Cultivating Performance Awareness in a Testing Project: A Focus on Machine-Readable Travel Documents

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

Andre B. Bondi
Stevens Institute of Technology
Hoboken, New Jersey, USA
abondi@stevens.edu

Eman Abdullah AlOmar
Stevens Institute of Technology
Hoboken, New Jersey, USA
ealomar@stevens.edu

Yu Tao
Stevens Institute of Technology
Hoboken, New Jersey, USA
ytao@stevens.edu

ABSTRACT

This paper presents a course project to integrate performance engineering concepts into a software testing and quality assurance curriculum. It uses the real-world context of validating and testing Machine-Readable Travel Documents (MRTDs) to integrate multiple testing techniques, including unit testing, mocking, mutation testing, and performance measurement. This integration allows students to "connect the dots" between different testing methodologies, enhancing their ability to apply them holistically in software testing projects. A key goal of the project is to help students understand how performance testing naturally fits into the overall testing process—just as it would in real-world practice—alongside functional testing. Students engage in hands-on exercises that require evaluating both functional correctness (e.g., conformance to MRTD standards) and performance attributes, such as execution time and the cost of encoding and decoding large sets of input records. The preliminary results suggest that this approach not only deepens students' understanding of performance engineering but also encourages them to view testing as a multifaceted process. We share this project with other educators as a framework for incorporating performance testing into software testing curricula, ensuring that students can practice critical testing skills in a real-world context.

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. Software Testing. Performance Testing.

ICPE Companion '25, May 5–9, 2025, Toronto, ON, Canada.

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/XXXXXX.XXXXXX

ACM Reference Format:
Lu Xiao, Andre B. Bondi, Eman Abdullah AlOmar, and Yu Tao. 2025. Cultivating Performance Awareness in a Testing Project: A Focus on Machine-Readable Travel Documents. In *Companion of the 16th ACM/SPEC International Conference on Performance Engineering (ICPE Companion '25), May 5–9, 2025, Toronto, ON, Canada.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION

In the field of software engineering, non-functional requirements such as performance and scalability are essential for the success of software systems [28, 33, 34]. The importance of performance evaluation for computer science was prominently advocated in the 1980s [8]. Despite that, performance engineering remains underemphasized in most undergraduate curricula. Although real-world applications rely heavily on system scalability, resource utilization, and response time, these topics are often only briefly covered in traditional software development courses, leaving many graduates with limited exposure to performance modeling and analysis [3, 29, 30]. This educational gap becomes evident in the workplace, where engineers are expected to design not only functionally correct systems but also ensure they perform efficiently under various constraints [31]. Recent studies show that new graduates frequently struggle with understanding and addressing performance concerns, often having to develop these critical skills, such as performance optimization and bottleneck analysis, through self-learning on the job rather than through formal education [10, 21].

The consequences of neglecting performance considerations are significant, as seen in high-profile failures like the crashes of healthcare.gov when it was rolled out and of COVID-19 vaccination scheduling websites at the height of the recent pandemic. These incidents highlight the risks of overlooking performance during the software development lifecycle [11]. They underscore the urgent need for educational programs that equip students with the tools to proactively address performance issues from the outset [1].

At Stevens Institute of Technology, our software testing course covers a range of testing methodologies, such as unit testing, test-driven development, mocking, and test automation. In addition to functional testing, students are introduced to performance testing and other non-functional aspects, helping them evaluate how systems behave under different constraints [12][4],[5]. However, feedback from past students revealed challenges in connecting various

testing techniques and in incorporating non-functional requirements like performance into overall testing. To address this, we developed the Machine-Readable Travel Document (MRTD) project, which integrates different testing techniques while emphasizing the critical importance of performance testing in real-world scenarios.

The MRTD project was inspired by a real-world incident reported by The Guardian [23], in which a 101-year-old Italian man was mistakenly categorized as a baby by the UK's immigration system due to errors in processing his MRTD data. The problem arose because the standard passport format only stores the last two digits of the bearer's year of birth. This incident sparked classroom discussions on how MRTDs handle sensitive data like birth dates and the broader implications of data formats being constrained by international standards for documents and restricted information storage space. These discussions led to the development of the MRTD project, which emphasizes not only functional correctness, such as validating conformance to international standards, but also non-functional testing aspects like performance. The project requires students to apply a variety of testing techniques, including unit testing, mutation testing, and mocking for unimplemented components such as database interactions and hardware scanners. Additionally, the project incorporates performance testing by asking students to measure execution times for encoding and decoding large sets of MRTD records, providing insights into system scalability and processing efficiency. By combining these functional and non-functional testing elements, the MRTD project highlights the critical importance of both accuracy and performance in systems that process large volumes of standardized data, making it an ideal case study for this paper.

We conducted a preliminary evaluation of this project in the class taught during the 2024 fall semester. Among the 40 undergraduate and graduate students in the session, eight students consented and completed a survey that asked about their confidence about performance. Overall, the survey results highlight the effectiveness and usefulness of the different parts of the project in achieving their respective learning objectives. Students have also reported improved confidence in three key learning objectives of this project: 1) applying unit testing techniques, such as mocking and mutation testing; 2) connecting different testing techniques, from requirements analysis to performance measurement; and 3) specifically, integrating performance measurements into the testing life cycle. However, two students reported reduced confidence in integrating performance analysis, which motivated us to further improve the design of the project. For example, we plan to emphasize performance engineering concepts more explicitly in different parts of the project in future iterations.

2 MRTD PROJECT DESIGN

An MRTD should present the information necessary for automated inspection in any country using visual inspection and machine-readable (optical character recognition) means. Figure 1 is an example composed of two parts: a visual inspection zone (VIZ) and a machine-readable zone (MRZ). The MRZ contains two lines. The first line specifies the Type of document, such as a passport, visa, or identity card, the issuing country, and the name of the holder. The second line specifies the passport number, country code, birth date,

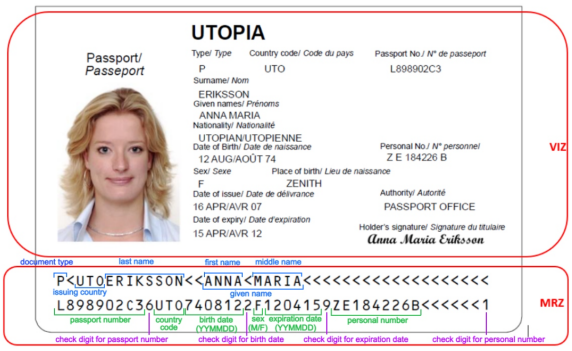


Figure 1: An Example Machine Readable Travel Document. Source: [14].

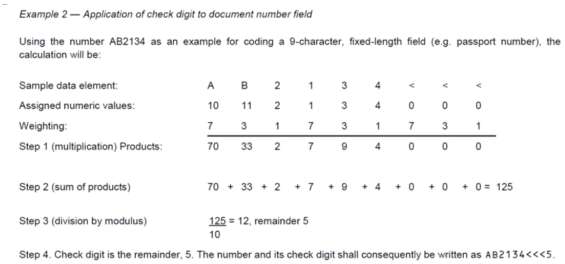


Figure 2: Check Digit Example. Source: [14].

gender, expiration date, and personal number. In addition to these information fields, there are four check digits inserted in between and at the end of the information fields. Figure 2 illustrates the algorithm for calculating the check code. In the example, they are "6", "2", "9", and "1". The check digit is used to check the correctness of the information fields. We provide the detailed illustration of how the check digit is calculated in the assignment based on the MRTD specification [14].

Students are tasked to implement a "system" that "reads" the MRZ of a travel document, verifies its check digits, and reports any mismatches. Assume the system scans the MRZ using a hardware device to obtain two text lines. Students implement an algorithm to decode these strings into their respective fields and identify the check digits. In the reverse direction, students implement an algorithm to encode provided document information, including name, DOB, etc., back into the MRZ format. If any field's data does not match its check digit, the system must indicate where the discrepancy occurred.

The students were guided to accomplish this project through the following parts:

2.1 Part 0: Project Planning

Before the students embark on this project, we ask the team to draw a Gantt chart [20, 27] to depict the tasks and the planned timeline. The plan should show the dependencies between the tasks. Upon completion of the project, students should add another column to

the Gantt chart to show who did the work and when each task was completed.

The project planning phase encourages students to collaborate, enhancing their communication and teamwork skills. We required students to create a Gantt chart. This helps them learn to visualize project timelines and task dependencies. In particular, students were required to consider performance requirements and verification in the planning phase, instead of treating them as an afterthought. This initial planning also fosters a sense of ownership and accountability, as students must define their roles and responsibilities. This experience mirrors industry practices, where clear project management is crucial for successful software delivery [22].

2.2 Part 1: Requirements Analysis

We first ask the students to read the specification documentation [14] of the MRTD and the project requirements. The expectation is for students to identify any ambiguity in the current requirement specifications. Students can obtain additional information from the MRTD document or make assumptions to remove ambiguity. Particularly, we intentionally define the performance requirement vaguely as *"The system shall efficiently process the encoding and decoding of information."* Students are expected to redefine the respective performance requirements with measurable metrics. Additionally, ambiguities may arise in the interpretation of date formats and input validation against international standards, etc.

By reviewing and clarifying project requirements, students practice confronting real-world challenges where requirements are often incomplete or ambiguous [7, 24]. This process also highlights the relevance of performance requirements and conformance to standards, encouraging students to consider broader implications beyond mere functionality. Students practice essential skills in communication and documentation, preparing them for collaborative environments where clear specifications are critical for project success.

2.3 Part 2: Implementation and Unit Testing

Next, students focus on implementing and testing the encoding and decoding algorithms for the MRTD system. In this part, students practice three key techniques in Unit testing, which is a key component of software quality assurance [6, 16, 26]: 1) test-driven design [2], 2) mocking [18], and 3) mutation testing [32], which are elaborated below:

Students first follow the test-driven design approach [2, 9], where they define test cases before they fill in the functions to make the test cases pass. This underscores the concept that testing is integral to the coding process [2, 13, 19]. Meanwhile, an MRTD system naturally relies on a hardware device to scan traveler's information and access a database to retrieve or store related information. This provides the opportunity for students to use mocking to isolate the dependencies on hardware scanning or accessing an SQL database in the context of this project. This task prepares them for real-world scenarios where not all components are readily available. Additionally, we asked students to use MutPy [1] to perform mutation testing to examine how well their test cases are likely to capture bugs in the algorithms. This exercise encourages them to critically evaluate their testing strategies, fostering a mindset of continuous improvement [15, 25].

Of particular note, as part of the test coverage report, we ask students to examine the execution time of their test cases, and reason how the execution time might be impacted if they had used the real physical scanner and had accessed an external database, without using mocking. The execution efficiency of test cases is critical in modern CI/CD environments but is seldom explicitly mentioned. Admittedly, this task mainly focuses on unit testing, which is not directly relevant to the performance engineering of the MRTD project. However, it provides an opportunity for students to understand a key motivation of using mocking—improving test execution efficiency—in a more general context.

2.4 Part 3: Performance Measurement

In this part, students were asked to measure the performance of their encoding and decoding algorithms with large input data sets. To this end, we had the students read two input files. The first input file contains 10,000 encoded fictitious passport records supposedly scanned from MRTDs. Each line of the input file contains both lines of a fictitious record corresponding to the two lines in Figure 1 separated by a semicolon. The students' task is to measure the execution times to process the first 100 records, the first 1000 records, and the first n thousand records for n running from 1 to 10. Likewise, the second input file contains 10,000 decoded records. Each block contains the issuing country, last name, given name, passport number, country code, birth date, sex, expiration date, and personal number that are needed to generate the records to appear on the MRTD. The students will use Python timing libraries [2] to measure how long the program takes to run. Finally, students were asked to use Excel (or some other tool) to plot the data to show how the execution time changes with the increase of input size, and also write one or two paragraphs explaining their results.

By providing students with large test files and requiring them to measure execution times, this phase emphasizes the importance of performance metrics and measurements in software applications. It encourages students to analyze the efficiency of their implementations and make data-driven decisions regarding optimizations. The practical experience gained in plotting execution times cultivates an understanding of the impact of input size on performance, bridging theoretical knowledge with practical application.

2.5 Part 4: Test Planning in (Hypothetical) Practice

In real-world software projects, not all system details or requirements are fully known at the start. The MRTD application is no exception: it may have evolving user needs and unforeseen constraints. Consequently, students are asked to write a more comprehensive test plan that reflects these uncertainties. Students were asked to analyze several core assumptions and decisions, including: Which features are most critical? What risks (technical, operational, performance-related) might jeopardize project success? How generous or limited is the testing budget? How does this affect the overall project scope, particularly in terms of both functionality

[1] https://pypi.org/project/MutPy/

[2] https://realpython.com/python-timer/

and testing depth? What does "success" mean from both a verification (meeting specified requirements) and validation (meeting end-user needs) perspective? How do we account for performance as part of success? These assumptions provide a foundation for the test plan. Note that while budgets and timelines may constrain the project, testing itself should remain a priority. If a budget is limited, teams might descope certain features, but the testing of the remaining critical functionality—including performance—must not be compromised.

This exercise mirrors industry practices where thorough test planning is key to software quality. Students are exposed to the depth of strategic decision-making required for professional software development. Incorporating performance engineering ensures they also learn how to anticipate and address scalability or speed bottlenecks proactively instead of treating it as an after-thought.

3 PRELIMINARY EVALUATION

3.1 Evaluation Design

This study was approved by the Stevens Institutional Review Board (IRB). We used an anonymous survey to gather feedback on the students' experiences, challenges, and learning outcomes from this project. All participants provided their informed consent prior to participating in the study. The survey can be found here [3]. As suggested by Kitchenham and Pfleeger [17], we use a 5-point ordered response scale ('Likert scale') questions, open-ended questions, and multiple choice questions. The survey is composed of five logical parts:

The first part asks about participants' educational background, such as program of study and years of experience with programming.

The second part asks about the effectiveness of different parts of the project in achieving their respective learning outcomes. For instance, for Part 0, we ask *"How effective was creating and maintaining a Gantt chart in helping you organize and execute the project?"* We also ask students to provide open-ended input by asking *"What did you learn about project management and teamwork from this part?"* As another example, for Part 3, we ask *"How much do you think this project helps you integrate performance measurement into the general testing practice?"* and also ask an open-ended question, *"What did you learn about the relationship between performance metrics and software quality?"* Furthermore, we also ask a multiple-choice question, where students select the part(s) that they think are most helpful.

The third part asks students to self-evaluate their confidence in applying three key aspects aimed by the project: 1) applying software testing techniques, including unit testing, mocking, and mutation testing; 2) the integration of performance analysis into the software development lifecycle; and 3) connecting different testing techniques, including performance testing, in a cohesive testing plan. The students were asked to evaluate these three aspects on a scale of 1 to 5 based on their confidence as "before" and "after" this project.

Finally, we ask students open-ended questions about their overall comments, suggestions, and challenges of the project.

[3] https://github.com/lxiao6/SSW567-MRTD/

3.2 Results

We distributed the survey to a total of 40 students (30 undergraduate and 10 graduate students) taking this course in the 2024 Fall semester. We collected a total of 13 responses, out of which 5 were incomplete and thus excluded from the results. Therefore, our results are derived from 8 complete survey results with participants' consent.

Participant Background. All participants are above 18 years old and are majoring in Software Engineering. They have from 1 to 5 years of programming experience, with an average of 2.6 years and a standard deviation of 1.3.

Effectiveness of Different Parts. Figure 3 shows the rating of effectiveness we received on each part. As we can see, for all five parts of the project, the majority of participants rated their effectiveness/usefulness for their learning of their respective objectives 4 or 5 on a scale of 1 to 5 (5 means most effective and 1 least effective). More specifically, the average and standard deviations on the five parts are: Part 0 $\langle 4.25, 0.97 \rangle$; Part 1 $\langle 4.13, 0.78 \rangle$; Part 2 $\langle 4.25, 0.83 \rangle$; Part 3 $\langle 3.75, 1.2 \rangle$; and Part 4 $\langle 4.13, 0.78 \rangle$. The ratings overall suggest the effectiveness and usefulness of each part of the project.

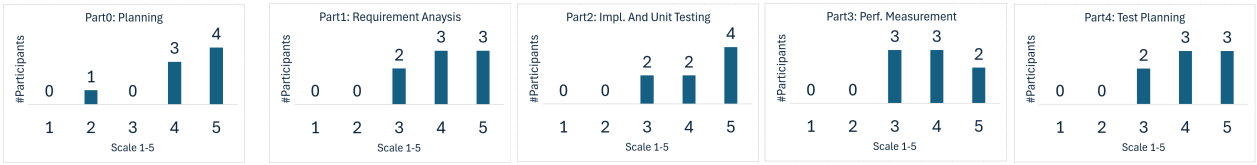
When asked to select the most useful parts of the project, Part 0, Part 2, and Part 3 received 5 votes each, Part 1 received 4 votes, and Part 4 received 2 votes.

Key Learning Outcomes. Figure 4 shows the comparison of students' self-reported confidence in the three key learning objectives in 1) applying the unit testing techniques shown in Figure 4a; 2) connecting different testing techniques from the parts of the project shown in Figure 4b; and 3) integrating performance analysis shown in Figure 4c. For the first two objectives, 5 and 6 (majority) of the 8 students reported improved confidence by up to 3 levels (from a scale of 2 to a scale of 5). For objective 3: the confidence in integrating performance analysis, 4 (50%) students reported increased confidence. However, 2 students reported decreased confidence by 1 level, and 2 students reported no change.

Open Comments and Challenges. We also received rich open-ended input regarding how and why students think the parts are useful to them. In Part 0, students commented *"From this part of the project, I learned the importance of clear planning and task prioritization in project management. Breaking the project into manageable tasks and defining deadlines helped keep the project on track...I learned that testing plays a vital role in ensuring the quality and reliability of the final product."* and also *"Advanced planning of each phase of the project, from requirements through every kind of testing, allowed for progress to be more consistently made without conflict, especially asynchronous progress."*

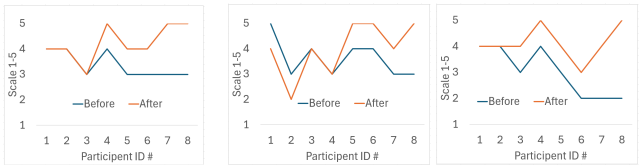
For Part 1, we received comments like *"I gained the insight that clear and detailed requirements are crucial for the success of any software project..."* and *"I realized that clear requirements are needed in order to make an effective project because without them, it is hard to develop proper functionality."*

For Part 2, students commented on challenges faced when first trying to practice mocking and mutation testing *"Using mocks and mutation testing was hard to use at first as I have never done it before. However, after reading documentation and constant testing, I was able*



(a) Part 0 **(b) Part 1** **(c) Part 2** **(d) Part 3** **(e) Part 4**

Figure 3: Rating on Different Parts of MRTD Project



(a) Unit Testing Techniques **(b) Connecting Testing Techniques** **(c) Integrating Performance Analysis**

Figure 4: Key Learning Objectives

to figure it out." and *"For mutation testing, it was tricky to identify how mutations impacted functionality. I addressed this by analyzing mutants, improving test coverage, and ensuring that edge cases were included.".*

For Part 3, students commented *"I learned that performance metrics are crucial indicators of software quality. They help assess how efficiently the system handles large datasets and performs under stress, directly affecting user experience. Even if a system functions correctly, poor performance can lead to delays or failure in real-world applications. Monitoring metrics like execution time and resource usage ensures that the software is not only correct but also scalable and responsive."*

For Part 4, students commented *"The most insightful aspects of test planning were risk identification and budget considerations. Identifying potential risks, such as performance bottlenecks or misalignment with standards, helped prioritize testing efforts and ensure that critical issues were addressed first. Budget considerations emphasized the importance of using open-source tools to minimize costs, while still achieving comprehensive testing coverage and performance validation. Balancing resources and risks helped create a focused, cost-effective testing strategy."*

As for the most important skills learned from the project, students commented *"The most valuable skill I gained from this project was the ability to integrate comprehensive testing strategies into the software development process. This included writing unit tests, performing mutation testing, and analyzing performance metrics. It taught me how to ensure both the correctness and efficiency of a system, while also emphasizing the importance of planning, risk management, and resource allocation in delivering high-quality software."*; *"The Gantt chart and the organization to it. I also learned how to test code efficiently and correctly."* and *"The most valuable skill I gained from this project was being able to test based on certain requirements and tailor the project based on these requirements."*

Students also made great suggestions for improvements: *"To enhance learning and clarity, I suggest incorporating real-world components, such as actual hardware interaction or a simple database for encoding/decoding, to provide a more practical experience. Additionally, clearer documentation of the MRZ format and encoding rules would ensure better understanding. Expanding testing scenarios to include more complex edge cases would help deepen knowledge of error handling. Using collaborative tools like Jira or Trello could improve task management and teamwork, while time-boxed performance tests under resource constraints would offer a more realistic view of system performance in real-world conditions."*

4 LESSONS LEARNED

This paper aims to provide a framework for educators to incorporate both functional and non-functional testing into their curricula using the Machine-Readable Travel Document (MRTD) project as a foundation. While the current scope of the project equips students with essential skills in testing and performance evaluation, there are numerous ways in which the depth and breadth of the project can be expanded to further enhance its educational impact.

One potential direction is to extend the project by introducing more complex, real-world scenarios. For instance, students could be tasked with managing various versions of the MRTD standard or handling documents from different countries with unique requirements. This would simulate the complexities of global systems, teaching students to navigate variations in standardized protocols.

In addition, the framework could be expanded to cover other non-functional requirements. For example, introducing security concerns—such as encryption of sensitive personal data or validation of data integrity—could provide a practical lesson in developing secure systems. Likewise, adding maintainability considerations, such as code complexity or adherence to software design principles, would encourage students to think beyond short-term solutions and consider the long-term sustainability of their code.

Concerning data presentation, our original plan was to evaluate students' understanding and confidence about software performance with questionnaires, as described in the previous section. We also had the opportunity to qualitatively evaluate students' understanding of their experimental results in the MRTD project and in a project involving load simulation in another course by examining their reports in detail. We found that the quality of their reports showed highly variable abilities to plan, analyze, and present quantitative results of their experiments in an organized and coherent manner. An assignment of this nature in open-ended terms allows us to assess students' abilities to deal with quantitative data but might deprive the instructors of the opportunity to

guide students on how this work should be done and presented for future maximum impact in the workplace. Based on this experience, we believe that it would be worthwhile to describe how performance data should be obtained and presented in detail, so that students are trained in the production of succinct, coherent reports on their experimental work. Therefore, instructors should be encouraged to devote about an hour of class time to data presentation and report preparation. Written guidelines could be given on how experimental assignments should be done.

5 CONCLUSION

The Machine-Readable Travel Document (MRTD) project offers a framework for integrating both functional and non-functional testing into software engineering education. Beginning with project planning and test strategy development, students engage in project management practices that reflect real-world testing environments. The core functional components include check digit validation and unit testing, where students apply a test-driven development approach to ensure conformance to international standards. Mutation testing further enhances the robustness of their test cases, while mocking allows them to isolate system components without implementing external dependencies. The project also emphasizes performance testing, challenging students to measure system efficiency while understanding the impact of testing overhead. By combining these diverse testing techniques, the MRTD project provides students with a hands-on learning experience that mirrors the complexities of real-world software systems and offers a flexible foundation for future expansions. Additionally, we plan to expand the sample size to strengthen the results.

ACKNOWLEDGEMENT

This work was supported in part by the U.S. National Science Foundation (NSF) under grants CCF-2044888 and DUE-2142531.

REFERENCES

[1] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. 2004. Model-based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering* 30, 5 (2004), 295–310.
[2] Kent Beck. 2022. *Test driven development: By example.* Addison-Wesley Professional.
[3] A. B. Bondi. 2014. *Foundations of Software and System Performance Engineering: Process, Performance Modeling, Requirements, Testing, Scalability, and Practice.* Addison-Wesley.
[4] Andre B Bondi and Razieh Saremi. 2021. Experience with Teaching Performance Measurement and Testing in a Course on Functional Testing. In *Companion of the ACM/SPEC International Conference on Performance Engineering*. 115–120.
[5] 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*. 345–349.
[6] Ermira Daka and Gordon Fraser. 2014. A survey on unit testing practices and problems. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 201–211.
[7] Senay Tuna Demirel and Resul Das. 2018. Software requirement analysis: Research challenges and technical approaches. In *2018 6th International Symposium on Digital Forensic and Security (ISDFS)*. IEEE, 1–6.
[8] Peter J Denning. 1981. ACM president's letter: performance analysis: experimental computer science as its best. *Commun. ACM* 24, 11 (1981), 725–727.
[9] Torgeir Dingsøyr, Sridhar Nerur, VenuGopal Balijepally, and Nils Brede Moe. 2012. A decade of agile methodologies: Towards explaining agile software development. , 1213–1221 pages.
[10] Robert F. Dugan. 2004. Performance Lies My Professor Told Me: The Case for Teaching Software Performance Engineering to Undergraduates. In *ACM SIGSOFT Software Engineering Notes*. 37–48.
[11] Paul Ford. 2013. The Obamacare Website Didn't Have to Fail: How to Do Better Next Time. *Bloomberg Businessweek* (2013).
[12] Vahid Garousi, Austen Rainer, Per Lauvås Jr, and Andrea Arcuri. 2020. Software-testing education: A systematic literature mapping. *Journal of Systems and Software* 165 (2020), 110570.
[13] Boby George and Laurie Williams. 2004. A structured experiment of test-driven development. *Information and software Technology* 46, 5 (2004), 337–342.
[14] 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.
[15] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering* 37, 5 (2010), 649–678.
[16] Vladimir Khorikov. 2020. *Unit Testing Principles, Practices, and Patterns.* Simon and Schuster.
[17] Barbara A Kitchenham and Shari L Pfleeger. 2008. Personal opinion surveys. In *Guide to advanced empirical software engineering*. Springer, 63–92.
[18] Tim Mackinnon, Steve Freeman, and Philip Craig. 2000. Endo-testing: unit testing with mock objects. *Extreme programming examined* (2000), 287–301.
[19] E Michael Maximilien and Laurie Williams. 2003. Assessing test-driven development at IBM. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 564–569.
[20] Harvey Maylor. 2001. Beyond the Gantt chart:: Project management moving on. *European management journal* 19, 1 (2001), 92–100.
[21] Daniel A. Menasce. 2002. Software, Performance, or Engineering?. In *Proceedings of the 3rd International Workshop on Software and Performance*. 239–242.
[22] Farzana Asad Mir and Ashly H Pinnington. 2014. Exploring the value of project management: linking project management performance and project success. *International journal of project management* 32, 2 (2014), 202–217.
[23] Lisa O'Carroll and Angela Giuffrida. 2020. Home Office tells man, 101, his parents must confirm ID. https://www.theguardian.com/uk-news/2020/feb/12/home-office-tells-man-101-his-parents-must-confirm-id. [Online; accessed 07-October-2024].
[24] Dhirendra Pandey, Ugrasen Suman, and A Kumar Ramani. 2010. An effective requirement engineering process model for software development and requirements management. In *2010 International Conference on Advances in Recent Technologies in Communication and Computing*. IEEE, 287–291.
[25] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. 2019. Mutation testing advances: an analysis and survey. In *Advances in computers*. Vol. 112. Elsevier, 275–378.
[26] Per Runeson. 2006. A survey of unit testing practices. *IEEE software* 23, 4 (2006), 22–29.
[27] Tom Seymour and Sara Hussein. 2014. The history of project management. *International Journal of Management & Information Systems (Online)* 18, 4 (2014), 233–240.
[28] Connie U Smith and Lloyd G Williams. 2002. *Performance solutions: a practical guide to creating responsive, scalable software.* Vol. 1. Addison-Wesley Reading.
[29] Connie U. Smith and Lloyd G. Williams. 2002. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison-Wesley.
[30] Connie U Smith and Lloyd G Williams. 2003. Best practices for software performance engineering. In *Int. CMG Conference*. 83–92.
[31] Lloyd G. Williams and Connie U. Smith. 2015. Software Performance Antipatterns. *Workshop on Software and Performance* (2015), 127–136.
[32] Martin R Woodward. 1993. Mutation testing—its origin and evolution. *Information and Software Technology* 35, 3 (1993), 163–169.
[33] Shahed Zaman, Bram Adams, and Ahmed E Hassan. 2011. Security versus performance bugs: a case study on firefox. In *Proceedings of the 8th working conference on mining software repositories*. ACM, 93–102.
[34] Shahed Zaman, Bram Adams, and Ahmed E Hassan. 2012. A qualitative study on performance bugs. In *2012 9th IEEE working conference on mining software repositories (MSR)*. IEEE, 199–208.