Science, Engineering and Technology Vol. 5, No. 2, pp. 20-28
www.setjournal.com *https://doi.org/10.54327/set2025/v5.i2.305*

A Comparative Case Study of Two Pedagogical Approaches in Web Development Education: From a Traditional Java Environment to a Modern Ruby on Rails Ecosystem

Keiichi Takahashi

Humanity-Oriented Science and Engineering, Kindai University, Iizuka 8208555, Japan.

Abstract

This study presents a comparative case study of the evolution of a software development course at Kindai University. We analyze two distinct pedagogical ecosystems: a traditional course based on Java Servlet/JSP with a local integrated development environment (IDE), and its subsequent iteration, a modern course employing the Ruby on Rails framework (a Web Application Framework, or WAF), Git for version control, a cloud-based IDE, and Platform as a Service (PaaS) for deployment. This study was not a controlled experiment isolating the effects of a WAF but rather an exploratory analysis of how a shift in the entire toolchain impacted student outcomes and perceptions. Quantitative analyses of student projects over three years for each course revealed that the modern Ruby-based ecosystem resulted in applications with approximately 50% more screens and screen transitions, despite requiring approximately 40% less source code. Furthermore, student surveys indicated significantly higher comprehension and interest in the modern courses. However, the number of data models and user stories remained consistent, suggesting that upstream design thinking was less affected by the technology stack. These findings suggest that adopting a modern, integrated development ecosystem can foster a more productive and engaging learning experience. We conclude by discussing the implications of these findings for curriculum design, emphasizing the value of incorporating contemporary, industry-aligned toolchains into software engineering education, while acknowledging that the observed benefits stem from the synergistic effect of multiple technologies rather than from a single component.

Keywords: *Computer Science Education, Software Engineering Curriculum, Web Application Frameworks (WAF), Software Development Toolchain, Comparative Case Study.*

1. Introduction

In recent years, web application development technologies have evolved rapidly. Accordingly, in software development education at institutions of higher learning, the introduction of tools and techniques aligned with real-world practices is increasingly required [1]. Web Application Frameworks (WAFs) have become essential technologies in modern software development environments, with increasing adoption in educational settings [2].

In contrast, WAFs are equipped with numerous auto-generation functions and support mechanisms through libraries, which contribute to improving development efficiency. However, these features tend to create a "black-box" effect, obscuring internal processes and potentially hindering the fundamental understanding of the model by beginners [3]. Despite these advantages, the contribution of WAF to students' learning outcomes in educational contexts remains insufficiently verified.

Corresponding author: *Keiichi Takahashi (ktakahas@fuk.kindai.ac.jp)*
Received: *4 July 2025*; Revised: *2 October 2025*; Accepted: *6 October 2025*; Published: *10 October 2025*

This paper presents a comparative case study of the evolution of a web application development course in a real-world educational setting. Rather than a controlled experiment, this study documents and analyzes the shift between two distinct pedagogical ecosystems: a traditional course centered on Java/JSP and its subsequent iteration built upon a modern, integrated ecosystem including the Ruby on Rails framework (WAF), Git, a cloud integrated development environment (IDE), and Platform as a Service (PaaS) deployment. This study aimed to explore the observed differences in student deliverables, development processes, and learning perceptions that arose during this technological transition. By analyzing this evolution, we seek to provide valuable insights for educators facing similar challenges in adapting their curricula to rapid changes in software development practices.

The remainder of this paper is organized as follows. Section 2 provides the study background and discusses the educational features of WAFs. Section 3 reviews the related literature and situates this study within that context. Section 4 details the course design and implementation. Section 5 presents the results of the analysis. Section 6 discusses the educational implications of the findings. Finally, Section 7 concludes the paper and suggests future research directions.

2. Background and Motivation

In web application development, WAFs are widely used as an essential foundation for improving productivity and maintainability of the applications. Representative WAFs include ASP.NET, Laravel, Ruby on Rails, and Django [4]. All of these are based on the Model-View-Controller (MVC) architecture and assist developers in efficiently producing high-quality software through features such as auto-generation and rich libraries [5].

WAFs are increasingly being introduced into educational settings, and by using them, students can experience development methods closer to industry practice. Because complex features, such as screen transitions and database integration, can be implemented in a relatively short time, this approach may enhance student motivation and a sense of achievement [2].

However, there are educational concerns associated with the introduction of WAFs. Because many processes in a WAF are encapsulated within the framework, it can be difficult for beginners to understand the operations that occur behind the scenes [6], [7]. For example, when errors occur, students who do not understand how the framework works may receive error messages from the framework's internal modules, and learning how to handle such errors may take time. Therefore, while education using WAFs can be efficient when students follow procedures correctly, it has been argued that it is not necessarily suitable for deepening students' understanding of the essential mechanisms of web applications [8], [9].

At Kindai University, a course on web application development using Java was offered. This Java-based course does not use a WAF; instead, it uses basic technologies such as Servlets, Java Server Pages (JSP), and Java Database Connectivity (JDBC) to help students gradually learn the basic structure and processing flow of web applications through development [10]. Although this method offers greater implementation flexibility, one drawback is that students must write a larger amount of code, making even simple functions time-consuming.

To address this issue, the department considered introducing WAF. Although WAFs exist for Java, Ruby on Rails (Rails) was selected because it is easier for beginners to learn and is widely used in industrial projects [11], [12]. Rails is written in Ruby, which emphasizes programmer-friendly code. Rails offer concise coding and extensive auto-generation features that allow even beginners to implement core functions quickly and easily. Additionally, Rails works well with modern development tools such as Git for version control, Render for web app deployment, and AWS Cloud9, a cloud-based IDE, making it a good foundation for incorporating modern development practices into education [13], [14].

Based on this background, the present study compares the differences between Java- and Ruby-based courses in the same curriculum. By aligning conditions such as development time, instructional structure, team composition, and submission format, we aim to clarify how the introduction of a WAF affects learning outcomes and student activities.

3. Related Work

In recent years, software development education has increasingly incorporated Project-Based Learning (PBL)

to help students cultivate technical skills and teamwork abilities through hands-on development experiences [15]. Many examples have reported the use of web application development as a central activity, enabling students to learn development workflows and tools that closely resemble actual practices.

In particular, the use of Web Application Frameworks (WAFs), such as Rails and Django, is expanding in educational contexts both domestically and internationally [14], [16]. Using WAFs, complex processes such as screen generation and database integration can be implemented more easily, enabling students to build practical web applications within a short period. Because even beginners can produce visible results with relatively little effort, this approach is also considered effective for boosting motivation and engagement [17].

Moreover, the use of Git and AWS Cloud9 in education is progressing [18]. Cloud-based IDEs, such as AWS Cloud9, reduce the burden of environment setup, which is often difficult for beginners, and allow students to continue learning outside the classroom. These tools are reported to enhance self-directed learning and promote better collaboration among team members [19].

However, few studies have compared the educational effects of WAFs with those of courses that do not use WAFs. Most prior research has focused on reporting the educational outcomes of individual courses that have adopted WAFs without quantitatively comparing the outcomes or learning behaviors between WAF and non-WAF approaches [20]. Although concerns have been raised that the abstraction provided by WAFs makes it harder for students to understand the underlying mechanisms of software, this issue has not been sufficiently examined in empirical studies [21].

Therefore, this study aims to clarify the quantitative and qualitative educational effects of WAFs by comparing the deliverables from a Ruby-based course that uses a WAF with those from a Java-based course that does not use a WAF. Such comparisons are rarely seen in previous studies and are expected to provide valuable insights into the advantages and challenges of using WAFs by beginners.

4. Methodology

4.1. Course design

The two courses analyzed in this study—one Java-based and one Ruby-based—were both programming practicum courses on web application development for third-year students at Kindai University. The courses were designed to provide hands-on experience of the full development process through team-based projects that reflect real-world practices.

The two courses shared the following structural elements.

- Classes held once a week, consisting of two consecutive periods (3 hours total) for 14 sessions
- *First 6 sessions:* Individual learning of advanced web development techniques
- *Final 8 sessions:* Planning, designing, and implementing a web application in two-person teams
- *Final session:* Presentation of deliverables

Team development began with upstream processes, such as defining user stories, designing data models, and planning screen transitions, followed by implementation using the appropriate development tools. Students were allowed to freely choose their development themes; however, instructors provided advice during the planning phase to ensure that the projects were both feasible and educationally effective.

The key differences between the courses were the development tools and environments used (Figure 1).

- *Java-based course:* Web application development using Java Servlet, JSP, and JDBC. Eclipse was used as the IDE, and source code was manually shared among team members, and no version-control system was used.
- *Ruby-based course:* Development was conducted using WAF Ruby on Rails. AWS Cloud9 was used as the development environment. Git was used for version control, GitHub as the remote repository, and deployment was performed using the Render platform.

Teams were formed by instructors based on students' performance in the individual technical exercises in the first half of the course, with the goal of balancing skill levels within each team.

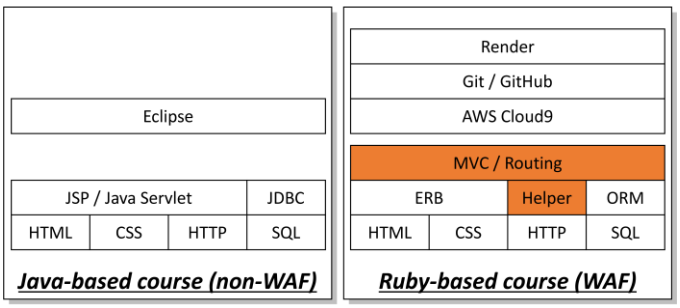

Figure 1. Technical components covered in the Java- and Ruby-based courses. The orange-colored elements indicate the components supported by the WAF.

4.2. Grading criteria

Students' grades were determined based on four components.

- *Individual reports (30%):* Progress on individual technical exercises and written reports during the team development phase
- *Developed system (40%):* Quality of the final application, including design, implementation, completeness, and usability
- *Team contribution (20%):* Quantitatively assessed using each member's share of the total lines of code (LOC)
- *Presentation (10%):* Quality of the final presentation, including explanation and supporting materials

Team contribution was calculated as each student's LOC divided by the team's total LOC. If a member's contribution was significantly low, their score was adjusted accordingly.

4.3. Development support environment and educational considerations

In the Ruby-based course, Amazon's AWS Cloud9 platform (provided for educational use) was employed as the development environment to eliminate the complexity of local setup. This allowed all students to work within a unified Linux environment, operate terminals, and experience practices such as version control and deployment to a PaaS, closely resembling real-world software development.

In contrast, the Java-based course relied on desktop computers in the university's computer lab, which often led to delays in setup and team-based file sharing.

4.4. Study design and limitations

It is crucial to acknowledge the design and limitations of this study. This research is presented as an exploratory comparative case study that examines the evolution of a course over several years rather than as a controlled experiment. As such, there are several significant confounding variables between the two ecosystems. The key differences include (1) the programming language (Java vs. Ruby), (2) the use of a WAF, (3) the mandatory use of a version control system (Git/GitHub) in the Ruby course, (4) the development environment (a local IDE vs. a cloud-based IDE), and (5) the deployment method (manual vs. PaaS). Therefore, this study does not seek to isolate the causal effects of any single variable. Instead, it aims to provide a comprehensive analysis of the observed differences in outcomes and student perceptions when a pedagogical approach transitions from a traditional to a modern integrated one. The quantitative metrics presented, such as LOC, should be interpreted with caution. Given the differences in languages and frameworks, LOC was not used here as a direct measure of productivity, but rather as a descriptive indicator to illustrate the different nature of the development work undertaken by students in each environment.

5. Results and Analysis

5.1. Number of teams

The Java- and Ruby-based courses were each offered three times over six years. Figure 2 shows the number of teams for each type of implementation in the project. Each team consisted of two members. The deliverables analyzed for both the Java- and Ruby-based courses were aggregated across the three offerings. There were 25 and 28 teams in the Java-and Ruby-based courses, respectively.

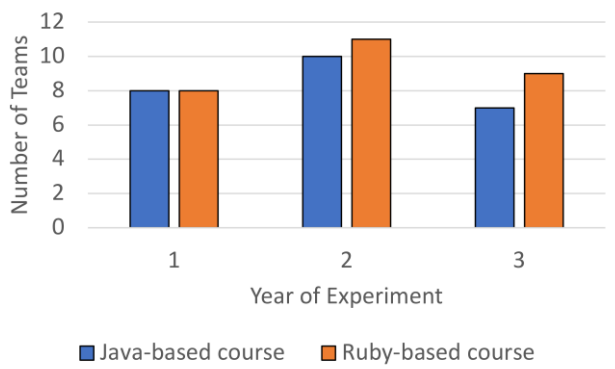

Figure 2. Number of teams across the three implementations of the Java-based and Ruby-based courses.

5.2. Source code LOC (Lines of Code)

The source code produced by each team in both courses was aggregated, and the number of LOC was measured. For the Ruby-based course, files automatically generated by Rails were excluded, and only the code written directly by students was included in the count. A boxplot of these results is presented in Figure 3. The median LOC for the Java-based course was 1252, while the Ruby-based course had a median LOC of 738. Thus, the students in the Java-based course wrote approximately 70% more source code.

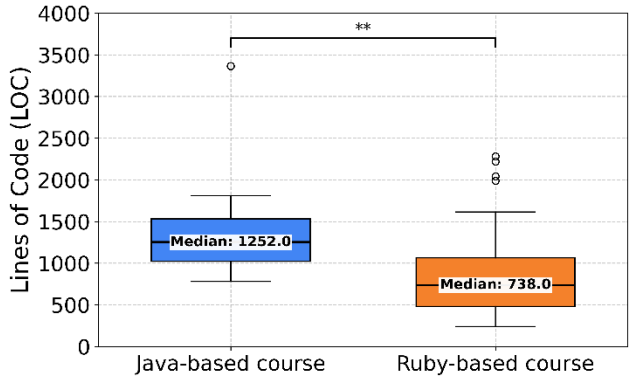

Figure 3. LOC comparison between Java-based and Ruby-based courses. The median LOC for the Java-based course was 70% higher than that for the Ruby-based course.

Java generally requires more lines of code than Ruby. However, because the course durations were identical, this result suggests that the students in the Java-based course had to spend more time writing code. Welch's t-test indicated a statistically significant difference between the two groups ($p < 0.01$). Therefore, it was demonstrated that the non-WAF Java-based teams produced significantly more code than the Ruby-based teams that used WAF.

5.3. Number of screens and screen transitions

Figures 4 and 5 show the number of screens and screen transitions implemented by each team in both courses. The median number of screens was six for Java-based applications and nine for Ruby-based applications. The median number of screen transitions was 9 for Java-based applications and 13.5 for Ruby-based applications. In both cases, the Ruby-based teams implemented approximately 50% more than the Java-based teams. Welch's t-test confirmed statistically significant differences ($p < 0.05$).

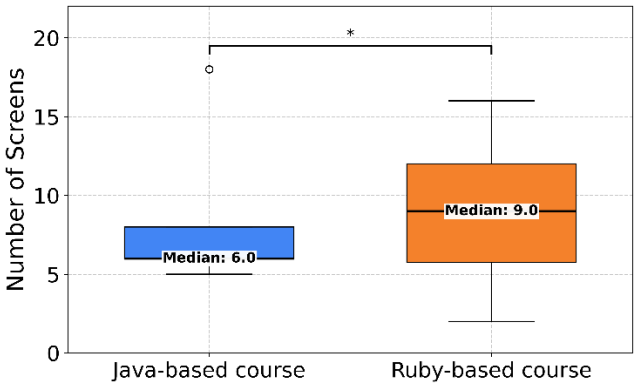

Figure 4. Number of screens implemented. The Ruby-based course had 50% more screens than the Java-based course, based on the median values.

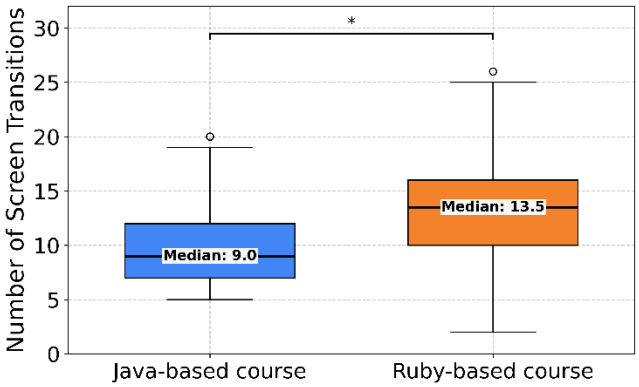

Figure 5. Number of screen transitions: The Ruby-based course had 50% more transitions than the Java-based course based on median values.

Web applications typically require dynamic screen displays based on various data. As the number of screens increases, the complexity of data coordination between the screens also increases. Therefore, comparing the screen and transition counts allows for an indirect evaluation of the functional complexity. These results suggest that Ruby-based teams implemented more functions.

5.4. Number of models and user stories

The number of models corresponded to the number of the database tables. In general, a larger number of models implies greater implementation complexity. The number of user stories was obtained from each team's submitted design documents. There were no restrictions on the number of user stories; the teams were free to define them during the planning phase of the project.

Figures 6 and 7 show the number of models and user stories, respectively. The median number of models was four for both the courses. The median number of user stories was 6 for Java-based and 5.5 for Ruby-based, indicating a slight difference. Welch's t-test showed no statistically significant difference between the two groups.

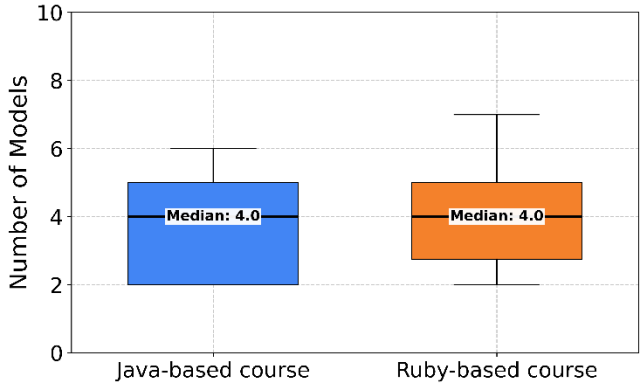

Figure 6. Number of models. Both courses had identical median values.

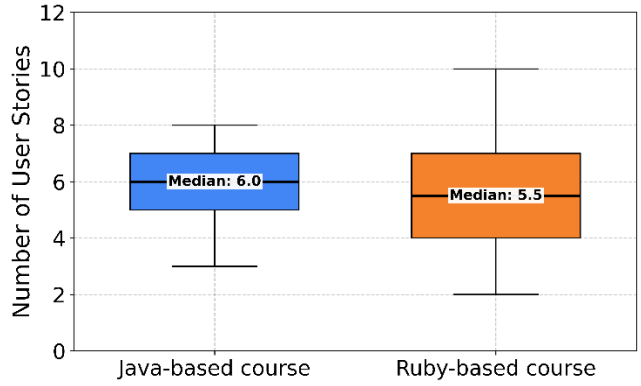

Figure 7. Number of user stories. The Java-based course had a slightly higher median, but the difference was minimal.

Because the number of models and user stories is determined during the initial planning phase, it likely depends on students' ability to extract requirements. The lack of a significant difference suggests that students' application planning skills were similar across both courses.

5.5. Course evaluation survey

At this university, course evaluation surveys are conducted during the final sessions of each course. The survey consisted of standardized questions on a five-point Likert scale, supplemented with free-text comments. The responses were anonymized and aggregated. Outliers (e.g., incomplete or inconsistent answers) were excluded. Welch's t-test was employed to assess statistical significance. For this analysis, we extracted responses related to students' behavior and attitude toward the course and compared the results between the Java-based course (N=50) and the Ruby-based course (N=58). The following six questions were analyzed. Responses to Q1–Q4 were measured on a five-point Likert scale.

- *Q1: Did you understand the course content?*
- *Q2: Did the course stimulate your interest in the subject?*
- *Q3: Did you stay focused during class and avoid unrelated conversations or activities?*
- *Q4: Was the classroom environment and equipment satisfactory?*
- *Q5: How many hours per week, on average, did you spend on self-study outside the class?*
- *Q6: Please rate this course on a scale of 1–10.*

The average responses to Q1–Q4 are shown in Figure 8. For all items, the Ruby-based course received higher ratings from the students. Welch's t-test indicated statistically significant differences for Q1, Q2, and Q4.

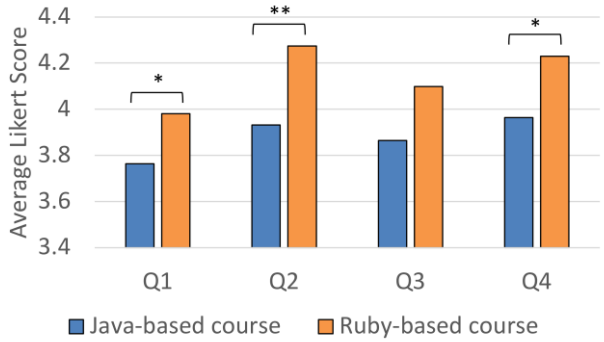

Figure 8. Average scores for Q1 to Q4. The Ruby-based course received higher ratings in all categories, with significant differences in Q1, Q2, and Q4 scores.

The average responses to Q5 and Q6 are presented in Table 1. Q5 assessed students' study time outside class, and Q6 assessed their overall evaluation of the course. No significant differences were observed between the two groups.

Table 1. Average scores for Q5 and Q6 (self-study time and overall course evaluation).

Question	Java-based course	Ruby-based course
Q5	2.6 hours	2.9 hours
Q6	8.1 / 10	8.3 / 10

6. Discussion

The results of this comparative case study highlight significant differences in student outcomes and perceptions between a traditional Java-based course and a course built on a modern integrated Rails ecosystem. This section discusses the potential factors contributing to these differences, interpreting them not as the effect of a single variable but as the synergistic outcome of a shift in the entire pedagogical toolchain.

6.1. The impact of a modern ecosystem on development outcomes

The analysis revealed that students in the Ruby-based course produced applications with approximately 50% more screens and transitions, despite writing approximately 40% less code than their Java-based counterparts did. This notable difference in output can be attributed to a combination of factors in the modern ecosystem. The Rails framework itself contributed significantly through features such as scaffolding for CRUD operations and a strong adherence to the "Convention over Configuration" (CoC) principle, which reduced boilerplate code. The rich ecosystem of libraries (gems) further lowers the barrier to implementing complex features. However, other components of the ecosystem are also critical. The use of Git and GitHub facilitated collaboration, reducing the friction of manual file sharing that characterized the Java course. Furthermore, the cloud-based IDE (AWS Cloud9) eliminated time-consuming local environment setup and troubleshooting, allowing students to focus more on the limited class time on development itself. Thus, the observed productivity gain is likely a composite effect of the framework's efficiency, smoother collaboration, and a frictionless development environment.

6.2. The Role of the Toolchain in the Student Learning Experience

Student surveys provide insights into the qualitative aspects of the learning experience. The significantly higher scores for the Ruby-based course in Q1 (comprehension) and Q2 (interest) suggest a more engaging and comprehensible learning experience. This can be linked to several factors. For instance, the higher score in Q1 (comprehension) may be partly attributable to the clear structure imposed by the MVC architecture, which provides students with a consistent and predictable way of organizing their application. More importantly, exposure to industry-standard tools such as Git, GitHub, and cloud deployment (Render) likely boosted students' interest and motivation (Q2). The most striking difference was in Q4 (classroom environment and equipment satisfactory), where the Ruby-based course was rated significantly higher than the others. This result cannot be reasonably attributed to the WAF but directly corresponds to the use of the AWS Cloud9 IDE. It eliminated the common frustrations of local environment configuration, providing a consistent and accessible platform for all students, which is a crucial factor in student satisfaction.

6.3. Depth, breadth, and the evolving role of fundamental knowledge

This study also sheds light on the classic trade-off between the depth and breadth of technical understanding. The Java-based course, by requiring students to handle HTTP requests and database connections manually using Servlets and JDBC, arguably provided a deeper understanding of the fundamental mechanisms of web applications. In contrast, the modern Ruby ecosystem, while abstracting these low-level details, offers a greater breadth of experience. Students engaged in a complete, modern development workflow, including version control, cloud development, and automated deployment. Neither approach is inherently superior; rather, they serve different educational goals. In particular, the Java/JSP-based course, although less popular among students, provides valuable opportunities for them to understand the lower-level mechanisms of web applications in detail, which can be beneficial for their long-term competence as software engineers. The findings support the value of a curriculum that balances both aspects: ensuring that

students grasp fundamental concepts is critical for long-term adaptability, while exposure to modern, high-productivity toolchains is essential for preparing them for contemporary industry practices.

6.4. Implications for curriculum design

Based on the findings of this study, several implications for curriculum design in software education can be suggested. First, a curriculum that begins with fundamental technologies before introducing more abstract and high-level frameworks may be effective. Second, although the use of modern, integrated environments is useful for producing industry-ready graduates, conceptual understanding must not be sacrificed. Third, regardless of the core technology taught, incorporating tools such as Git and cloud-based IDEs can significantly enhance the quality of the learning environment and collaborative experience. Finally, with the efficiency gains from modern toolchains, it is feasible to design projects that are larger in scale or are structured around iterative development cycles.

7. Conclusion

This study presents a comparative case study of the evolution of a web development course, analyzing the shift from a traditional Java-based environment to a modern integrated Rails ecosystem. Our findings show that the modern ecosystem enabled students to develop more functionally complex applications with significantly less code and was associated with higher levels of student-reported comprehension and interest than the traditional ecosystem. We conclude that the adoption of a comprehensive, modern toolchain—encompassing not only a WAF but also version control, a cloud IDE, and automated deployment—can foster a more productive, motivating, and educationally valuable experience for students preparing to enter the software industry than the traditional toolchain can.

However, it is critical to reiterate that these benefits cannot be attributed to the WAF alone, but rather appear to be the synergistic effect of the entire technology stack used. The abstraction provided by these modern tools may also reduce students' exposure to the fundamental mechanics of web applications, highlighting the importance of a balanced curriculum that intentionally addresses both foundational concepts and modern practices in web development. This balance ensures that while students benefit from modern toolchains, they also acquire deeper technical knowledge that remains essential for long-term adaptability. Our primary recommendation for educators is to consider the entire development ecosystem when designing courses and actively incorporate industry-aligned tools to bridge the gap between academia and professional practice.

Future research should address the limitations of this study. More controlled experiments are needed, for example, comparing courses that use the same programming language with and without a WAF (e.g., a plain Java course versus a Java Spring Boot course). To rigorously evaluate learning outcomes, subsequent studies should employ direct measures of knowledge acquisition, such as pre- and post-course conceptual tests, in addition to student surveys. Finally, the growing influence of AI-powered code generation tools presents a new, significant variable in software engineering education, and their impact on framework-based teaching warrants dedicated investigation.

Competing Interest Statement

The authors declare that they have no known conflicts financial interests or personal relationships that could have influenced the work reported in this article.

Data Availability Statement

No data or additional materials were utilized for the research described in the article.

References

[1] N. Gharaibeh, "Improving Web Application Development Course," in *Proc. Fifth National Conf. Saudi Computers Colleges (NCCC)*, 2022, pp. 165-171, doi: 10.1109/NCCC57165.2022.10067861.

[2] S. Ivanova and G. Georgiev, "Using modern web frameworks when developing an education application: a practical approach," *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, 2019, pp. 1485-1491, doi: 10.23919/MIPRO.2019.8756914.

[3] P. Hrkút, M. Meško and M. Ďuračík, "A Custom Framework for Innovative Approach of Teaching Web Development Courses," *2024 36th Conference of Open Innovations Association (FRUCT)*, Lappeenranta, Finland, 2024, pp. 256-261, doi: 10.23919/FRUCT64283.2024.10749859.

[4] A. Aborujilah, J. Adamu, S. M. Shariff and Z. Awang Long, "Descriptive Analysis of Built-in Security Features in Web Development Frameworks," *2022 16th International Conference on Ubiquitous Information Management and Communication (IMCOM)*, Seoul, Korea, Republic of, 2022, pp. 1-8, doi: 10.1109/IMCOM53663.2022.9721750.

[5] S. Ahmad, T. Rana, and A. Maqbool, "A Model-Driven Framework for the Development of MVC-Based (Web) Application," *Arabian Journal for Science and Engineering*, vol. 47, pp. 1733-1747, 2021, doi: 10.1007/s13369-021-06087-4.

[6] B. Correa, F. Isaza, R. Mazo, R. Mazo, and G. Giraldo, "CME – A Web Application Framework Learning Technique Based on Concerns, Micro-Learning and Examples," In *Mikkonen, T., Klamma, R., Hernández, J. (eds) Web Engineering. ICWE 2018*. Lecture Notes in Computer Science, vol 10845. Springer, Cham. https://doi.org/10.1007/978-3-319-91662-0_2.

[7] M. Miura, "Block Sweetie: Learning Web Application Development by Block Arrangement," *2018 Thirteenth International Conference on Knowledge, Information and Creativity Support Systems (KICSS)*, Pattaya, Thailand, 2018, pp. 1-6, doi: 10.1109/KICSS45055.2018.8950653.

[8] N. Ichanska, "Tools and instruments for developing a web application with student knowledge level testing," *Mechanics And Mathematical Methods*, 2024, doi: 10.31650/2618-0650-2024-6-1-95-106.

[9] C. R. Jaimez-González and M. Castillo-Cortes, "Web Application to Support the Learning of Programming Through the Graphic Visualization of Programs", *Int. J. Emerg. Technol. Learn.*, vol. 15, no. 06, pp. pp. 33–49, Mar. 2020, doi: 10.3991/ijet.v15i06.12157.

[10] M. Gribanova-Podkina, "Database Connection Technologies from JSP Pages and Java Web Application Servlets," *Cybernetics and Programming*, 2019, doi: 10.25136/2306-4196.2019.2.19589.

[11] P. Łuczak, A. Poniszewska-Marańda, and V. Karovič, "The Process of Creating Web Applications in Ruby on Rails," in *Developments in Information & Knowledge Management for Business Applications*, 2020, doi: 10.1007/978-3-030-62151-3_9.

[12] H. F. Putri, R. S. Perdana, Y. Gunawan, K. Manaf, H. H. Solihin and B. Subaeki, "Analysis of Supporting Information Systems for Seminar Activities Using the Ruby on Rails Framework," *2023 17th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, Lombok, Indonesia, 2023, pp. 1-4, doi: 10.1109/TSSA59948.2023.10367025.

[13] D. Naranjo, J. Prieto, G. Moltó, and A. Calatrava, "A Visual Dashboard to Track Learning Analytics for Educational Cloud Computing," *Sensors (Basel, Switzerland)*, vol. 19, no. 13, pp. 2952, 2019, doi: 10.3390/s19132952.

[14] L. Lu, "Design and Implementation of an Interactive Information System for University Education under the Cloud Service Model," *2020 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS)*, Shenyang, China, 2020, pp. 377-381, doi: 10.1109/TOCS50858.2020.9339620.

[15] D. Ståhl, K. Sandahl and L. Buffoni, "An Eco-System Approach to Project-Based Learning in Software Engineering Education," in *IEEE Transactions on Education*, vol. 65, no. 4, pp. 514-523, Nov. 2022, doi: 10.1109/TE.2021.3137344.

[16] A. Paul, M. Dela, R. Paul, and P. Abad, "Leveraging Technology for Teaching and Learning: Developing a Django-based Quiz Application for Education," *Journal of Information Systems Engineering and Management*, 2025, doi: 10.52783/jisem.v10i13s.2008.

[17] M. Wilkinson, A. Schechter, B. Lukens, I. Wright, and J. Cardarelli, "A Web Development and Cloud Deployment Framework for a Software Engineering Course," in *Proc. 55th ACM Technical Symposium on Computer Science Education V. 2*, 2024, doi: 10.1145/3626253.3635410.

[18] Y. Borse and S. Gokhale, "Cloud Computing Platform for Education System: A Review," *International Journal of Computer Applications*, vol. 177, no. 9, 2019, doi: 10.5120/ijca2019919475.

[19] N. Baanqud, H. Al-Samarraie, A. Alzahrani, and O. Alfarraj, "Engagement in cloud-supported collaborative learning and student knowledge construction: a modeling study," *International Journal of Educational Technology in Higher Education*, vol. 17, no. 56, 2020, doi: 10.1186/s41239-020-00232-z.

[20] B. Pedraça de Souza, D. S. Costa, D. O. Costa, B. A. Bonifácio and P. S. Fernandes, "Using Frameworks for Rapid Applications Development as Learning Object for Teaching Web Programming," *2018 XIII Latin American Conference on Learning Technologies (LACLO)*, Sao Paulo, Brazil, 2018, pp. 356-362, doi: 10.1109/LACLO.2018.00068.

[21] J. Cito, J. Shen, and M. Rinard, "An Empirical Study on the Impact of Deimplicitization on Comprehension in Programs Using Application Frameworks," in *Proc. IEEE/ACM 17th Int. Conf. Mining Software Repositories (MSR)*, 2020, pp. 598-601, doi: 10.1145/3379597.3387507.