Early Progress on Enhancing Existing Software Engineering Courses to Cultivate Performance Awareness

André Benjamin Bondi
Stevens Institute of Technology
Hoboken, New Jersey, USA
Software Performance and Scalability Consulting LLC
Red Bank, New Jersey, USA
andrebbondi@gmail.com

Lu Xiao
Stevens Institute of Technology
Hoboken, New Jersey, USA
lxiao6@stevens.edu

ABSTRACT

Software engineering and computer science courses are frequently focused on particular areas in a way that neglects such cross-cutting quality attributes as performance, reliability, and security. We will describe the progress we have made developing enhancements to some of our existing software engineering courses to draw attention and even lay the foundations of an awareness of performance considerations in the software development life cycle. In doing so, we wish to make performance considerations integral to the software engineering mindset while avoiding the need to remove current material from our existing courses. This work is part of an NSF-funded project for undergraduate curriculum development.

CCS CONCEPTS

• **Software and its engineering** → *Software organization and properties*; **Software performance**; • **Social and professional topics** → *Software engineering education.*

KEYWORDS

Performance engineering. Software engineering curriculum development.

ACM Reference Format:
André Benjamin Bondi and Lu Xiao. 2023. Early Progress on Enhancing Existing Software Engineering Courses to Cultivate Performance Awareness. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3578245.3584352

1 INTRODUCTION

In the first author's industrial experience, and in that of other authors, e.g., [20], the performance of a software system is often an afterthought rather than a quality attribute to be considered at every stage of a software development life cycle. The importance of an understanding of performance concerns among all stakeholders is underscored by a study by Bass *et al* [3], which indicated that performance failures were the single largest cause of software project cancellations. It is for these reasons that we feel it is important to cultivate performance awareness among students taking software engineering and computer science courses.

In our various roles as an industrial practitioner of software performance engineering and as students and instructors of computer science and software engineering courses, we have observed that performance awareness is far from ubiquitous, and that performance concerns and means of addressing them receive little if any attention in standard courses and texts.

The goal of our project is to show how performance concerns and concepts can be incorporated into standard courses without taking large amounts of class time and homework time away from the courses' core subject matter. For example, we have augmented course material on testing concurrent programming with explanations of how performance measurement can be used to diagnose concurrent programming problems. Working with colleagues in engineering education development (Dr. Gail Baxter) and sociology (Dr. Yu Tao), we plan to evaluate how our course enhancements have increased performance awareness by comparing course sections who hear about them with those who do not.

2 RELATION TO PREVIOUS WORK

Bondi and Saremi [8] describe how they augmented an existing course on functional testing and quality assurance with material on resource usage measurement, web response time measurement, and performance requirements. We describe augmentations to other aspects of the course in a subsequent section.

Some textbooks on operating systems [9] and computer networking have a considerable performance component [5]. Somerville [22] briefly mentions performance concerns as they relate to testing and to the impact of performance requirements on architecture.

Many books on performance engineering place a stronger emphasis on performance measurement [18] and analysis and modeling [15][13], its application to specific system problems [11][17][19], and the role of performance engineering in the software development life cycle [6] than we would have time to do in any or all of the software engineering courses we explore here. The material we propose to add to the standard courses might pique students' interest in learning about performance in depth, but it will not give the students the same breadth or depth of performance knowledge as a course based on any one of these books. [1]

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04…$15.00
https://doi.org/10.1145/3578245.3584352

[1] The operating systems class taught by Prof. Peter Denning at Purdue University in the late 1970s and early 1980s was an inspiring example to the first author. Apart from devoting a few lectures at the end of the course to operational analysis and basic queueing network models, Prof. Denning drew attention to the effects on performance

3 ADDING PERFORMANCE CONTENT TO EXISTING COURSES AT STEVENS

3.1 Objectives and Constraints

When adding performance content to existing courses, we need to ensure that the new material fits into the instructor's existing narrative flow while not detracting from the usual topics of the course. At the same time, we must ensure that the performance content is not oversimplified. Moreover, we need to reinforce the performance content by providing exercises that augment the homework assignments that support the usual goals of a given course. In the present project, we are examining ways to do this for four courses, two of which are dual senior/first year graduate courses, and two of which are senior-level or junior-level undergraduate courses. The titles of these four courses are Testing and Quality Assurance, Software Metrics, Model-Based Design, and Object-Oriented Development.

Unlike courses in the social and natural sciences, most computer science and software engineering courses do not stress probability, statistics, or the analysis and presentation of quantitative data. Because we cannot take too much class time away from the intended topics of our courses, we either have to rely on students' prior training in quantitative disciplines or give firm and detailed guidance about how quantitative data should be presented in their homework assignments. This is consistent with the sentiments expressed in discussion groups at the First International Workshop on Teaching Performance Analysis of Computer Systems, part of the Performance 2021 conference. Videos of these discussions may be found here: https://www.performance2021.deib.polimi.it/teapacs/ .

3.2 A Course on Functional Testing and Quality Assurance

Because the authors taught this course in the semester preceding the present workshop, it was the first on which we focused our enhancement efforts. All programming exercises in this course are done in Python. This enables the class to use a vast number of libraries and software tools without charge.

In [8], we described the extension of a course on functional testing and quality assurance with the use of JMeter to drive web loads and the use of native resource usage measurements on their laptops to discern how an application actually works. In the present course enhancements, we also show the students how to build a performance test plan from performance requirements and basic performance models such as those described in [10] and [6]. This material was preceded by a lecture on using functional requirements as the basis of functional test plans. These two practices are mutually reinforcing. Indeed, one could augment each functional requirement with a corresponding performance requirement and a set of workload requirements to build a performance test plan.

Among the other performance-related topics we cover in this course are:

- The planning of performance tests to establish whether a refactoring effort has improved or degraded performance, thus mitigating the risk that the refactoring could degrade performance,
- The opportunity to identify the existence of serious performance issues if unit tests in single user mode take longer than desired, using standard timestamp libraries,
- Using a discrete time Markov chain model to drive load tests on web sites [7]. This technique builds on the concept of functional tests that are driven by state machines. These can be used to track code coverage [16],
- The use of orthogonal array testing (OATS) to reduce the number of test cases, and hence the execution time of integration or unit testing during builds,
- The use of Python timestamp libraries to measure the execution times of the various stages of a build, so that students understand which steps take the longest and how the steps might be optimised.

Notice that each of these performance topics ties back to at least one existing course topic, e.g., functional testing, configuration management, or requirements-driven testing. Thus, the performance topics reinforce the teaching of the original concepts in the testing course.

3.3 Model-Based Software Engineering

This is a senior-level course. The textbook is [20]. The course provides a survey of the role of UML in software design. The course already has substantial performance content, including basic queueing network models, the utilization law, the forced flow law, Little's Law, and bottleneck analysis as described in [10]. It also contains lectures on discrete time and discrete event simulation and the use of execution graphs with time estimates to estimate code execution times [21]. Performance awareness might be enhanced by the addition of topics such as the performance characteristics of user interfaces and the use of swim lanes in UML sequence charts to coarsely identify potential foci of overload. A senior completing this course in its present form would have some awareness of performance in systems. The Stevens software engineering course on object-oriented development is a prerequisite for this course. We consider it next.

3.4 A Course on Object-Oriented Development

This is a Java-based undergraduate course. It covers object classes, inheritance, exception handling, and elementary data structures. It describes the differences between linked lists, arrays, strings, trees, stacks, and queues. The costs of insertions, deletions, searching, and sorting on these are described using big-Oh notation, as in a typical undergraduate data structures course.

There are several opportunities for making the students aware of performance considerations in this course. For example,

- We can assign hypothetical processing and I/O costs to basic operations and then show how these can be mapped to resource utilization. Similarly, we can examine the spatial costs of the various algorithms and implementation choices and show how these would map to memory consumption. This will reinforce student awareness of the resource usage attributable to algorithmic choice and implementation.

of scheduling rules, context switching, the use of semaphores instead of locks, concurrent programming principles, and page replacement algorithms. Thus, students were taught to make connections between overall system performance and the performance of various aspects of operating system implementation[9][12].

- We would assign the students to implement a simple numerical algorithm such as Gaussian elimination in Java code, and then have them measure the elapsed time to solve a very large system of linear equations using interpreted Java code, compiled Java code, Java code running under JIT compilation, and a package from a numerical library implemented in compiled C code.
- We would have the students explore how repeated insertions, creations, and deletions of objects could precipitate Java garbage collection, which can degrade the performance of an application.
- Similarly, one could use rudimentary models to examine the performance trade-offs between using persistent objects that are kept in a pool of constant size and transient objects that are only kept for as long as they are needed. One could also discuss the operational risks of repeated object creation and deletion, such as memory leaks resulting from the failure to destroy objects when no longer needed. For example, memory occupancy could expand to the point where it could cause object pool exhaustion, extra paging, garbage collection, poor locality of reference, and security problems such as violating addressing restrictions. Recursions that do not terminate could cause run time stack overflows leading to a system crash. If the students are running code in a linux environment, we could ask them to track the occupancy of the swap space using a program that has been deliberately salted with a memory leak.

3.5 A Course on Software Metrics

This course is primarily concerned with metrics quantifying code complexity, maintainability, software reliability, and how they can be used to estimate development effort. Early in the course, the point is made that it is important to choose measurements and metrics that are informative about the questions at hand and that tell an actionable story. This is an important element of performance awareness. Among the opportunities for cultivating performance awareness are:

- The response times of individual transactions and use cases under light and heavy loads could be examined, and perhaps related to code complexity metrics.
- The results of code execution profiling could be compared with execution graphs to see if the code segments where the program is spending its time can be related to cyclomatic complexity, the presence of bugs, and the cost of exception handling.
- The Return on Investment (ROI) of addressing performance issues in real-life software projects. In our prior work [27, 28], we measured the "return" of fixing performance issues as the improvement factor of the performance metrics, such as response time and throughput, as well as the "investment" as indicated by the number of discussions among developers before they address the performance issues. We will design an assignment for students to practice the ROI analysis on selected, real-life performance issues to understand the "economics" behind managing performance issues in software projects.
- The effect of code size on performance and resource usage and the sizes of the code and data segments could influence memory occupancy, paging behavior, and cache hit ratios. These could also be impacted by measures of modularity, function call frequency, and run-time stack activity. Before incorporating this in to the course, one would have to determine whether the level of detail required is excessively burdensome to the students.

4 GENERAL OBSERVATIONS

4.1 Data Presentation and Analysis

The presentation, communication, and interpretation of measurement data are essential elements of performance awareness. During our pilot efforts, we have noticed that quality of data presentation among students is far from uniform. To prevent this, we should explain sound data presentation practices to our students, including conventions for displaying plots, and require that students adhere to them. Examples of good data presentation and visualization practice can be found in [24] and [14]. Among the problems students need to be trained to avoid are:

- Graphs that have axes with no labels and curves with no legends.
- The absence of data point markers from plots. This prevents us from seeing deviations from fitted curves and from determining how the points are distributed along the X axis.

4.2 Choosing Meaningful, Informative Performance Metrics

One of the necessary conditions for performance awareness is the ability to determine whether a metric or measurement is informative about questions at hand or questions that might arise. In addition, students need to be aware that performance concerns can be associated with more than one metric. For example, a student who is performance aware should understand that processing is not always the cause of high response times, and that high response times could be attributed to excessive I/O, wasteful bandwidth usage, or software bottlenecks.

4.3 Requiring Students to Generate Performance Measurements of Their Own Code and AI-Generated Code

It is well known that many students began using *chatGPT* to prepare programming and essay assignments within days of the application's release [2]. This concern is also discussed in [25] and [4]. The implications of this for evaluating students' work are outside the scope of this paper. Still, whether the programs students submit are their own or automatically generated, we can require them to measure their execution times and, if run on a laptop, their resource usages, for various sets of input parameters and input files that could be generated at random or chosen by the instructor. At the time of writing, we are not aware of *chatGPT* or *copilot* being able to generate performance measurements of running programs, including those created by students. Students might be able to learn something from an exercise in which they have to measure programs generated by an AI tool as well as each other's programs

intended to accomplish the same task. Since measurements will typically be run on students' own machines, they might not be easy to fake. This requires further investigation.

5 NEXT STEPS

We will now discuss the next steps in our longer-term plan that are not elaborated in the previous sections. These include opportunities to incorporate performance topics in additional courses in our software engineering program, the plan to evaluate the effectiveness of the learning outcomes of the proposed curriculum changes, as well as the plan to disseminate the results of this project to increase the broader impacts in the community.

5.1 Homework Assignments to Stimulate Awareness

Homework assignments containing performance content are necessary to reinforce students' awareness of performance issues. These must be tailored to the principal subject matter of each course.

- In any of the courses with substantial programming content, with the possible exception of the course on software metrics, one could require students to insert calls to functions that deliberately slow programs down, and have the students compare the resource usage and execution times with and without these degrading calls.[2]
- In the course on functional testing, we require students to measure and compare the execution times of programs when reading and processing very small, moderate, and very large files in identical formats. We also require the students to compare the performance and resource usage of test harnesses with and without mutation test cases [1].
- In the course on object-oriented development, examples of refactoring might include the substitution of one sorting algorithm by another and the interchanging of the nesting order of loops in a numerical computation. We might also consider the possibility of students running their own programs through a tool to detect performance antipatterns, such as that proposed in [23], but this might entail making load generation tools available to students and enabling them to generate loads in a way that does not compromise any systems they do not own.
- Performance-related homework assignments in the course on model-driven design could include exercises on the basic queueing laws and their application to the planning of simulations and the analyses of their outputs. In particular, the Response Time Law for closed queueing networks could be used to show how long think times can drive down simulated throughputs.

5.2 Other Curriculum Additions

In each of the courses mentioned in the previous subsection, we focus on specific topics about software performance, such as choice of testing techniques, the specific modeling technique, and detailed design choices to ensure software performance. Because these topics are specific to each course, students will not gain a holistic experience of how they should care for performance during the entire life-cycle of a project, from inception to project delivery, as well as during following-up maintenance. Thus, we are planning on further curriculum additions to the project-based courses so that students can practice using software performance methods as an integral part of the development life-cycle in their own projects. Specifically, during their last two semesters, our undergraduate students take a sequence of two senior level design courses, where they propose, design, implement, test, and deliver a project of their own choosing. In the current setting of the course project, students mostly emphasize the functionality of their projects, without being asked to explicitly plan, design, and manage its performance. We plan to add more probing requirements in the different milestones of the project so that students would be guided to consider the performance of the project more systematically and formally. The objective is to let the students connect the "dots" they learned from their prior courses into "lines" through this two-semester-long course project.

5.3 Evaluation of Learning Outcomes

We plan to evaluate the proposed curriculum materials using both objective and subjective measures; and iteratively improve the curriculum materials based on the evaluation results.

On the one hand, we plan to evaluate the outcomes of each course based on students' subjective input. That is, we plan to provide an entering survey and an exiting survey to students in each course. The survey aims at collecting subjective evaluations of the course materials from students. For example, for the Software Testing course, we may ask the student to evaluate on a scale from 1 to 5 how well prepared are they in conducting certain testing tasks that relate to software performance. The survey questions will be customized based on the topics covered in the course. To specifically evaluate the effectiveness of the proposed curriculum materials, we plan to conduct the survey with the class that took the original course (i.e. before the proposed curriculum materials were incorporated), and also the class that actually take the updated course (i.e. after implementing the proposed curriculum materials). The effectiveness of the course materials will be measured based on the improvement made by comparing the baseline class vs. the target class.

On the other hand, we also plan to evaluate students' learning outcomes based on objective measures, through quizzes and assignments. For example,

- We could ask the students to compute resource utilizations given service time and throughput without mentioning the utilization law, and compute attainable throughput given the response time, think time, and number of load drivers, or given the demand on the most heavily loaded resource.
- We could ask the students to draft response time requirements for different kinds of applications.

Tasks like these will be designed to test students' knowledge of relevant performance topics that we aim to cover. The effectiveness of the curriculum materials will be measured by comparing the scores of students in the baseline class vs. the target class.

We will develop improvement plans based on the above subjective and objective evaluation results.

[2]This idea is inspired by a comment by one of the WEPPE2023 referees.

5.4 Dissemination of Results

Our overarching goal is to increase the impact of this project on the broader community in software engineering and performance engineering. We plan to focus on the following activities for this goal:

- Making the curriculum modules we create in this project available in a public repository to benefit the broader community. This was suggested by our advisory board prior to the commencement of the project.
- Holding workshops to inform and engage instructors in software engineering, computer science, and performance engineering, to facilitate its adoption elsewhere, and to obtain feedback and suggestions from them on how to improve our work.

6 CONCLUSIONS

The goal of our research is to show how existing undergraduate courses in software engineering could be enhanced to promote the cultivation of performance-aware software engineers. We have presented an outline for adding performance topics to four courses, and have explored the possibilities of doing so in detail in our senior/graduate level course on functional testing and configuration management. While doing so, we have identified the need to provide firm guidance to students on the graphical presentation of performance measurement data. A data presentation standard is necessary for communicating performance insights and concerns to a variety of stakeholders in the software development life cycle. We have also outlined how we might evaluate the effectiveness of our curriculum enhancements, so that we and others can see how to improve them.

ACKNOWLEDGMENTS

This work is supported by NSF grant #2142531. The goals of this project were first described publicly at an invited panel discussion hosted by the Computer Measurement Group at CMGIMPACT2022 [26]. The original funding applicant and principal investigator for this project was Ye Yang, who is now at Amazon Web Services. We thank her for initiating this project and for guiding the research proposal process from its inception to its funding. The second author is now the principal investigator. We would also like to thank the referees for their useful comments.

We would like to especially thank the members of this project's advisory board for their valuable comments. The members are Alex Podelko, Connie U. Smith, Kishor Trivedi, Igor Trubin, and Gregg Vesonder.

REFERENCES

[1] Domenico Amalfitano, Ana C.R. Paiva, Alexis Inquel, Luís Pinto, Anna Rita Fasolino, and René Just. 2022. How do Java mutation tools differ? *Commun. ACM* 65, 12 (2022), 74–89.
[2] Laura Meckler and Pranshu Verma. 2022 (accessed December 28th, 2022). Teachers are on alert for inevitable cheating after release of ChatGPT. *The Washington Post* (2022 (accessed December 28th, 2022)). https://www.washingtonpost.com/education/2022/12/28/chatbot-cheating-ai-chatbotgpt-teachers/
[3] Len Bass, Robert Nord, William Wood, and David Zubrow. 2007. Risk Themes Discovered through Architecture Evaluations. In *2007 Working IEEE/IFIP Conference on Software Architecture (WICSA'07)*. 1–1. https://doi.org/10.1109/WICSA.2007.37
[4] E. Berger. 2022. Coping with copilot. https://blog.sigplan.org/2022/08/18/coping-with-copilot/
[5] Dimitri Bertsekas and Robert Gallager. 1987. *Data Networks* (2nd ed.). Prentice Hall.
[6] A. B. Bondi. 2014. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice.* Addison-Wesley.
[7] Andre B. Bondi and Apeni Lotha. 2019. Building a representative, effective model to randomly generate valid sequences of web page visits for load testing. In *CMG/IMPACT 2019*. Computer Measurement Group.
[8] Andre B. Bondi and Razieh L. Saremi. 2021. Experience with Teaching Performance Measurement and Testing in a Course on Functional Testing. In *ICPE '21: ACM/SPEC International Conference on Performance Engineering, April 19-21, 2021, Companion Volume*. ACM, 115–120.
[9] E. G. Coffman and P. J. Denning. 1973. *Operating Systems Theory*. Prentice Hall.
[10] P. J. Denning and J. P. Buzen. 1978. The Operational Analysis of Queueing Network Models. *Comput. Surveys* 10, 3 (1978), 335–261.
[11] Neil Gunther. 2000. *The Practical Performance Analyst*. iUniverse Inc.
[12] A. N. Habermann. 1975. *Introduction to Operating System Design*. SRA.
[13] Mor Harchol-Balter. 2013. *Performance Modeling and Design of Computer Systems: Queuing Theory in Action*. Cambridge University Press.
[14] Darell Huff. 1975. *How to Lie with Statistics*. Penguin.
[15] Raj Jain. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley.
[16] Paul C Jorgensen. 2018. *Software testing: a craftsman's approach*. CRC Press.
[17] Edward D. Lazowska, John Zahorjan, G. Scott Graham, and Kenneth C. Sevcik. 1984. *Quantitative System Performance, Computer System Analysis Using Queuing Network Models*. Prentice Hall.
[18] D. J. Lilja. 2000. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press.
[19] Daniel A. Menasce and Virgilio A. F. Almeida. 2002. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall.
[20] C.U. Smith and L.G. Williams. 2001. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley. https://books.google.com/books?id=X5VlQgAACAAJ
[21] Connie U. Smith. 1986. Independent general principles for constructing responsive software systems. *ACM Transactions on Computer Systems* 4 (1986), 1–31.
[22] Ian Somerville. 2016. *Software Engineering* (tenth ed.). Pearson.
[23] Catia Trubiani, Alexander Bran, André van Hoorn, Alberto Avritzer, and Holger Knoche. 2018. Exploiting load testing and profiling for performance antipattern detection. *Information and Software Technology* 95 (2018), 329–345.
[24] E.R. Tufte. 2001. *The Visual Display of Quantitative Information*. Graphics Press.
[25] Matt Walsh. 2003. The End of Programming. *CACM* 66 (2003), 34–35. https://dl.acm.org/doi/10.1145/3570220
[26] Ye Yang and André B. Bondi (moderators). 2022. Roadmap for Cultivating Performance-Aware Software Engineers. Panel discussion. https://www.cmg.org/2022/02/panel-roadmap-for-cultivating-performance-aware-software-engineers/
[27] Yutong Zhao, Lu Xiao, Andre B Bondi, Bihuan Chen, and Yang Liu. 2022. A Large-Scale Empirical Study of Real-LifePerformance Issues in Open Source Projects. *IEEE Transactions on Software Engineering* (2022).
[28] Yutong Zhao, Lu Xiao, Xiao Wang, Lei Sun, Bihuan Chen, Yang Liu, and Andre B Bondi. 2020. How are performance issues caused and resolved?-an empirical study from a design perspective. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 181–192.