



When Research Software Goes to Class: Lessons From Embedding Research Software Into Teaching

ISSUES IN RESEARCH
SOFTWARE

MICHAEL DORNER

ANDREAS BAUER

FLORIAN ANGERMEIR

**Author affiliations can be found in the back matter of this article*

u[ubiquity press

ABSTRACT

Background: Software is at the core of most scientific discoveries today, and the reliability of research results increasingly depends on the quality of the software that underpins them. However, research software is often developed under constraints that prioritize scientific progress over engineering rigor, leaving little to no incentive for maintenance, documentation, or quality assurance.

Objective: This study examines whether embedding an existing research software into a software testing course can contribute to improving the quality of the research software and identifies the associated challenges.

Method: In an *in vivo* experiment, we embedded a large-scale network simulation into a university course on software testing at Blekinge Institute of Technology, Sweden, as a group project and observed the effects on the research software.

Results: We found that the research software benefited from the embedding through substantially improved documentation and fewer hardware and software dependencies. However, the embedding required significant additional effort from us, and although the student teams produced thoughtful and well-designed test suites, none of their code contributions could be merged into the research software due to uncertainties around intellectual property.

Conclusion: We strongly believe that embedding research software engineering activities into teaching can enhance the quality of research software while providing students with exposure to research. However, the uncertainty about the intellectual property of students' code contributions substantially limits its potential.

CORRESPONDING AUTHOR:

Michael Dorner

Technische Hochschule
Nürnberg Georg Simon Ohm,
Germany; Formerly with
Blekinge Institute of
Technology, Sweden

mail@michaeldorfner.de

KEYWORDS:

research software engineering;
testing; teaching; education;
open source; license;
intellectual property;
experiment

TO CITE THIS ARTICLE:

Dorner M, Bauer A,
Angermeir F. 2026 When
Research Software Goes to
Class: Lessons From
Embedding Research Software
Into Teaching. *Journal of Open
Research Software*, 14: 19.
DOI: [https://doi.org/10.5334/
jors.548](https://doi.org/10.5334/jors.548)

1 INTRODUCTION

Software underpins almost all research and is, in one way or another, at the core of most scientific discoveries today. Researchers across different disciplines develop or adapt software to collect, analyze, and visualize data to push the limits of the unknown. Thus, the reliability of scientific results highly depends on the quality of the research software that supports them. However, ensuring high software quality remains a persistent challenge for research software: While software testing for quality assurance in industrial software engineering is widely acknowledged [12, 13, 22], efforts to ensure the quality of research software are often not rewarded in academia [10] and are rarely included in researchers' training [20].

In this research, we investigate whether embedding research software into a regular software engineering teaching context can serve as a practical approach for assuring and improving the quality of the software. Our focus is exclusively on the effects on the research software and its development process, not on student learning or pedagogical outcomes.

In detail, we set out to answer the following research question: What effects does embedding research software into a software engineering teaching context have on the quality of the embedded research software?

To answer our research question, we conducted an *in vivo* experiment in which we embedded research software, which simulates large-scale communication networks, into a university course on software testing at Blekinge Institute of Technology, Sweden, where student groups developed automated test suites for the research software. We collected and qualitatively analyzed observations on how this embedding affected the research software.

The remainder of this article is structured as follows: In Section 2, we briefly discuss the background and related work of our study. We then describe our experimental design with its core components in Section 3 in detail. In Section 4, we report the resulting effects of our experiment on the research software. After discussing the limitations of our study in Section 5, we conclude our paper by raising a fundamental question in the context of embedding research software engineering into teaching: How to handle the intellectual property of students properly?

2 BACKGROUND

In light of our experiment on embedding research software into teaching, we would like to highlight the current approaches and initiatives to foster research software engineering in teaching (Section 2.1) and using open-source software in software-testing education (Section 2.2).

2.1 RESEARCH SOFTWARE ENGINEERING IN TEACHING

As the research software engineering publication monitor shows, research software engineering has gained more attention over the recent years [11]. A proper education has become one of the key concerns in research software engineering [5]. This need for education is addressed by different organizations such as *HIFIS*,¹ *INTERSECT*,² *CodeRefinery*,³ or *BSSw*,⁴ which provide learning materials, offer workshops or teaching, and maintain communities to share research software engineering teaching experiences.

Despite its importance for modern science, surprisingly, research software engineering is rarely a part of curricula. General courses on computer science education (e.g., *The Missing Semester of Your CS Education*⁵ at MIT or the *The Missing CS Class* at University of California, Davis, [15]) or courses specific to research software engineering (e.g., University of Potsdam [2], Technical University of Dresden [17]) are notable and well-received exceptions.

Even though there is progress towards addressing education, we agree with Goth et al. [16] that a systemic institutionalization of research software engineering education is needed.

2.2 OPEN-SOURCE SOFTWARE FOR TEACHING SOFTWARE TESTING

Research on software-testing education has gained more attention over the last years [14]. However, providing authentic, real-world software quality assurance and testing experiences within the context of a Computer Science or Software Engineering curriculum is challenging [4, 6]. Different studies used open-source software projects for this reason [4, 6, 18, 24]. Open-source software is software that is available under a license that grants the right to use, modify, and distribute the software—modified or not—to everyone free of charge (adapted from [21]).

None of the studies have addressed the ethical and potentially legal issue that we will discuss in this article: Contributing to open-source projects requires the copyright of students, which puts us researchers and teachers in a conflicting position since we (teachers) grade students for work that we (researchers) exploit.

Venson and Alfayez [24] proposed and discussed their teaching approach for a software testing course that integrates theory and practical experience through the utilization of both team-based learning and active contributions to open-source software projects. Although not in the context of research software testing, we can partially confirm their reflections and recommendations in our experiment: Almost all students had no experience with contributing to open-source software. Therefore, the quality of documentation plays a pivotal role in the integration of open-source software into testing courses.

In contrast to our experiment, Venson and Alfayez [24] awarded students for code contributions via pull requests with extra points. Pull requests are a convenient way for software developers to collaborate on open-source projects by proposing changes.⁶ However, they observed that ‘this incentive prompted students to submit pull requests lacking genuine contributions, which inundated the [open-source software] community with superfluous requests.’ As a result, they discontinued offering extra points for pull requests. In our work, we did not incentivize contributions, for example, by offering bonus points or other rewards; we did not observe any code contribution in the form of a pull request.

3 EXPERIMENTAL DESIGN

In this section, we describe the design of our *in vivo* experiment, which examines the effects of embedding research software into a teaching context.

An *in vivo* experiment (or field experiment [23]) is conducted in a natural setting with a high degree of realism. The researcher deliberately introduces a change (or *intervention*) into this existing setting and observes its effects [1]. Unlike laboratory experiments, where the setting is constructed or controlled for the purpose of experimentation, the natural setting in an *in vivo* experiment exists independently of the researcher [23].

In our case, the natural setting is a course on software testing that forms part of the regular curriculum for the Software Engineering and Artificial Intelligence programs at Blekinge Institute of Technology, Sweden. The deliberate change we introduced was to embed research software into this existing course structure. The course includes a project work in which student groups develop automated test suites. Instead of using a typical software project or an open-source component, as it is often used in teaching [4, 6, 18, 24], we embedded an actual research software into this existing course

structure and observed the resulting effects on that research software. Using such real research software increases the relevance of our observations, as toy systems often fail to represent the complexity of real software, which can limit the external validity of empirical findings [25].

Figure 1 provides an overview of the experimental setting.

In the following sections, we describe the two core components of our experimental design: the natural setting of the experiment and the research software we embedded into that setting.

3.1 SETTING

The setting for our experiment is a software testing course for students of two study programs at Blekinge Institute of Technology: Bachelor of Science (BSc) in Software Engineering and Master of Science (MSc) in Artificial Intelligence. The course was mandatory for students in the Bachelor’s program in Software Engineering but optional for those enrolled in the Master’s program in Artificial Intelligence.

This software testing course introduces hands-on testing and quality assurance techniques for software systems. The course aims to make its participants realize how testing can improve software quality if it is effectively integrated into the software development processes, and understand how this can be achieved using both established and new techniques in software testing. The aim is also to convey practical experience with tools that support and automate these techniques.

As such, the course is split into two parts. In the first part, 10 lectures provide the theoretical foundations of software testing. Two practical assignments deepen the understanding of the concepts introduced in the lectures and introduce students to the technical aspects of software testing in Python. In the second part of the course, students work in groups and develop a test suite for a software project. In previous course iterations, the software projects ranged from open-source to projects

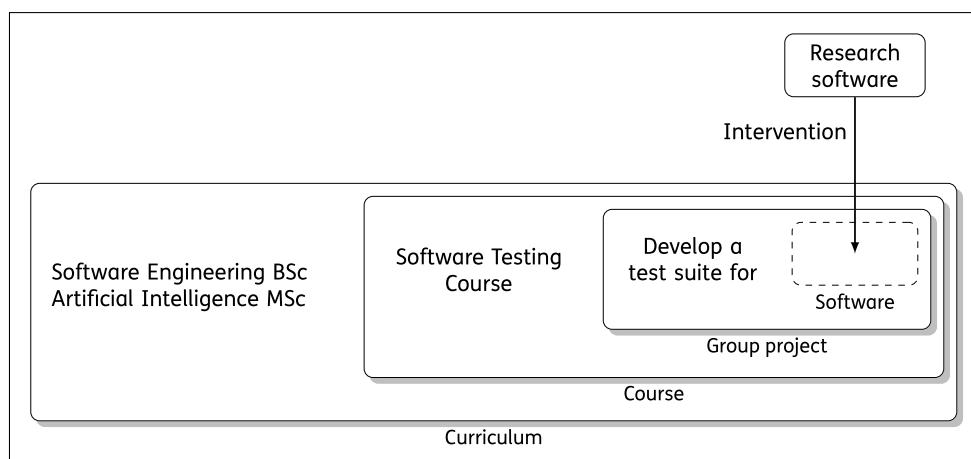


Figure 1 Overview of the research setting of our experiment and its intervention.

that students developed. Afterward, the students summarize their findings as a written report.

We conducted the experiment in the Spring of 2023 with 40 enrolled students. Twelve teams of three to four students worked on the given tasks.

3.2 RESEARCH SOFTWARE

As part of the experiment, we asked the students to develop a test suite for research software that we developed previously to simulate information diffusion in code reviews at Microsoft, Spotify, and Trivago modeled as communication networks [7].⁷

The research software models the communication networks emerging from code review as time-varying hypergraphs in which each vertex represents a developer and each edge represents a code review, which can connect multiple developers exchanging information. Hypergraphs are a generalization of traditional graphs and capture time-dependent and higher-order interactions in social and communication networks in which edges may connect more than two vertices.

The time dependency also extends the notion of a minimal distance in time-varying hypergraphs: a path cannot only be the shortest in terms of its topological distance but also the fastest or foremost path in terms of the temporal distance. Dijkstra's algorithm can be used to find the minimal paths and, therefore, minimal distances in such graphs. However, Dijkstra's algorithm for time-varying hypergraphs has not been described in the literature or implemented before.

The communication networks modeled as time-varying hypergraphs in our simulation can become huge. For example, the communication network emerging from Microsoft's code review consists of over 309,740 code reviews (edges) and 37,103 developers (vertices). The simulation is, therefore, computationally expensive: On conventional hardware, it takes several days to simulate the information diffusion and to process the results.

Before we started the experiment, the simulation had a minimal test suite covering tests for the graph data structure but not for the algorithm. Furthermore, a minimal but complete `README` file contained instructions on the installation and how to use it. As the course started, the research software was already publicly available, but the related publication [8] was not published.

This simulation is a good fit for our experiment for the following reasons:

- The minimal existing test suite helped the students to get started without revealing too much of the required solution. Yet, there is no single solution to the problems, but different solution paths.
- The algorithm is novel. The lack of a reference implementation makes thorough testing non-trivial and yet very important.

- Manual testing does not scale with the sheer size and complexity of the communication networks. Also, brute-force testing approaches are not feasible due to the computational complexity.

We defined requirements for the students to implement the test suite for the research project as follows:

- The test suite must comprise unit tests to automatically test the research software.
- Test coverage metrics must be reported and discussed.
- The test suite must cover two Python versions (3.8 and 3.11).
- Beyond these minimal requirements, each team must select a focus of their overall test suite. Possible areas to focus on are performance, unit, integration, testing, or visual testing.

Before the group-project phase of the course started, we informed the students about the experiment and provided those instructions along with the `README` file and the scientific literature that introduced the mathematical background [8, 9] to the student teams. After the first week, the student teams decided on one focus area of the test suite.

4 RESULTS

We observed three effects on our research software as a result of embedding into teaching. Although these effects, such as improved documentation and reduced hardware and software dependencies, might have occurred over time, they were specifically triggered by our deliberate embedding. Therefore, they are direct outcomes of this embedding and are detailed in the following sections.

4.1 TESTS BUT NO TEST CODE INTEGRATION

The student teams tested the research software extensively and developed diverse and well-engineered testing suites for our research software as part of their projects. The implemented unit and integration tests validate the expected behavior of the research software. Two project groups extended the suite beyond classical testing to catch memory corruption and safety bugs through fuzz testing. Although no bugs were found, the different testing approaches and diverse implementations of test suits provide a comprehensive perspective on the quality of the research software. As authors of the research software, we would not have been able to provide such in-depth testing for the research software.

However, we could not integrate any test code developed by students into the main project. This

shortcoming is rooted in the way contributions to open-source projects are handled. Our research software is open-source and licensed under the so-called MIT license,⁸ aligning with best practices for research software development [19]. Contributing to any open-source project requires the contributor, as the owner of the intellectual property, to grant permission, per the project's open-source license, to use, copy, distribute, or modify their contributions free of charge. That implies that researchers are only allowed to use student contributions if the students make their code available under the same or a compatible open-source license. In our academic setting, an inherent conflict arises: Teachers evaluate the students' contributions while also benefiting from their contributions. Thus, the teachers (as research software authors) exploit those contributions, although not necessarily for financial gain. Hence, an ethical conflict arises since students may feel forced to grant permission for their intellectual property to avoid any potential negative impact on their grading. In jurisdictions like Germany or Sweden, this also becomes a legal issue. To avoid ethical and legal issues, we reached out for legal advice and to the student council, seeking support. All parties advised us to underline the optional and voluntary character of the potential contributions to the research software. If the research software is intended to be commercialized or if a commercial product is built on top of it, the ethical and legal issues increase in severity.

Another obstacle for group projects is that all group members must agree to publish their code under an open-source license. It would be nearly impossible to unravel contributions from individual students without at least touching the intellectual property of other team members. We did not explore how willing individual group members would have been to contribute to the main project while avoiding potential threats to the voluntary nature of contributions.

Additionally, integrating code into the main software project is not effortless. Very rarely, code is flawless and can be merged into the main software project without further considerations, changes, or discussions. At this point, we would like to emphasize that we feel sympathetic towards the students: Spending additional and yet significant efforts that are not rewarded but might be acknowledged in an academic context as they would become co-authors of research software is not very compelling.

In summary, our research software was tested thoroughly in this course, although we could not motivate the student groups to integrate their code into the main research software.

4.2 REDUCED HARDWARE AND SOFTWARE DEPENDENCIES

Because we relied on students' own computers rather than a unified hardware setup, we minimized the

hardware and software dependencies of the research software to simplify the installation and to accommodate the wide range of hardware, operating systems, and software environments.

Before the course, the research software was tested on macOS and Linux (Ubuntu). In preparation for the course, we tested the research software on the common operating systems Windows, macOS, and Linux (Ubuntu) to avoid complications with students' operating systems and development environments.

We also removed one software component and made it optional. Although the component `orjson`⁹ improves the performance of the file processing substantially, `json`¹⁰ from the Python standard library serves the same purpose and requires no additional steps to use.

Being agnostic on hardware and operating systems and having fewer external dependencies contribute to easier use, maintenance, and extension of our research software, facilitating a broader adoption among the research community. Further, minimizing software dependencies can decrease the fault proneness of a software system [3].

For us, as developers of research software, minimizing hardware and software dependencies required only a modest additional effort, yet it meaningfully improved the research software's portability, maintainability, usability, and overall software quality.

4.3 IMPROVED DOCUMENTATION

We found that scientific literature [8, 9] and a minimal documentation in the form of a `README` file were not sufficient for students. To lower the barriers for students at the beginning and during the course, we continuously improved the documentation in size and quality. We observed that continuously improving documentation enabled efficient communication with the students.

Although the documentation improved, less experienced students requested additional support to understand the domain and technical details of research software, which is crucial for an efficient test suite. Therefore, we offered each group weekly and individual students one-on-one sessions on request. In particular, during the first weeks, this offer was well received, and all groups requested such sessions.

The continuous support through additional weekly meetings for all groups and the continuous improvement of the documentation required significant additional efforts, which exceeded the efforts that would have been required for an open-source project, for example, where the domain and functionality were more obvious.

We believe the improved documentation will help researchers reproduce or replicate our simulation, ultimately fostering scientific growth through traceability, transparency, and usability. This finding confirms

the reflection and recommendation by Venson and Alfayez [24].

5 LIMITATIONS

A key limitation of this study is that it was not planned as a research project from the outset. The first author was assigned to teach the course at short notice, which limited the opportunity to design the experiment in advance, set up structured student feedback instruments, or obtain ethical approval for collecting and analyzing student learning data. Consequently, student perspectives and learning outcomes are not included in this study. We plan to replicate our experiment for the next iterations of the course and then collect student feedback to better understand the challenges and benefits for students as future researchers and future research software engineers.

As for any *in vivo* experiment, there are two inherent limitations [23]: First, our findings are limited in their transferability to other settings, which means other courses and also other research software. Second, although the setting of the experiment is realistic, it is subject to confounding factors such as the specific type of research software or previously attended courses that limit the precision of measurement. However, we believe that our high-level findings are still transferrable to other settings since the course introduces the students to the foundations of software testing and does not require prior knowledge of software testing. Additionally, two different curricula participated: Bachelor of Science (BSc) in Software Engineering and Master of Science (MSc) in Artificial Intelligence, which broadens the transferability to other curricula. Last, the research software uses Python, including Jupyter Notebooks, which are common in research software and teaching.

A considerable limitation of this study is that we do not quantify the (additional) effort for the integration since we did not track the hours spent on the integration or previous courses. Our estimates are based on personal experiences and relative to previous instances of the courses. However, we believe that a relative effort, as reported in this study, also provides valuable insights for other research software developers and teachers, even though a more precise effect size is missing.

Since our research software, like a large subset of research software in general, is licensed under an open-source license and is, therefore, open-source software, we assume an overlap in embedding open-source software into teaching environments. However, we focus exclusively on research software and its ethical implications without delving into this overlap since the software not only serves an educational purpose but also supports the broader goal of enhancing the quality and impact of research software.

6 CONCLUSION

In our experiment, we embedded a real-world research software into the teaching context and observed the resulting effects on it.

On the one hand, embedding research software into teaching significantly enhanced the testability and maintainability of the software, thereby increasing confidence in its quality. Several student teams developed outstanding and thoughtful test suites, providing a depth and breadth of testing that would not have been feasible for the research team alone. The embedding also required us to improve the documentation to support student onboarding and to reduce hardware and software dependencies, which simplified the setup and further strengthened its maintainability, preparing it for future scientific reuse and extension. Therefore, we believe that embedding research software into teaching creates a versatile and powerful incubator for research software.

On the other hand, those positive effects did not come for free. Due to the complexity of the topic, we had to invest a significant amount of time in preparing the research software for the course and in providing support to student teams.

We believe that the most significant challenge, however, is the uncertainty about the intellectual property of student code contributions to the research software. Contributing to open-source software projects requires contributors to license their copyrights under the project's open-source license, which puts us researchers and teachers in a conflicting position: We (teachers) grade students for work that we (researchers) exploit, although maybe not financially. This conflict raises at least ethical, but also—in jurisdictions like Germany or Sweden—legal questions. We could not incentivize or motivate students to go the extra mile to integrate their excellent work into the research software. It remains unclear how this dilemma could be resolved. This uncertainty regarding students' contributions extends beyond research software and is observable in contributions through bachelor's and master's theses.

To build on the findings of this study, future research is required to focus on two key directions: First, exploring the learning experiences of students is crucial, as this important aspect was not covered in our current work. This exploration can provide valuable insights into the educational impact of embedding research software and its engineering into teaching. Second, interdisciplinary research is needed to examine the legal and ethical perspectives related to the issues we raised. Addressing these perspectives can help remove the existing uncertainty and guide more informed practices in this area.

Although there are not yet well-worn paths for embedding research software into teaching, we believe

that research software engineering and teaching can cross-fertilize. On the one hand, the research software gets an incubator before being reused or extended in the scientific community. On the other hand, students get in touch with the latest research in their respective fields very early on as part of their curriculum. This hands-on experience with research software engineering can help to train and prepare students, the next generation of researchers.

NOTES

- 1 <https://hifis.net/services/software/training.html>.
- 2 <https://intersect-training.org/>.
- 3 <https://coderefinery.org/>.
- 4 <https://bssw.io/>.
- 5 <https://missing.csail.mit.edu/>.
- 6 <https://docs.github.com/en/pull-requests>.
- 7 <https://github.com/michaeldorfner/information-diffusion-boundaries-in-code-review>.
- 8 <https://opensource.org/licenses/mit>.
- 9 <https://github.com/jjl/orjson>.
- 10 <https://docs.python.org/3/library/json.html>.

ACKNOWLEDGEMENTS

We thank the participants of the deRSE'24 poster session for their insightful discussions and the anonymous reviewers for their insightful and constructive feedback.


FUNDING STATEMENT


This work was supported by the KKS Foundation through the SERT Project (Research Profile Grant 2018/010) at Blekinge Institute of Technology, Sweden.

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR AFFILIATIONS

Michael Dorner  orcid.org/0000-0001-8879-6450
Technische Hochschule Nürnberg Georg Simon Ohm, Germany;
Formerly with Blekinge Institute of Technology, Sweden

Andreas Bauer  orcid.org/0000-0002-2916-4020
Technische Hochschule Nürnberg Georg Simon Ohm, Germany;
Formerly with Blekinge Institute of Technology, Sweden

Florian Angermeir  orcid.org/0000-0001-7903-8236
fortiss, Germany; Blekinge Institute of Technology, Sweden

REFERENCES

1. **Ayala C, Turhan B, Franch X, Juristo N.** Use and Misuse of the Term “Experiment” in Mining Software Repositories Research. *IEEE Transactions on Software Engineering*. 2022;48(11):4229–4248. DOI: <https://doi.org/10.1109/TSE.2021.3113558>
2. **Bertrand N, Devkate A, Lamprecht AL.** Introducing an RSE Course at the University of Potsdam – First-Time Experiences and Plans for the Next Iteration. In: *deRSE24 – Conference for Research Software Engineering in Germany*; 2024.
3. **Cataldo M, Mockus A, Roberts JA, Herbsleb JD.** Software Dependencies, Work Dependencies, and Their Impact on Failures. *IEEE Transactions on Software Engineering*. 2009;35(6):864–878. DOI: <https://doi.org/10.1109/TSE.2009.42>
4. **Chen Z, Memon A, Luo B.** Combining research and education of software testing. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. 2014;1179–1180. DOI: <https://doi.org/10.1145/2554850.2555124>
5. **Cohen J, Katz DS, Barker M, Chue Hong N, Haines R, Jay C.** The Four Pillars of Research Software Engineering. *IEEE Software*. 2021;38(1):97–105. DOI: <https://doi.org/10.1109/MS.2020.2973362>
6. **Deng L, Dehlinger J, Chakraborty S.** Teaching Software Testing with Free and Open Source Software. In: *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*; 2020. pp. 412–418. DOI: <https://doi.org/10.1109/ICSTW50294.2020.00074>
7. **Dorner M, Bauer A.** *michaeldorfner/information-diffusion-boundaries-in-code-review: 1.0*. Version 1.0; 2023. DOI: <https://doi.org/10.5281/zenodo.10417852>
8. **Dorner M, Mendez D, Wnuk K, Zabardast E, Czerwonka J.** The Upper Bound of Information Diffusion in Code Review. *Empirical Software Engineering*; 2024 (Forthcoming).
9. **Dorner M, Smite D, Mendez D, Wnuk K, Czerwonka J.** Only Time Will Tell: Modelling Information Diffusion in Code Review with Time-Varying Hypergraphs. In: *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*; 2022. pp. 195–204. DOI: <https://doi.org/10.1145/3544902.3546254>
10. **Eisty NU, Carver JC.** Testing research software: a survey. *Empirical Software Engineering*. 2022;27. DOI: <https://doi.org/10.1007/s10664-022-10184-9>
11. **Fritz J, Speck R.** Do Scientists Write About Software? An RSE Publication Monitor. Tech. rep. Jülich Supercomputing Center; 2024. DOI: <https://doi.org/10.34734/FZJ-2024-01802>
12. **Garousi V, Felderer M, Kuhmann M, Herkiloğlu K.** What Industry Wants from Academia in Software Testing?: Hearing Practitioners’ Opinions. In: *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. Karlskrona Sweden: ACM; 2017. pp. 65–69. DOI: <https://doi.org/10.1145/3084226.3084264>
13. **Garousi V, Mäntylä MV.** A Systematic Literature Review of Literature Reviews in Software Testing. *Information and Software Technology*. 2016;80:195–216. DOI: <https://doi.org/10.1016/j.infsof.2016.09.002>

1. **Ayala C, Turhan B, Franch X, Juristo N.** Use and Misuse of the Term “Experiment” in Mining Software Repositories

14. **Garousi V, Rainer A, Lauvås P, Arcuri A.** Software-testing education: A systematic literature mapping. *Journal of Systems and Software*. 2020;165:110570. DOI: <https://doi.org/10.1016/j.jss.2020.110570>
15. **Gilson G, Ott S, Ledesma NR, Prabhu A, Porquet-Lupine J.** Design and Evaluation of “The Missing CS Class,” a Student-led Undergraduate Course to Reduce the Academia-industry Gap. In: *SIGCSE*; 2022. pp. 467–473. DOI: <https://doi.org/10.1145/3478431.3499422>
16. **Goth F, Alves R, Braun M, Castro LJ, Chourdakis G, Christ S, Cohen J, Erxleben F, Grad JN, Hagdorn M, Hodges T, Juckeland G, Kempf D, Lamprecht AL, Linxweiler J, Löffler F, Martone M, Schwarzmeier M, Seibold H, Thiele JP, von Waldow H, Wittke S.** *Foundational Competencies and Responsibilities of a Research Software Engineer*; 2024.
17. **Juckeland G.** Introduction to Research Software Engineering – Experiences from first time RSE class at the Computer Science Faculty of TU Dresden. In: *deRSE24 – Conference for Research Software Engineering in Germany*; 2024.
18. **Krutz DE, Malachowsky SA, Reichlmayr T.** Using a real world project in a software testing course. *Proceedings of the 45th ACM technical symposium on Computer science education*; 2014. pp. 49–54. DOI: <https://doi.org/10.1145/2538862.2538955>
19. **Mendez D, Graziotin D, Wagner S, Seibold H.** *Open Science in Software Engineering*. Springer International Publishing; 2020. pp. 477–501. DOI: https://doi.org/10.1007/978-3-030-32489-6_17
20. **Nangia U, Katz DS.** *Track 1 Paper: Surveying the U.S. National Postdoctoral Association Regarding Software Use and Training in Research*; 2017. DOI: <https://doi.org/10.6084/m9.figshare.5328442.v3>
21. **Riehle D.** The Future of the Open Source Definition. *Computer*. 2023;56(12):95–99. DOI: <https://doi.org/10.1109/MC.2023.3311648>
22. **Sánchez-Gordón ML, Colomo-Palacios R.** From Certifications to International Standards in Software Testing: Mapping from ISQTB to ISO/IEC/IEEE 29119-2. In: Larrucea X, Santamaria I, O’Connor RV, Messnarz R, editors. *Systems, Software and Services Process Improvement*. Cham, Switzerland: Springer International Publishing; 2018. pp. 43–55. DOI: https://doi.org/10.1007/978-3-319-97925-0_4
23. **Stol KJ, Fitzgerald B.** The ABC of Software Engineering Research. *ACM Transactions on Software Engineering and Methodology*. 2018;27(3):1–51. DOI: <https://doi.org/10.1145/3241743>
24. **Venson E, Alfayez R.** Bridging Theory to Practice in Software Testing Teaching through Team-based Learning (TBL) and Open Source Software (OSS) Contribution. In: *The 46th International Conference on Software Engineering Education and Training (ICSE-SEET’24)*; 2024. DOI: <https://doi.org/10.1145/3639474.3640081>
25. **Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A.** *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. DOI: <https://doi.org/10.1007/978-3-642-29044-2>

TO CITE THIS ARTICLE:

Dorner M, Bauer A, Angermeier F. 2026 When Research Software Goes to Class: Lessons From Embedding Research Software Into Teaching. *Journal of Open Research Software*, 14: 19. DOI: <https://doi.org/10.5334/jors.548>

Submitted: 19 December 2024 **Accepted:** 15 January 2026 **Published:** 23 March 2026

COPYRIGHT:

© 2026 The Author(s). This is an open-access article distributed under the terms of the Creative Commons Attribution 4.0 International License (CC-BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited. See <http://creativecommons.org/licenses/by/4.0/>.

Journal of Open Research Software is a peer-reviewed open access journal published by Ubiquity Press.