Jurnal Teknologi Informasi dan Pendidikan
Volume 19, No. 1, March 2026
https://doi.org/10.24036/jtip.v19i1.1091

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

Enhancing Asynchronous Code Review in Programming Education: A Systematic Literature Review of Visual-Contextual Feedback Systems

Muhammad Rizqi Jamhari[1*]✉, Agus Juhana[1]
[1]Multimedia Education, Universitas Pendidikan Indonesia, Bandung, Indonesia
**Corresponding Author:* rizqijamhari221@upi.edu

Article Information

Article history:

No. 1091
Rec. December 25, 2025
Rev. February 24, 2026
Acc. March 03, 2026
Pub. March 11, 2026
Page. 1323 – 1335

Keywords:

- *Systematic Literature Review*
- *PRISMA*
- *Code Review*
- *Cognitive Load*
- *Programming Education*

ABSTRACT

Asynchronous code review in programming education frequently suffers from disorganized, text-heavy feedback that increases learners' extraneous cognitive load and complicates error identification. Despite the growing adoption of visual and contextual annotation tools, the literature remains fragmented, and no systematic synthesis has established their comparative effectiveness. Following PRISMA 2020 guidelines, we searched five databases (Scopus, Web of Science, IEEE Xplore, ERIC, and ACM Digital Library) in September 2025, yielding 987 initial records. After rigorous screening, 35 peer-reviewed studies published between 2015 and 2025 were included to evaluate the impact of these tools on learner comprehension, workflow efficiency, and instructor workload. The synthesized empirical data indicate that situating visual feedback adjacent to code significantly reduces extraneous cognitive load measured via validated instruments like NASA-TLX and accelerates syntax error remediation (RQ1, RQ2). Furthermore, automated visual grading frameworks reduce instructor evaluation latency by 12% to 58% in large cohorts while maintaining a constant grading load, though initial configuration time and a lack of pedagogical depth for nuanced design choices remain primary constraints (RQ3). These findings suggest that educators should adopt hybrid feedback architectures, pairing automated visual triage with human oversight to maximize both operational efficiency and pedagogical depth.

How to Cite:

Jamhari, M. R., & Juhana, A. (2026). Enhancing Asynchronous Code Review in Programming Education: A Systematic Literature Review of Visual-Contextual Feedback Systems. Jurnal Teknologi Informasi Dan Pendidikan, 19(1), 1323-1335. https://doi.org/10.24036/jtip.v19i1.1091

Jurnal Teknologi Informasi dan Pendidikan
Volume 19, No. 1, March 2026
https://doi.org/10.24036/jtip.v19i1.1091

1. INTRODUCTION

Programming skills are learned in different ways. Learners' background, technology experience, and teaching style all play a role [1], [2]. Understanding these differences can improve teaching. This study discusses visual feedback for learners at all skill levels: beginner, intermediate, and advanced. Each group needs different methods. Visuals are usually more effective than text for explaining concepts [3], [4]. For example, clear visuals and web layouts help understanding [5], [6], while too much text can confuse and make it hard to see how code works [7].

Traditional feedback can cause confusion when not linked to specific code. Students then have to search for the right context [8], [9]. This means they must interpret feedback and find connections before making changes. This takes time that could be spent learning. Although previous studies have highlighted inefficiencies in students' review processes, recent evidence suggests that while tooling plays a role in review speed, the primary limiting factor is reviewers' willingness to dedicate time to examining others' code changes. Linked feedback helps reduce wasted effort. Studies show this is a common problem in beginner programming courses, especially during independent work [10], [11].

Recent advances in programming education tools have produced a proliferation of systems designed to minimize cognitive load through targeted information presentation. Features such as visual coding environments, live execution previews, and immediate feedback mechanisms have been shown to facilitate quicker identification of learning outcomes [3], [12], [13]. Applications that generate guided execution diagrams or link annotations directly to interface elements represent a particularly promising category, as they situate feedback within the learner's immediate working context [14], [15]. Nevertheless, these tools have largely been developed in isolation, and few systems meaningfully integrate insights from novice programming pedagogy, interactive development environments, and delayed feedback models [10], [16]. This fragmentation limits the transferability of findings and hinders educators seeking evidence-based tool selection. Examining this technological landscape through the intersecting lenses of educational psychology, cognitive science, and visual design is therefore essential for developing a comprehensive account of tool effectiveness. A multi-representational feedback strategy one that combines context-enriched visual representations with structured textual commentary has emerged as a promising approach. Recent interdisciplinary evidence suggests that such strategies yield higher learner engagement and comprehension compared to single-modality (visual-only or text-only) feedback [3], [17]. However, the extent to which these benefits persist across different learner profiles, programming task types, and class sizes remains insufficiently examined in the systematic literature, motivating the present review. Three primary issues are examined in the analysis:

- RQ 1: How do context-enriched visual representations impact learners' comprehension, efficiency, and cognitive load compared to textual feedback?

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

- RQ 2: What are the effects of visual feedback tools on students' workflow efficiency and error remediation, and do these tools enhance learning outcomes without compromising accuracy?
- RQ 3: How do visual feedback tools influence instructors' grading efficiency and workload, particularly in large class settings, and who benefits most from these tools in terms of time savings and accuracy?

2. RESEARCH METHOD

Application of the PRISMA guidelines facilitated the selection of terms, definition of eligibility criteria, and documentation of decisions during study selection. This approach enabled consistent evaluation of research across different fields.

2.1. Search and Selection Strategy

A systematic search was conducted in September 2025, adhering to the PRISMA 2020 reporting framework. Five electronic databases were queried: Scopus, Web of Science (WoS), IEEE Xplore, Education Resources Information Center (ERIC), and the ACM Digital Library. The final query string applied across databases was: ("visual feedback" OR "contextual annotation" OR "code annotation") AND ("code review" OR "programming assessment") AND ("asynchronous" OR "online learning") AND ("cognitive load" OR "comprehension" OR "grading efficiency"). Individual database adaptations were made to accommodate platform-specific syntax requirements. Searches were limited to publications from January 2015 through September 2025, written in English. In addition to database searching, a manual search was performed through (1) backward reference checking of all full-text included studies (snowball sampling), and (2) forward citation tracking of seminal papers using Google Scholar. The Publish or Perish (PoP) software was employed to retrieve and manage citation data during the initial identification phase (see Figure 1). Duplicate records were identified and removed using manual cross-checking by the primary reviewer. The complete screening and selection process is documented in the PRISMA 2020 flow diagram (Figure 2). Stage results:

Jurnal Teknologi Informasi dan Pendidikan
Volume 19, No. 1, March 2026
https://doi.org/10.24036/jtip.v19i1.1091

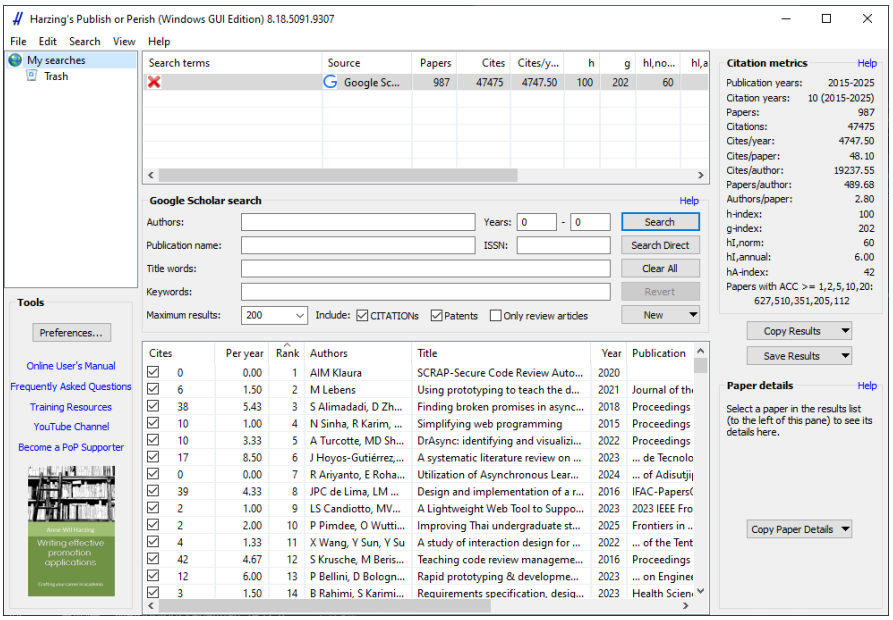

Figure 1. Software Publish or Perish

- Initial retrieval: 987 records
- After removing duplicates: 787 records
- After title screening: 152 records
- After abstract screening: 81 records
- After full-text screening: 35 records
- A total of 35 publications were included as the primary dataset before expansion via snowball sampling.

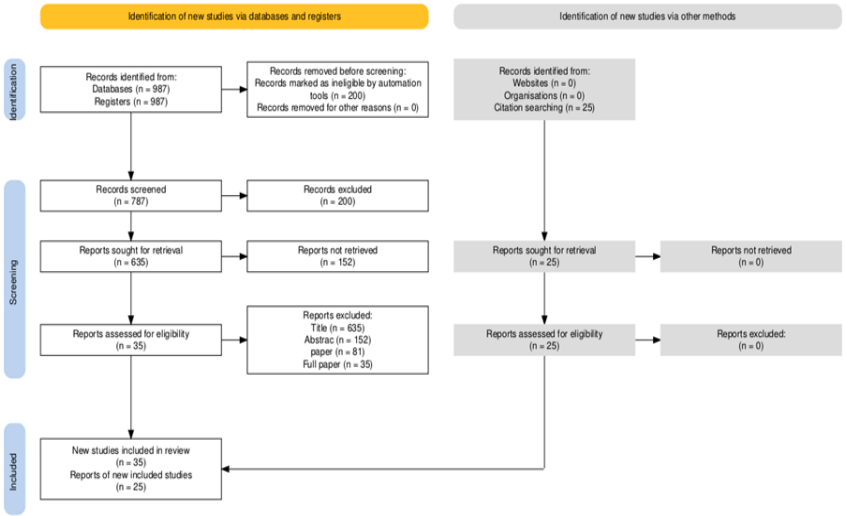

Figure 2. PRISMA Flow diagram

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

2.2. Inclusion and Exclusion Criteria

The study considered peer-reviewed papers, including conference proceedings and journal articles, published between 2015 and 2025 that investigated the automation of code review tasks using real datasets, with particular emphasis on empirical research related to teaching coding or design. Only peer-reviewed, full-text English-language studies were included, while books, book sections, review articles, critical perspectives, opinion pieces, and research lacking empirical evidence were excluded.

Table 1. Study Inclusion and Exclusion Criteria

Criteria	Inclusion Criteria (Studies Included)	Exclusion Criteria (Studies Excluded)
Time Range	Studies published between January 1, 2015, and December 31, 2025.	Studies outside this time range.
Publication Type	- Journal Articles (international, peer-reviewed) - Conference Proceedings (peer-reviewed)	- Books - Book Chapters - Literature Reviews (SLR, Systematic Review, Mapping Study) - Book Reviews
Language	Studies written in English.	Studies in languages other than English.
Availability	Full-text articles were accessible.	Non-full-text publications (e.g., abstract only, posters).
Topic & Content	(I1) Study discusses feedback systems or methods in programming/design education. (I2) Study presents an empirical evaluation, case study, or tested prototype.	(E3) Studies that are purely theoretical or conceptual without evaluation/implementation.

2.3. Data Extraction and Analysis

A comprehensive dataset was assembled by aggregating related studies using the Publish or Perish (PoP) approach. Key variables included task categorization, review automation techniques, evaluation metrics, and other relevant outcomes, following procedures established in systematic reviews of automated code review such as. To ensure findings addressed the collected data were mapped to the research questions as follows: participant characteristic variables facilitated analysis of differential learner responses to feedback types, supporting examination of cognitive load and comprehension outcomes (RQ1): classroom context data enabled investigation of pedagogical differences across educational settings relevant to workflow efficiency and error remediation (RQ2): and

feedback type variables and evaluation instruments provided evidence for assessing the influence of feedback on learning efficacy, aligning with the study of personalized code understandability scores as described in recent dynamic assessment methods. and cognitive engagement (directly informing RQ3 on instructor efficiency and workload). Synthesis of similar findings enabled identification of core patterns relevant to each research question, with data extraction structured purposefully, as instructors can use features such as prompt-pooling in systems like Autograder+ to guide feedback style through selected prompt templates.

Key actionable insights emerged from this analysis, indicating that integrating visual cues in feedback can significantly enhance students' immediate understanding and retention. For instructors, employing systems with real-time visual feedback not only reduces the cognitive load on students but also streamlines the grading process, particularly in large classes. Systems that categorize errors and offer visual guidance can aid students in faster error correction and foster a more engaging learning environment. Implementing such tools can lead to improved learning outcomes without diminishing grading accuracy.

2.4. Quality Assessment

Only peer-reviewed articles were included, while unfinished drafts and unapproved internal documents were excluded. Each selected study was evaluated for clarity of research questions, relevance of evidence, and validity of outcome correlations.

Table 2. Quality Checklist

No.	Quality Criterion	Response Options	Score
Q1	Are the research objectives clearly stated?	Yes / Partial / No	2 / 1 / 0
Q2	Is the research design explicitly described and appropriate for the RQ?	Yes / Partial / No	2 / 1 / 0
Q3	Are the visual feedback tools or interventions described in sufficient detail for replication?	Yes / Partial / No	2 / 1 / 0
Q4	Are the measurement instruments for cognitive load or comprehension identified?	Yes / Partial / No	2 / 1 / 0
Q5	Are outcome measures reported quantitatively?	Yes / Partial / No	2 / 1 / 0
Q6	Are inclusion/exclusion criteria for study participants described?	Yes / Partial / No	2 / 1 / 0
Q7	Are limitations of the study acknowledged by the authors?	Yes / No	1 / 0
Q8	Is the study published in a peer-reviewed venue (journal or conference proceedings)?	Yes / No	1 / 0

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

3. RESULTS AND DISCUSSION

3.1 Overview of Findings

A notable trend across the literature is the decline of traditional text-based coding tools in favor of systems that employ visual elements to enhance user engagement [11], [12], [13]. These platforms allow users to manipulate code logic visually and, in some cases, provide step-by-step execution views [3], [18]. Additionally, automated systems, including autograders and artificial intelligence, have become prevalent due to the demand for immediate feedback [19], [20], [21], [22]. However, deeper analysis reveals that while automation addresses the need for rapid responses, it introduces new pedagogical challenges. Specifically, automation is limited in its capacity to provide the nuanced, formative feedback that supports higher-order thinking and skill development. It tends to focus on surface features such as code correctness, often missing opportunities to foster critical reflection, problem-solving strategies, and creative coding practices. Although automation can offer efficient, objective grading, it may inadvertently reinforce a narrow conception of programming proficiency, emphasizing syntactic correctness over conceptual understanding or innovation. Furthermore, automated feedback sometimes lacks contextual sensitivity, failing to adapt to the learner's developmental stage or individual learning path. Researchers have highlighted instances in which reliance on automation led students to overlook underlying errors or persist in unproductive trial-and-error cycles rather than engaging in substantive code revision. This underscores the importance of balancing automated assessment with opportunities for human-mediated interpretation and mentorship. The interplay between visual methods, automation, and human judgment is a recurring theme that underpins the research questions. Recent work by [23]. demonstrates that presenting clustered code statements to maintain their semantic relationships, visualizing correctness variations through histograms, and enabling interactive navigation with semantic filters can enhance learners' ability to connect feedback with specific mistakes, thereby reducing ambiguity and confusion in asynchronous code review.

3.2 RQ1: Impact on Comprehension and Cognitive load

In the reviewed literature, comprehension is operationalized through specific performance indicators rather than subjective self-assessment. The primary comprehension indicators include syntax error reduction rates, post-test completion scores, and process comprehension metrics such as flow accuracy and logical completeness [2], [12], [24]. When supported by visual-contextual tools, learners demonstrate a significantly higher proficiency in mapping abstract text patterns to functional logic [25]. Furthermore, to evaluate cognitive load, the included studies avoid generic assumptions of cognitive strain and instead utilize validated measurement instruments. The majority of the literature

employs the NASA Task Load Index (NASA-TLX) to quantify perceived mental, physical, and temporal demand during programming tasks [26], [27] while other studies leverage objective neurophysiological measures, such as Electroencephalogram (EEG) recordings, to track frontal theta and parietal alpha power ratios [5]. These empirical measurements confirm that situating actionable, visual feedback directly adjacent to the erroneous code significantly reduces learners' extraneous cognitive load. Additionally, automated systems assist students by clustering recurring mistakes; for example, specific tools analyze student errors in SQL, classify the error topologies, and deploy targeted visual cues, thereby expediting the error correction process without overwhelming the learner's working memory [28].

3.3 RQ2: Student Remediation and the Influence of Interaction

Students' behavior is strengthened through error-correction exercises, exemplified by the use of an interactive feedback tool. These exercises provide students with brief, regular electronic nudges throughout their editing process. The quick, efficient feedback on the application allows students to break challenging tasks into smaller components and receive the corrective feedback needed to make multiple successful changes[20]. In contrast to activities with latency and traditional tasks, this approach reduces second-guessing to a level below uncertainty. Research shows that adaptive simulations outperform more traditional methodologies[29]. Self-directed activities give learners an opportunity to make mistakes, recognize them, and correct them. Step-by-step digital facilitators can enhance participants' competencies in organizing learning experiences, which may help keep learners on task [11], [29]. The outcomes observed by others can lead to careful reflection on their instructional approaches[6].

Repeated practice enhances comprehension. Systems that permit students to resubmit assignments and revise their work foster both skill development and confidence. In digital classrooms, students who receive immediate feedback persist with challenging tasks more consistently than those awaiting instructor comments [30].

3.4 RQ3: Instructor Workflow Efficiency and Constraints

Quantitative evidence across the reviewed literature demonstrates that integrating visual-contextual feedback and automated evaluation significantly mitigates instructor workload, shifting the pedagogical bottleneck from manual syntax-checking to higher-order assessment. Data from system implementations indicate a drastic reduction in grading response times; for instance, AI-assisted visual code review systems have been measured to reduce evaluation latency by 12% to 58% compared to traditional online judges, while optimizing API call costs for large-scale deployments [31]. Furthermore, automated grading frameworks utilizing differential semantic analysis can evaluate student submissions with an overall accuracy of 92.80% in mere seconds, effectively eliminating the need for

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

instructors to manually trace logic errors across hundreds of submissions[15].Beyond full automation, hybrid approaches such as visually-guided peer code review frameworks also yield measurable efficiency. Deploying strict visual rubrics allows instructors to maintain a constant grading load regardless of class size, adding less than a 10% workload increase for students while achieving a grading standard deviation identical to that of Teaching Assistants (TAs). Similarly, the use of automated classifiers to attach visual hints to common logical errors successfully captures 87% to 91% of student mistakes, saving instructors from repeatedly typing identical feedback [32]. Despite these compelling efficiencies, significant operational constraints persist. The initial configuration of these tools such as defining robust test suites, adapting grading rubrics, and mapping error topologies to visual hints demands substantial upfront instructor time. Long or overly complex visual rubrics have been shown to decrease peer-review accuracy. Moreover, while these systems excel at syntax and basic logic validation, they lack the pedagogical depth to evaluate architectural clarity or subjective design choices, necessitating continuous human oversight for comprehensive evaluation.

Study	Intervention Type	Key Quantitative Metric (Efficiency / Accuracy)	Constraints Identified
Tseng et al. (2025)	AI-Assisted Code Review	12% - 58% reduction in grading response time.	N/A (Study focused primarily on API cost optimization).
Lee (2024)	Semantic Autograder	92.80% overall accuracy in autonomous logic verification.	System struggles with complex nested loops, causing timeout issues.
Song et al. (2020)	Visual Rubric Peer-Review	Constant instructor load; <10% student workload increase.	High upfront setup time; long or overly detailed rubrics reduce accuracy.
Haldeman et al. (2021)	Automated Error Hinting	87% - 91% accuracy in capturing and hinting logical errors.	Requires significant historical data to train the error classifiers effectively.

4. CONCLUSION

This study synthesizes findings concerning the differences and interrelations between traditional written feedback and visual feedback for programming tasks, offering an integrative perspective on their respective impacts. Recent systematic reviews have summarized empirical research on technology-assisted feedback, including both written and visual modalities, within computer science education, highlighting developments in feedback systems since 2015[6]. While previous research has investigated automated feedback that emulates human-written commentary, particularly via learnersourcing methods [28], more direct empirical comparisons are required to determine how visual and textual feedback distinctly contribute to learner understanding. The synthesis of evidence demonstrates that visual cues facilitate swifter concept acquisition and alleviate cognitive load for students, while prompt forms of feedback reduce time spent on corrections and

subsequently boost both learning efficiency and learner confidence [20], [33]. Integrating these insights, the review underscores that student cognition, instructional practices, and institutional resources are deeply interconnected and must be considered collectively to achieve meaningful learning outcomes. Practical implementation, therefore, calls for a holistic adoption of visual feedback technologies that not only quicken grading and comprehension [21], [34], but simultaneously supply instructors with robust, actionable data to guide real-time pedagogical adjustments. The synergy between immediate, user-friendly visual feedback tools and traditional strategies enables the development of dynamic classroom environments, promoting iterative improvement among both learners and educators. Nevertheless, the ultimate efficacy of these technologies depends on their usability and reliability, as tools that are cumbersome or inaccurate fail to yield sustained benefits in educational contexts [35]. In summary, an integrated approach leveraging visual feedback and immediate response systems maximizes efficiencies for instructors and students alike, supports deeper conceptual understanding, and enhances the overall effectiveness of programming education.

The scope of this review is constrained by several notable limitations. First, the selection of studies was restricted by the specific search parameters and inclusion criteria, which may have led to the exclusion of relevant research published under alternative terminology or in emerging venues not indexed by the chosen databases. As a result, studies employing innovative or interdisciplinary approaches outside traditional programming education literature may not have been captured. Second, only sources published in English were included, potentially excluding valuable perspectives from non-English literature. Third, the exclusion of informal and grey literature, such as blog posts, internal reports, dissertations, and unpublished studies, while increasing the reliability of findings, may also have limited the diversity of practical insights represented. Additionally, publication and selection biases may have influenced the composition of the reviewed evidence. Limitations in access to full texts and reliance on available metadata occasionally hindered comprehensive evaluation of all potentially relevant studies. These constraints restrict the generalizability of the findings and suggest that caution should be exercised when applying the recommendations beyond the reviewed literature. As a result, interpretations and practical applications derived from this review should account for these boundaries, particularly considering that some innovative or context-specific insights may be absent. Future collaboration between researchers and educators remains essential. Instructors are encouraged to engage in research initiatives by piloting innovative tools and sharing classroom data, as such contributions can yield practical insights and support the refinement of feedback mechanisms to suit evolving educational contexts.

Future research should prioritize the development of robust, validated methods for assessing cognitive load in learners, as well as rigorous experimental designs for determining the specific pedagogical and contextual factors that influence the effectiveness of visual feedback. In addition to refining these measurement techniques, further

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

investigation is necessary to identify how individual differences among learners, course contexts, and the nature of programming tasks interact with visual feedback mechanisms. Longitudinal studies are especially needed to examine whether the observed benefits of visual feedback on immediate task performance persist over time and contribute to deeper, sustained learning outcomes beyond initial task completion. Moreover, comparative research across instructional modalities and feedback system designs would clarify best practices and inform scalable implementation strategies for asynchronous code review environments.

REFERENCES

[1] M.-C. Wu, M.-C. Hung, M.-L. Chiu, C.-H. Chuang, and Y.-B. Lin, "Impact of Visual Cues and Feedback on Learning Performance: Different Learning Styles and Cognitive Loads," *I Int. Res. J. Eng. Sci.*, vol. 8, no. 5, pp. 1–6, 2022, doi: 10.14303/2315-5663.2022.81.

[2] C. H. Lai, P. W. Lin, and S. H. You, "Development and evaluation of a dynamic code visualization system for C programming education: The PVLS approach," *Soc. Sci. Humanit. Open*, vol. 12, no. July, p. 101962, 2025, doi: 10.1016/j.ssaho.2025.101962.

[3] M. Lu and Z. Hu, "Leveraging Multimodal Information for Web Front-End Development Instruction: Analyzing Effects on Cognitive Behavior, Interaction, and Persistent Learning," *Inf.*, vol. 16, no. 9, Sep. 2025, doi: 10.3390/info16090734.

[4] M. B. Garcia and A. M. F. Yousef, "Cognitive and affective effects of teachers' annotations and talking heads on asynchronous video lectures in a web development course," *Res. Pract. Technol. Enhanc. Learn.*, vol. 18, no. January, 2023, doi: 10.58459/rptel.2023.18020.

[5] X. Du, M. Dai, H. Tang, J. L. Hung, H. Li, and J. Zheng, "A multimodal analysis of college students' collaborative problem solving in virtual experimentation activities: a perspective of cognitive load," *J. Comput. High. Educ.*, vol. 35, no. 2, pp. 272–295, 2023, doi: 10.1007/s12528-022-09311-8.

[6] H. Gabbay and A. Cohen, "Investigating the effect of automated feedback on learning behavior in MOOCs for programming," *Proc. 15th Int. Conf. Educ. Data Mining, EDM 2022*, 2022, doi: 10.5281/zenodo.6853125.

[7] E. L. Glassman, J. Scott, R. Singh, P. J. Guo, and R. C. Miller, "OverCode: Visualizing variation in student solutions to programming problems at scale," *ACM Trans. Comput. Interact.*, vol. 22, no. 2, Mar. 2015, doi: 10.1145/2699751.

[8] M. Heintz and E. L. C. Law, "PDOtcapturer and PDOTanalyser: Participatory design online tools for asynchronous idea capturing and analysis," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Sep. 2018, pp. 622–633. doi: 10.1145/3240167.3240214.

[9] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at Microsoft," in *IEEE International Working Conference on Mining Software Repositories*, IEEE Computer Society, Aug. 2015, pp. 146–156. doi: 10.1109/MSR.2015.21.

[10] L. De Oliveira Brandão, Y. Bosse, and M. A. Gerosa, "Visual programming and automatic evaluation of exercises: An experience with a STEM course," *Proc. - Front. Educ. Conf. FIE*, vol. 2016-Novem, 2016, doi: 10.1109/FIE.2016.7757621.

[11] F. Y. Chung, "Programming Learning Platform with Visual Aids," *J. Int. Conf. Proc.*, vol. 8, no.

1, pp. 378–388, 2025, doi: 10.32535/jicp.v8i1.4002.

[12] K. Kuosa *et al.*, "Interactive visualization tools to improve learning and teaching in online learning environments," *Int. J. Distance Educ. Technol.*, vol. 14, no. 1, pp. 1–21, 2016, doi: 10.4018/IJDET.2016010101.

[13] J. H. Moon, S. Nyu, F. Q. China, S. S. Tian, and J. Hargis, "Teaching and Learning Creative Coding With Conversational AI," *Glob. Lokal Distance Educ. GLOKALde*, vol. 10, no. August, p. 29, 2024.

[14] C. H. Tseng, H. C. K. Lin, A. C. W. Huang, and J. R. Lin, "Investigating the effects of PuppyCodeReview, an AI-based code review system, on students' cognitive load," *SoftwareX*, vol. 31, no. July, p. 102283, 2025, doi: 10.1016/j.softx.2025.102283.

[15] X. Liu, S. Wang, P. Wang, and D. Wu, "Automatic grading of programming assignments: An approach based on formal semantics," *Proc. - 2019 IEEE/ACM 41st Int. Conf. Softw. Eng. Softw. Eng. Educ. Training, ICSE-SEET 2019*, pp. 126–137, 2019, doi: 10.1109/ICSE-SEET.2019.00022.

[16] X. Song, S. C. Goldstein, and M. Sakr, "Using Peer Code Review as an Educational Tool," *Annu. Conf. Innov. Technol. Comput. Sci. Educ. ITiCSE*, vol. 2019, pp. 173–179, 2020, doi: 10.1145/3341525.3387370.

[17] R. A. Rasheed, A. Kamsin, and N. A. Abdullah, "An approach for scaffolding students peer-learning self-regulation strategy in the online component of blended learning," *IEEE Access*, vol. 9, pp. 30721–30738, 2021, doi: 10.1109/ACCESS.2021.3059916.

[18] M. Grossman, M. Aziz, H. Chi, A. Tibrewal, S. Imam, and V. Sarkar, "Pedagogy and tools for teaching parallel computing at the sophomore undergraduate level," *J. Parallel Distrib. Comput.*, vol. 105, pp. 18–30, 2017, doi: 10.1016/j.jpdc.2016.12.026.

[19] S. Palahan, "PythonPal: Enhancing Online Programming Education Through Chatbot-Driven Personalized Feedback," *IEEE Trans. Learn. Technol.*, vol. 18, pp. 335–350, 2025, doi: 10.1109/TLT.2025.3545084.

[20] F. H. Wang, "Efficient generation of text feedback in object-oriented programming education using cached performer revision," *Mach. Learn. with Appl.*, vol. 13, no. June, p. 100481, 2023, doi: 10.1016/j.mlwa.2023.100481.

[21] M. Goodwin and T. Drange, "Teaching programming to large student groups through test driven development comparing established methods with teaching based on test driven development," *CSEDU 2016 - Proc. 8th Int. Conf. Comput. Support. Educ.*, vol. 1, no. 281, pp. 281–288, 2016, doi: 10.5220/0005789502810288.

[22] H. M. Qadir, R. A. Khan, M. Rasool, M. Sohaib, M. A. Shah, and M. J. Hasan, "An adaptive feedback system for the improvement of learners," *Sci. Rep.*, vol. 15, no. 1, pp. 1–13, 2025, doi: 10.1038/s41598-025-01429-w.

[23] C. H. Ho, H. Q. Zhang, J. Li, and M. Q. Zhang, "Development and Application of Interactive Teaching Systems for Online Design Courses," *Int. J. Distance Educ. Technol.*, vol. 21, no. 2, pp. 1–28, 2023, doi: 10.4018/ijdet.317365.

[24] T. Faas, L. Dombrowski, A. Young, and A. D. Miller, "Watch me code: Programming mentorship communities on Twitch.tv," *Proc. ACM Human-Computer Interact.*, vol. 2, no. CSCW, Nov. 2018, doi: 10.1145/3274319.

[25] A. Alam, I. Astuti, and D. Suratman, "Development of Web Programming Interactive Learning Multimedia in Vocational Middle School," *JTP - J. Teknol. Pendidik.*, vol. 24, no. 1, pp. 50–62, 2022, doi: 10.21009/jtp.v24i1.24242.

P.ISSN: 2086 – 4981
E.ISSN: 2620 – 6390
tip.ppj.unp.ac.id

[26] M. Khalid, A. Akanmu, A. Afolabi, H. Murzi, I. Awolusi, and P. Agee, "Design And Usability Evaluation Of An End-User Programming Environment For Equipping Construction Students With Sensor Data Analytics Skills," *J. Inf. Technol. Constr.*, vol. 30, pp. 213–242, 2025, doi: 10.36680/j.itcon.2025.010.

[27] Ş. H. Aksu, A. Adem, E. Çakıt, M. Dağdeviren, and W. Karwowski, "An examination of the interrelationships among NASA-TLX dimensions utilizing the DEMATEL method," *PLoS One*, vol. 20, no. 4 April, pp. 1–19, 2025, doi: 10.1371/journal.pone.0320638.

[28] C. Douce, "Teaching web technologies: understanding the tutor's perspective," *Open Learn.*, vol. 34, no. 1, pp. 78–88, Jan. 2019, doi: 10.1080/02680513.2018.1483226.

[29] M. Rogti, "The Effect of Mobile-based Interactive Multimedia on Thinking Engagement and Cooperation," *Int. J. Instr.*, vol. 17, no. 1, pp. 673–696, Jan. 2024, doi: 10.29333/iji.2024.17135a.

[30] L. Nagel, O. Karras, S. M. Amiri, and K. Schneider, "Turning asynchronicity into an opportunity: asynchronous communication for shared understanding with vision videos," *Requir. Eng.*, vol. 29, no. 1, pp. 49–71, Mar. 2024, doi: 10.1007/s00766-024-00414-5.

[31] L. Dong-Kyu, "A GPT-based Code Review System for Programming Language Learning," 2024, [Online]. Available: http://arxiv.org/abs/2407.04722

[32] G. Haldeman, M. Babeş-Vroman, A. Tjang, and T. D. Nguyen, "CSF: Formative Feedback in Autograding," *ACM Trans. Comput. Educ.*, vol. 21, no. 3, pp. 1–30, 2021, doi: 10.1145/3445983.

[33] S. Gunawardena, E. Tempero, and K. Blincoe, "Concerns identified in code review: A fine-grained, faceted classification," *Inf. Softw. Technol.*, vol. 153, no. August 2022, p. 107054, 2023, doi: 10.1016/j.infsof.2022.107054.

[34] M. Hull, C. Guerin, J. Chen, S. Routray, and D. H. Chau, "Towards Automatic Grading of D3.js Visualizations," Oct. 2021, [Online]. Available: http://arxiv.org/abs/2110.11227

[35] C. Garcia and N. Lemos, "International Journal On Informatics Visualization The Gamification of E-learning Environments for Learning Programming," vol. 7, no. June, pp. 455–462, 2023, [Online]. Available: www.joiv.org/index.php/joiv