived to explore ways to encourage students to think about performance considerations and concerns throughout the software life cycle by incorporating performance topics into existing software engineering courses. By doing so, we seek to reduce the risk that performance considerations will be an afterthought in the software development life cycle after graduation. Note that these topics do not constitute a performance curriculum in and of themselves, because they do not include the mathematics of performance modeling, the input and output analysis of simulations, or workload characterization. Rather, we aim to stimulate awareness of performance as an integral cross-cutting concern in the software development.

Our effort focuses on adding performance content to four courses that cover different facets of software development. First, the C200 Introduction to Object-Oriented Programming course covers basic programming and fundamental data structures, and object-oriented design. The next course, C300 Software Modeling and Simulation, explores formal models for specifying, designing, and automatically analyzing software systems. The third course, C400 Software Metrics and Estimation, introduces students to the principles and practices of software metrics and estimation, equipping them with tools to manage project scope, effort, and delivery timelines effectively. Lastly, C500 Software Testing and Quality Assurance covers testing methods, such as unit testing, test-driven development, mocking, and test automation.

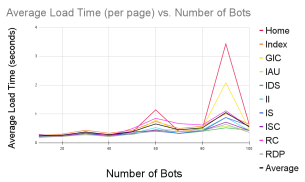
2 C200: Cultivating Performance Awareness in Object-oriented Programming

We designed a new module to highlight software performance engineering concepts and design an assignment that focuses on the performance comparison between an object-oriented program and a procedural program. Students are tasked with developing and implementing two equivalent solutions to the same problem: object-oriented design with inheritance and procedural programming with if-else conditions. The project focuses on an emerging pharmacy DWT (similar to CVS). Some of these requirements are defined as follows: (1) the pharmacy offers a variety of standard vaccinations, such as flu, COVID, chicken pox, whooping cough, etc; (2) each type of vaccination should specify its target demographic (3) some vaccinations should be offered in multiple doses with a specific period in between doses, (4) different types of vaccination may have different insurance eligibility, (5) the system should keep a record of the prior vaccination status of the patient, and (6) the system should allow a patient to make a reservation for vaccination appointment based on their preferences and the availability of the vaccine and a technician to administer it. Students then measure the execution time for each implementation to determine which is more efficient. The objective is to help students understand the trade-off between performance and maintainability introduced by object-oriented design.

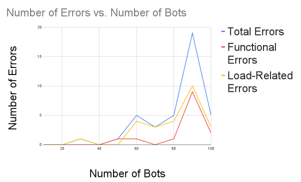
3 C300: Cultivating Performance Awareness in Software Modeling and Simulation

We incorporate an interactive module using eight Unity games, collectively called eFish'nSea[3], whose snapshots in Figure 1a. Each game metaphorically represents one of the eight recurring performance issues revealed in prior research[5]. Game 1: Loopy Fish highlights how unnecessary loop iterations can waste resources and time. Game 2: Macaroni Munchies demonstrates how poor coordination among concurrent processes leads to waiting and blocked resources. Game 3: Fishing Frenzy shows how systems that handle "normal" input efficiently can suffer under edge or rare inputs. Game 4: Fantastic Toast explores how selecting suboptimal APIs—or "tools"—can drastically hurt performance. Game 5: Moving Day demonstrates the impact of choosing ill-fitting data structures (e.g., overly large or small "boxes"). Game 6: Crabsworth's Cave reveals how failing to "cache" or store previously computed results can lead to unnecessarily re-computing the same work. Game 7: Load 'Em Up emphasizes that repeatedly processing the same data in tiny increments (e.g., many partial "shipments") inflates overhead. Lastly, Game 8: Shrimplock Holmes represents any remaining algorithmic inefficiencies that do not fit the other categories—such as an unfavorable sorting or search approach. The course also incorporates a simulation-based load testing assignment that uses bots—rather than real human users—to model high-traffic conditions. Students gradually scale up the number of simulated users and capture how response times, error rates, and system stability deteriorate at various load thresholds. Figures 1b and Figure 1c, shared with students, provide an example analysis.

(a) Game Overview



(b) Load Time



(c) Errors

Figure 1: C300 Module

4 C400: Cultivating Performance Awareness in Software Metrics and Estimation

At the core of the new module are two related concepts: performance issue reports and return on investment (ROI) analysis. Performance issue reports document system inefficiencies such as slow response times, excessive resource consumption, or poor scalability. Zhao et al. proposed the use of heuristic linguistic patterns and NLP techniques to identify performance-related issue reports[6]. The second foundational concept is ROI analysis, which evaluates the trade-offs between the investment required to address performance issues and the benefits gained from their resolution[5]. Investment is approximated using proxies such as the number of developers involved in issue discussions and the volume of associated comments. Returns are quantified through performance improvement factors, such as reduced response times or increased throughput. The module features an assignment designed to immerse students in the practical aspects of performance analysis in two phases: 1) performance issue tagging and 2) the ROI analysis using proxy metrics. We provide each student with up to 1000 issue reports from an Apache project. Students first review issue reports and apply linguistic pattern-based heuristics and NLP techniques to identify performance-related reports. Next, students conduct the ROI analysis using the proxy metrics for “investment” and “return” on fixing performance issues. Given the estimated percentage of performance issues in a real-world environment, each student analyzes about 60 to 100 performance issue reports.

5 C500: Cultivating Performance Awareness in Software Testing

We created the Machine-Readable Travel Document (MRTD) project, which brings together different testing techniques while stressing how important performance testing is in real-life settings. An MRTD must display information for both visual and OCR-based inspection. It typically consists of two parts: a Visual Inspection Zone (VIZ) and a Machine-Readable Zone (MRZ). The MRZ is split into two lines: the first line contains the document type (e.g., passport), issuing country, and holder’s name; the second line includes the passport number, country code, birth date, gender, expiration date, and a personal number. For this project, students develop a system that decodes both MRZ lines to validate the check digits and report any mismatches. They also implement an encoding feature that converts fields, such as the traveler’s name and date of birth, back into the MRZ format, inserting the correct check digits. We guide the students in accomplishing this project in five parts. Part 0: Project Planning: Students begin by creating a Gantt chart outlining project tasks, timelines, and dependencies. Part 1: Requirement Analysis: Next, students study the MRTD specification[2] and project requirements, identifying ambiguities such as unclear performance metrics. Part 2: Implementation and Unit Testing: students develop and test the encoding and decoding algorithms. They adopt test-driven design, mocking for hardware and database components, and apply mutation testing to gauge how thoroughly their test suite detects potential errors. Part 3: Performance Measurement: The students then measured how efficiently their algorithms handle large inputs containing 10,000 MRTD records, one encoded and one decoded. They record execution times and plot performance results with increasing input sizes. Part 4: Test Planning in (Hypothetical) Practice: Finally, students produce a comprehensive test plan, factoring in budget constraints, evolving requirements, and user needs.

6 Evaluation

Evaluation of C500 Module A survey was conducted among 40 students enrolled in the course (30 undergraduates and 10 graduates), with eight complete responses analyzed.

All five components of the project received high average ratings on a 5-point scale. Students rated Part 0 and Part 2 the highest, with averages of 4.25. Part 1 and Part 4 followed closely, each averaging 4.13. While Part 3 was rated slightly lower, it still achieved a respectable average of 3.75. In applying unit testing techniques, such as mocking and mutation testing, five students reported significant improvements of their confidence. Similarly, six students noted greater confidence in connecting various testing techniques, including functional and performance testing. However, confidence in integrating performance analysis was more variable; four students showed improvement, but two reported no change, and another two experienced a decrease. This mixed result indicated that performance testing posed challenges that warrant further attention. Students provided insightful feedback on the project. Many emphasized the value of project planning, particularly the Gantt charts. Others highlighted the importance of clarifying ambiguous requirements during the analysis phase. While most participants found this exercise valuable, some struggled to integrate performance analysis into their testing strategies.

Evaluation of C400 Module We used two surveys to gather students' feedback: one focused on the eFish'nSea game set and the other on the stress testing simulation assignment. Five students (among 20) participated and completed the survey of the games. The survey suggested that the majority of students employed the correct strategies in playing the games, showcasing their understanding of performance-related concepts. In most of the games, the majority (75% to 100%) of students applied the correct strategies to play the games—suggesting correct understanding of the intended performance concepts. While in Macaroni Munchies, 50% applied the correct strategy. This game, which emphasizes synchronization as a critical aspect of performance, suggested challenges for students. On a 5-point scale, students rated the games' general usefulness for understanding software performance with a mean score of 3.6, indicating moderate to high effectiveness. The games were perceived to positively contribute to programming abilities, particularly in performance optimization, with a mean score of 4.0. Students provided a mean score of 3.4 when evaluating the likelihood of recommending the games to others.

Eight students completed the survey of the simulation assignment. The survey shows that 87% of participants correctly identify relevant performance metrics, such as response time and throughput. Meanwhile, 62.5% of participants correctly avoided irrelevant metrics. These results indicated that students generally demonstrated a commendable understanding of relevant metrics but faced challenges distinguishing between performance-oriented and non-performance-oriented criteria. Students rated the simulation assignment highly in enhancing their understanding of performance concepts. On a 5-point scale, the assignment received a mean score of 4.25.

Future Evaluation Plan In C400, the evaluation will center on students' ability to analyze and apply software metrics effectively on performance reports. In C200, assignments will assess students' ability to identify and explain performance differences arising from different design paradigms. To complement these assignments, pre-and post-course surveys will gauge changes in students' confidence and competence in applying performance concepts. We also plan to iteratively evaluate and improve all four course modules with more future sessions.

7 Conclusion

Our goal in this research is to augment some existing courses with related performance topics to imbue students with the notion that performance should be considered at all stages of the software development life cycle. We enhanced lectures with performance-related material and offered related assignments. We evaluated students' perceptions of their performance awareness using questionnaires and by reviewing their work on assignments that have performance content. We found that careful guidance must be given on the form that their reports on performance tests should take, as one cannot take for granted that students, especially undergraduates, have already had sufficient experience with quantitative data to write reports on quantitative data that are cogent, informative, and succinct. This concern is analogous to the need to write unambiguous requirements for developers.

Acknowledgment

We acknowledge and accept the plagiarism and copyright policy of ASEE. This work was supported by the U.S. National Science Foundation under grants CCF-2044888 and DUE-2142531.

References

[1] A. B. Bondi. 2014. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice*. Addison-Wesley.

[2] International Civil Aviation Organization (ICAO) 2021. *DOC 9303, Machine Readable Travel Documents Part 3: Specifications Common to all MRTDs Eighth Edition, 2021*. International Civil Aviation Organization (ICAO), Montreal, PQ.

[3] Andrew Quinlan, Ryan Mercadante, Vincent Tufo, Jonathan Morrone, and Lu Xiao. 2024. eFish'nSea: Unity Game Set for Learning Software Performance Issues Root Causes and Resolutions. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*. 342–347.

[4] Connie U Smith and Lloyd G Williams. 2003. Best practices for software performance engineering. In *Int. CMG Conference*. 83–92.

[5] Yutong Zhao, Lu Xiao, Andre B Bondi, Bihuan Chen, and Yang Liu. 2022. A Large-Scale Empirical Study of Real-Life Performance Issues in Open Source Projects. *IEEE Transactions on Software Engineering* 49, 2 (2022), 924–946.

[6] Yutong Zhao, Lu Xiao, and Sunny Wong. 2024. A Platform-Agnostic Framework for Automatically Identifying Performance Issue Reports with Heuristic Linguistic Patterns. *IEEE Transactions on Software Engineering* (2024).