An Educational Software Tool for Critiquing and Supporting Software Design Courses

John Fajinmi * and Joseph Oloyede

Posted Date: 15 January 2025

doi: 10.20944/preprints202501.1130.v1

Keywords: Software design education; Automated; critique tool; Interactive learning; Personalized feedback; Design principles; UML diagrams; Machine learning in education; Software engineering pedagogy Scalable educational tools

Article

An Educational Software Tool for Critiquing and Supporting Software Design Courses

John Fajinmi

Affiliation 1

Abstract: This paper introduces an innovative educational software tool designed to critique and support students in software design courses. The tool aims to provide constructive feedback on various aspects of software design, focusing on key principles such as functionality, usability and architectural coherence. By automating the critique process, it offers students real-time insights into their design choices, fostering deeper learning and enhancing the quality of their work. The tool includes features such as error detection, design pattern identification, and suggestions for improvements, making it an invaluable resource for both students and instructors. Additionally, the tool promotes an interactive learning environment by encouraging iterative design, collaboration, and self-assessment. This approach not only strengthens students' understanding of software design but also supports the development of critical thinking skills essential for the industry. Through this paper, we explore the design, implementation, and potential impact of the tool in improving educational outcomes in software design courses.

Keywords: software design education; Automated; critique tool; Interactive learning; personalized feedback; design principles; UML diagrams; machine learning in education; software engineering pedagogy; scalable educational tools

---

1. Introduction

A. Background and Context

Software design is a core component of computer science education, where students are tasked with learning both theoretical and practical aspects of creating robust, efficient, and user-friendly systems. Despite the emphasis on project-based learning and real-world problem-solving, students often struggle with making design decisions that align with best practices and industry standards. This challenge is exacerbated in large classrooms, where individualized feedback is difficult to provide. The rapid evolution of software development methodologies further complicates the ability to keep up with current trends, tools, and techniques.

B. Need for an Educational Software Tool

Given the complexities of software design and the constraints in traditional educational settings, there is an increasing need for an educational tool that can assist students in their design process. Such a tool would provide timely, constructive feedback on design choices, highlight common mistakes, and suggest improvements. It would serve not only as a critique mechanism but also as a learning aid that supports the development of critical thinking and problem-solving skills. Additionally, this tool could help bridge the gap between theoretical concepts and practical application, ensuring that students gain a solid foundation in software design.

C. Objective of the Tool

The primary objective of this tool is to provide students with a comprehensive, interactive platform that critiques and supports their software design work throughout the learning process. By

 doi:10.20944/preprints202501.1130.v1

leveraging automated design analysis, the tool aims to offer real-time feedback on a student's design, focusing on essential design elements such as structure, functionality, usability, and adherence to design patterns. Furthermore, it will encourage self-assessment, enable iterative design, and foster collaboration among students. The tool's ultimate goal is to enhance the learning experience, improve design skills, and better prepare students for professional software development tasks.

2. Literature Review

A. Existing Educational Tools for Software Design

Over the years, various educational tools have been developed to assist students in learning software design. Many of these tools focus on specific aspects of the design process, such as code generation, architecture modeling, or user interface design. For instance, tools like Visual Paradigm and Lucidchart provide students with platforms for creating UML diagrams and visual models, helping them understand and represent the structural aspects of a software system. Additionally, some learning management systems (LMS) integrate feedback mechanisms that allow instructors to evaluate students' work and provide suggestions for improvement.

Despite these advances, many existing tools lack comprehensive critique capabilities or fail to provide timely, personalized feedback. Tools like CodeCombat and Scratch, while excellent for learning basic programming concepts, do not specifically target the higher-level design skills required in software engineering courses. Moreover, most tools are either too simplistic or too focused on coding rather than on holistic design evaluation. This gap underscores the need for an integrated tool that critiques design decisions, guides students in aligning with best practices, and provides actionable suggestions for improvement.

B. Theoretical Foundations

The development of an educational software tool for critiquing and supporting software design courses draws from several theoretical foundations related to learning, cognitive development, and educational technology. One of the key theories is constructivism, particularly the work of Piaget and Vygotsky, which emphasizes the importance of active learning, hands-on problem-solving, and social interaction in the learning process. According to this framework, students learn best when they engage in authentic tasks, receive feedback, and are encouraged to reflect on their learning journey. The tool proposed in this study aligns with these principles by offering iterative feedback, promoting critical thinking, and fostering a deeper understanding of software design through practice.

Another foundational theory is cognitive load theory (Sweller, 1988), which focuses on how instructional design can optimize the use of working memory. By automating design critiques, the tool aims to reduce the cognitive load of students by providing structured, digestible feedback that allows them to focus on solving complex design problems without being overwhelmed by excessive information. Additionally, the tool integrates principles from scaffolding, which involves providing temporary support to students until they can independently perform tasks. This allows the tool to assist students at different stages of their design process and to gradually reduce the level of support as their skills improve.

Furthermore, feedback intervention theory (Kluger & DeNisi, 1996) supports the idea that effective feedback can significantly improve performance, especially when it is specific, timely, and actionable. The educational software tool will leverage this theory to deliver constructive critiques that guide students toward improving their design solutions while encouraging continuous learning and refinement.

By combining these theories, the proposed tool is grounded in a robust pedagogical framework aimed at enhancing software design education and empowering students to develop the skills needed to succeed in the field.

 doi:10.20944/preprints202501.1130.v1

3. Tool Design and Development

A. Requirements and Features

The educational software tool for critiquing and supporting software design courses is designed to meet several essential requirements to ensure its effectiveness in a learning environment. These requirements can be categorized into functional, non-functional, and pedagogical aspects.

Functional Requirements:

- Automated Design Feedback: The tool must analyze students' design submissions (e.g., UML diagrams, class structures, design patterns) and provide immediate, automated feedback highlighting strengths, weaknesses, and areas for improvement.
- Design Pattern Recognition: It should be capable of identifying common design patterns (e.g., Singleton, Factory, Observer) used or misused in student submissions.
- Error Detection: The tool should detect structural issues such as incomplete designs, improper relationships between components, or violations of established design principles.
- Interactive Interface: The tool must offer an intuitive, user-friendly interface that allows students to easily input their designs and view feedback, along with suggestions for improvement.
- Real-Time Feedback: Immediate feedback will be given as students submit their designs, allowing for continuous learning without long delays.
- Iterative Design Support: The tool should encourage iterative improvement by allowing students to refine their designs based on critiques and resubmit them for further analysis.

Non-Functional Requirements:

- Scalability: The tool should handle multiple simultaneous users, especially in large classroom settings.
- Cross-Platform Compatibility: It should function on various operating systems and devices (e.g., web-based, mobile).
- Security and Privacy: The system must ensure that student data, such as design submissions and feedback, are securely stored and handled in compliance with privacy standards.
- Performance: The tool must deliver feedback with minimal delay, ensuring a seamless user experience.

Pedagogical Features:

- Guided Learning Path: The tool should provide scaffolding, helping students learn progressively by offering different levels of critique based on their expertise.
- Collaboration and Peer Review: Encourage peer interaction by allowing students to share their designs with peers for feedback and collaborative improvement.
- Self-Assessment Capabilities: The tool should help students assess their progress by reflecting on past critiques and identifying areas where improvement is needed.

B. System Architecture

The system architecture of the tool is designed to support scalability, flexibility, and ease of use. It consists of three main components: the front-end interface, the back-end analysis engine, and the database.

Front-End Interface:

- The front-end will be a web-based platform that provides a clean, intuitive user interface for students to interact with the tool. Students can upload their design submissions, view critiques, and track their progress. This interface will also provide a dashboard for instructors to monitor student performance and provide additional input if needed.

Back-End Analysis Engine:

- The core functionality of the tool lies in its back-end engine, which will use predefined algorithms to analyze the design submissions. The engine will evaluate UML diagrams, class relationships, and design patterns to provide relevant feedback. It will leverage machine

 doi:10.20944/preprints202501.1130.v1

learning techniques to improve its feedback accuracy over time, based on patterns of past student submissions.

Database:

- The database will store student profiles, design submissions, feedback logs, and other relevant information. It will track students' progress, providing instructors with data on individual and group performance. The database will also store predefined templates for design patterns, best practices, and guidelines to facilitate critique generation.

C. Critique Methodology

The critique methodology adopted by the tool is built on a combination of rule-based analysis, pattern recognition, and machine learning. The goal is to provide constructive, actionable feedback that enhances student learning.

Rule-Based Analysis:

The tool will initially rely on a rule-based system to analyze design elements. For example, it will check for the correct usage of design patterns, such as verifying whether the correct classes are instantiated in the correct contexts, or if dependencies are properly managed. If an error is detected, the system will provide specific feedback (e.g., "The Factory pattern is incorrectly applied in the class X. You should instantiate the product interface in the client class instead").

Pattern Recognition:

The tool will recognize common design patterns based on predefined templates and structures. If a student successfully implements a design pattern (e.g., MVC), the tool will acknowledge this and suggest possible improvements, such as optimizing the design for scalability or ensuring that the pattern adheres to SOLID principles.

Machine Learning (Advanced Feedback):

Over time, the tool will incorporate machine learning techniques to analyze past critiques and refine its feedback. By learning from common mistakes and successful designs submitted by students, the tool will provide increasingly personalized and accurate feedback. It will also identify trends, such as specific design weaknesses in a cohort of students, and tailor suggestions accordingly.

Feedback Types:

- Descriptive Feedback: Offers explanations on why a particular design choice may or may not be appropriate, encouraging understanding and learning.
- Corrective Feedback: Suggests alternative approaches or corrections to improve the design.
- Comparative Feedback: Provides examples of better design approaches or references from industry best practices to encourage continuous improvement.

This feedback methodology is intended to foster critical thinking, self-reflection, and a deeper understanding of software design principles, ultimately guiding students to create better, more efficient software systems.

4. Evaluation and Testing

A. Pilot Study or User Testing

To assess the effectiveness of the educational software tool, a pilot study will be conducted involving a sample of students enrolled in software design courses. The pilot study aims to evaluate the tool's usability, accuracy of feedback, and impact on student learning. The testing process will consist of the following steps:

Selection of Participants:

A group of students, ideally from different skill levels (beginner, intermediate, and advanced), will be selected to participate in the pilot study. This diverse group will allow for a comprehensive evaluation of how well the tool meets the needs of various learners.

 doi:10.20944/preprints202501.1130.v1

Training and Onboarding:

Students will be provided with a brief introduction to the tool, including a demonstration of its features and functionalities. This will ensure they understand how to use the platform and its feedback features effectively.

Design Task Completion:

Participants will be tasked with submitting their software design projects (e.g., UML diagrams, system architectures) through the tool. They will be encouraged to apply their existing knowledge and use the tool iteratively for design revisions.

Feedback Collection:

After submitting their designs, students will receive real-time critiques from the tool. Participants will be asked to reflect on the feedback, make revisions, and submit updated versions of their designs. They will also be encouraged to interact with the tool to receive multiple rounds of feedback.

Interviews and Surveys:

At the end of the study, students will complete a survey and participate in interviews to gather qualitative and quantitative data on their experiences. These surveys will address aspects such as usability, effectiveness of the feedback, usefulness of design suggestions, and overall satisfaction. Instructors may also be interviewed to gather their perspectives on how the tool supports their teaching objectives.

Data Analysis:

The collected data will be analyzed to determine the effectiveness of the tool. Key metrics, such as improvement in student design quality, engagement with the tool, and the perceived usefulness of feedback, will be examined to assess its impact on learning.

B. Metrics for Success

To evaluate the success of the tool, both qualitative and quantitative metrics will be used. These metrics will provide insights into the tool's effectiveness in enhancing student learning and improving design outcomes.

Usability Metrics:

- Ease of Use: Based on survey responses, students will rate how easy it was to navigate the tool and use its features.
- User Engagement: The frequency with which students use the tool and interact with its feedback features will be tracked. Higher engagement suggests the tool's value in the learning process.
- Feedback Interaction: The number of iterations a student completes (i.e., submitting a design, receiving feedback, making revisions) will be measured. More iterations indicate a more engaged and reflective learning process.

Learning Outcomes:

- Improvement in Design Quality: A pre- and post-test analysis will be conducted by comparing the initial and revised designs. Metrics such as adherence to best practices, correct application of design patterns, and overall design structure will be evaluated to gauge how well the tool improves students' design skills.
- Test Scores: If applicable, students' performance on relevant exams or assignments related to software design (such as conceptual tests on design patterns and principles) will be compared before and after using the tool.

Feedback Effectiveness:

- Perceived Usefulness of Feedback: Students will rate the usefulness of the feedback provided by the tool, including how actionable the critiques were and whether they helped improve their designs. This will be collected through surveys and interviews.

 doi:10.20944/preprints202501.1130.v1

- Accuracy of Feedback: Instructors or experts in software design will evaluate a sample of student submissions and their corresponding feedback. This will assess whether the tool provides accurate, relevant, and helpful suggestions for improvement.

User Satisfaction:

- Student Satisfaction: The level of satisfaction will be measured using Likert-scale questions in surveys, focusing on students' overall satisfaction with the tool and its impact on their learning experience.
- Instructor Satisfaction: Instructors' feedback will be gathered to evaluate how well the tool aligns with course goals, its potential to reduce their workload in providing individualized feedback, and its effectiveness in aiding students' learning.

Long-Term Impact on Learning:

- Retention of Knowledge: After a set period, a follow-up test or assessment will be given to participants to determine whether the tool has had a lasting impact on their understanding of software design principles.
- Application of Skills in Future Projects: In subsequent assignments or courses, students will be asked to apply the skills they developed using the tool. This can help evaluate the long-term benefits of using the tool in practical settings.

By gathering data on these metrics, the evaluation will provide a comprehensive understanding of the tool's effectiveness, identify areas for improvement, and guide future iterations of the system to ensure it meets the needs of students and instructors alike.

5. Challenges and Limitations

A. Technical Challenges

Complexity of Design Critique Algorithms:

Developing an automated critique system capable of accurately evaluating complex software designs is technically challenging. The tool must understand and interpret different types of design artifacts (e.g., UML diagrams, class structures, design patterns) and provide context-sensitive feedback. Ensuring the feedback is both relevant and accurate across various design approaches, while accounting for the diverse ways students might represent their designs, can be difficult.

Scalability and Performance:

As the tool is expected to handle multiple simultaneous users, especially in large educational settings, ensuring scalability and maintaining performance across many users is a key technical challenge. Optimizing the system for speed and responsiveness while providing real-time feedback will require careful design and the use of efficient algorithms.

Integration with Existing Tools:

In many educational environments, instructors and students may already be using other software tools for design, such as Visual Studio, Eclipse, or online collaboration platforms. Ensuring the new critique tool integrates seamlessly with these existing tools, while allowing for easy import/export of design files and maintaining compatibility with various file formats, presents an additional technical challenge.

Machine Learning Model Accuracy:

Incorporating machine learning into the feedback system is intended to improve the accuracy of critiques over time. However, ensuring that the model is trained on a sufficiently large and diverse dataset of student submissions to avoid biases or errors in feedback is an ongoing challenge. Additionally, the system must be continuously monitored and updated to maintain its relevance to evolving software design trends and methodologies.

B. Pedagogical Limitations

Over-Reliance on Automated Feedback:

 doi:10.20944/preprints202501.1130.v1

While automated feedback can be helpful, students may become overly reliant on the tool, potentially neglecting their ability to critically assess their own designs without external assistance. This may reduce the development of independent problem-solving and critical thinking skills, which are essential for professional software engineers. To mitigate this, the tool should encourage students to engage with feedback actively and think critically about suggested changes rather than simply accepting the tool's critique.

Contextual Understanding of Design Choices:

Software design is a highly context-dependent process, and a design that may not adhere to conventional best practices could still be appropriate for a particular problem or project. The tool may struggle to capture this nuance, potentially offering critiques that do not fully account for the specific goals or constraints of a given design task. As a result, the tool's feedback might be perceived as rigid or lacking in flexibility, which could discourage students from experimenting with novel or unconventional design approaches.

Limited Support for Complex Design Decisions:

While the tool can provide feedback on standard design patterns and principles, it may not be equipped to evaluate more complex design decisions that involve multiple systems or cutting-edge technologies. Students working on more advanced or research-oriented projects may find that the tool's critique is insufficient for addressing specialized needs, leaving gaps in the support it provides.

Instructor Role and Customization:

The tool's feedback may not fully align with the instructor's teaching approach, course objectives, or personal preferences in grading and critique. Instructors may find that the tool's automated feedback does not adequately reflect the nuances of their expectations. The challenge, therefore, lies in offering instructors enough flexibility to tailor the feedback system and integrate it with their pedagogical goals.

C. User Adoption

Resistance to New Technology:

Some students and instructors may be resistant to adopting new educational technology. Students may feel overwhelmed or skeptical about the usefulness of an automated critique tool, especially if they are accustomed to traditional feedback methods from instructors. Instructors may also hesitate to rely on the tool, fearing that it will replace their own judgment or result in less personalized feedback. Overcoming this resistance requires clear communication about the tool's benefits and its role in complementing, rather than replacing, the learning process.

Technological Familiarity:

The tool may require a certain level of technical proficiency, both in terms of software usage and understanding of the underlying design principles. Students with limited experience in using such tools or unfamiliar with the particular design concepts the tool evaluates may find it difficult to use the tool effectively. To overcome this, onboarding materials, tutorials, and user support should be provided to ensure smooth adoption, particularly for those with varying levels of experience.

Student Motivation:

Some students may not be intrinsically motivated to engage with the feedback provided by the tool, particularly if they perceive it as a secondary form of feedback compared to instructor evaluation. If students view the tool as an additional, time-consuming task rather than an integral part of their learning process, they may not make full use of its features. Encouraging active participation through incentives, integration with course assessments, or providing personalized feedback on tool usage could help boost student motivation.

Instructor Buy-In and Training:

Successful adoption of the tool also depends on instructor engagement. Instructors may need training on how to use the tool effectively, incorporate it into their teaching strategy, and integrate it into their existing course structures. Without proper support and alignment with teaching goals,

 doi:10.20944/preprints202501.1130.v1

instructors may not fully embrace the tool, limiting its widespread adoption and impact on student learning.

In conclusion, while the educational software tool holds significant potential to improve student learning in software design courses, its success depends on addressing these technical, pedagogical, and adoption-related challenges. Continuous evaluation and refinement will be essential to overcome these limitations and ensure the tool meets the needs of both students and instructors.

6. Future Directions

A. Enhancements to the Tool

- Advanced Feedback Mechanisms: Future versions of the tool can incorporate more sophisticated feedback mechanisms, such as natural language processing (NLP) to provide more context-sensitive, detailed, and personalized critiques. This could include suggestions for improvement framed in clearer language, explanations of why a particular design approach may be beneficial or detrimental, and tailored recommendations for further study. Additionally, the system could integrate more advanced machine learning models to provide predictive analysis based on past trends, further improving the accuracy and usefulness of the feedback.
- Integration of Collaborative Features: Enhancements could include adding collaborative features, allowing students to work in teams and receive feedback on group designs. This would encourage peer learning and improve communication skills, as students would be able to critique each other's designs. Additionally, real-time collaboration features could be integrated to enable students to work on design projects together within the tool, making it easier to share ideas and engage in collective problem-solving.
- Support for Diverse Design Models: The tool could be extended to support a wider variety of design models, such as flowcharts, entity-relationship diagrams, and architectural diagrams. By accommodating various design paradigms, the tool would become more versatile and applicable to a broader range of software design tasks. Additionally, incorporating support for newer or emerging design frameworks (e.g., microservices architecture) would make the tool more future-proof.
- Gamification and Adaptive Learning: Introducing elements of gamification, such as design challenges, leaderboards, and rewards for improvement, could enhance student engagement and motivation. The tool could also adapt to the student's learning pace and proficiency, offering progressively more complex tasks and feedback as the student improves. This adaptive learning feature would ensure that students of all skill levels benefit from the tool in a way that is tailored to their individual progress.
- Integration with Version Control Systems: Future versions of the tool could integrate with version control systems (e.g., Git), allowing students to track the evolution of their designs over time and receive feedback on their design iterations. This integration would also help instructors monitor student progress and provide more targeted guidance.

B. Expanding to Other Areas

- Broader Application to Software Development Lifecycles: In the future, the tool could expand to cover more phases of the software development lifecycle, beyond just design. For example, it could include features for evaluating code quality, implementation correctness, and system testing. This would provide a comprehensive support system for students throughout their entire software development process, from initial design through to final deployment.
- Support for Different Programming Languages and Frameworks: Expanding the tool to support various programming languages and frameworks (e.g., Java, Python, C++, JavaScript, or emerging technologies like Flutter or Rust) would make the tool relevant for a broader set of students. Additionally, incorporating the evaluation of code structure and best practices for

 doi:10.20944/preprints202501.1130.v1

different languages would allow students to receive feedback that is both specific and relevant to the technologies they are using.

- Extension to Other Design Disciplines: While the current focus is on software design, the tool's core principles could be adapted for use in other design disciplines, such as web development, UX/UI design, or even hardware architecture. By expanding the tool's functionality to accommodate different types of design, it could support a wide range of courses related to design thinking and systems thinking across various disciplines.
- Artificial Intelligence and Algorithm Design: The tool could be extended to help students in fields such as artificial intelligence, machine learning, and algorithm design. By offering critiques on algorithmic efficiency, structure, and optimization, the tool would help students grasp key concepts related to designing and implementing algorithms. Feedback could be based on factors such as time complexity, space complexity, and correctness, thus supporting advanced coursework.

C. Collaboration with Educational Institutions

- Partnerships with Universities and Colleges: Establishing partnerships with universities and colleges would allow the tool to be integrated into software design curricula, making it a valuable asset for instructors and students. Collaboration could lead to a more comprehensive understanding of the tool's effectiveness in real-world teaching environments and provide an opportunity for ongoing feedback and development based on the needs of academic institutions.
- Faculty Training and Support: To ensure the tool's adoption and successful implementation, it would be essential to offer faculty training programs. These programs would help instructors understand how to incorporate the tool into their teaching methods, provide guidance on how to use its features effectively, and discuss best practices for leveraging the tool to enhance student learning. Instructors could also be provided with data and reports generated by the tool to guide their teaching and help identify common design weaknesses among students.
- Integration into Online Learning Platforms: Collaborating with online learning platforms (e.g., Coursera, Udemy, edX) could expand the tool's reach to students in remote or non-traditional educational settings. This would allow the tool to support students worldwide, promoting its use in a variety of course formats, such as self-paced, blended, or fully online courses. Integration with Learning Management Systems (LMS) like Moodle or Canvas would further streamline the adoption process.
- Research and Development Collaboration: Collaboration with research institutions or academic researchers in the field of software engineering education could foster continuous improvement and innovation in the tool. Research partnerships would allow for studies on the effectiveness of the tool in diverse educational settings, helping refine its features and better understand its impact on student learning outcomes. These collaborations could also lead to the development of new features based on cutting-edge research in software design education.

By exploring these future directions, the tool can evolve to become a more comprehensive, adaptive, and versatile platform that not only improves student learning in software design but also expands its applicability to broader educational contexts, supporting students and instructors in various disciplines and settings.

7. Conclusion

A. Summary of the Tool's Impact

The educational software tool for critiquing and supporting software design courses represents a significant advancement in how software design education can be delivered and experienced. By automating the critique process, the tool provides students with immediate, actionable feedback on their design work, helping them identify strengths and areas for improvement. This allows for more

 doi:10.20944/preprints202501.1130.v1

iterative learning, where students can refine their designs continuously, leading to a deeper understanding of design principles and better design outcomes.

Moreover, the tool helps bridge the gap between theoretical knowledge and practical application, providing an interactive platform where students can apply and test their skills in real-world contexts. It supports diverse learning styles and accommodates different levels of expertise, offering personalized feedback that encourages active learning and self-reflection. The system's potential to integrate with existing course structures, adapt to different software design methodologies, and provide scalable feedback for large numbers of students makes it an invaluable asset in modern educational environments.

Instructors, too, benefit from the tool's ability to automate time-consuming feedback tasks, allowing them to focus more on higher-level teaching activities, such as mentoring and personalized guidance. The tool can be easily integrated into various learning management systems, making it adaptable for a wide range of educational institutions. Additionally, by using the tool, instructors can gain insights into common student misconceptions, design weaknesses, and areas requiring more focus, further enhancing their teaching strategies.

B. Call to Action

As educational institutions continue to adapt to technological advancements, it is essential to embrace tools that not only improve the learning experience but also foster creativity, critical thinking, and problem-solving skills in students. We encourage educators, administrators, and software design curriculum developers to consider integrating this tool into their teaching frameworks, as it has the potential to transform how students engage with design concepts and improve their practical skills.

Furthermore, we urge developers to continue refining the tool, incorporating user feedback, and exploring new features that can expand its utility. Collaboration with educational institutions, faculty members, and students will be vital in ensuring the tool evolves to meet the diverse needs of both students and instructors, and to guarantee its continued relevance in a rapidly changing technological landscape.

Finally, as we continue to explore innovative approaches to software design education, we call for increased investment in tools that support the learning journey in meaningful ways. Whether through partnerships, further research, or direct engagement with educational communities, the future of software design education is bright, and this tool can play a key role in shaping that future.

References

1. Abdulkareem, S. M., Ali, N. M., Admodisastro, N., & Sultan, A. B. M. (2017). Class Diagram Critic: A design critic tool for UML class diagram. *Advanced Science Letters*, *23*(11), 11567–11571. https://doi.org/10.1166/asl.2017.10330
2. Ali, N. M., Admodisastro, N., & Abdulkareem, S. M. (2013). An educational software design critiquing tool to support software design course. *International Conference on Advanced Computer Science Applications and Technologies*, 31–36. https://doi.org/10.1109/acsat.2013.14
3. N. M. Ali, N. Admodisastro and S. M. Abdulkareem, "An Educational Software Design Critiquing Tool to Support Software Design Course," *2013 International Conference on Advanced Computer Science Applications and Technologies*, Kuching, Malaysia, 2013, pp. 31-36, doi: 10.1109/ACSAT.2013.14.
4. Ali, N. M., Admodisastro, N., & Abdulkareem, S. M. (2013, December). An educational software design critiquing tool to support software design course. In *2013 International Conference on Advanced Computer Science Applications and Technologies* (pp. 31-36). IEEE.
5. Abdulkareem, S. M., Ali, N. M., Admodisastro, N., & Sultan, A. B. M. (2017). Class Diagram Critic: A Design Critic Tool for UML Class Diagram. *Advanced Science Letters*, *23*(11), 11567-11571.

 doi:10.20944/preprints202501.1130.v1

6. ABDULKAREEM, S. M. (2015). CRITIC-BASED AND COLLABORATIVE APPROACH FOR UML CLASS DIAGRAM.
7. ABDULKAREEM, SORAN MAHMOOD. "CRITIC-BASED AND COLLABORATIVE APPROACH FOR UML CLASS DIAGRAM." (2015).