Toward True Differentiated Instruction with PathMX and Curriculum as Code

Mark Johnson[1] and Chris Alvin[1]
Department of Computer Science[1]
Furman University
Greenville, SC, 29613
{mark@fellowhumans.com,chris.alvin@furman.edu}

Abstract

Students enter CS1 courses with widely varying programming experience, mathematical maturity, and learning preferences. We propose a framework called PathMX that addresses this challenge through a "curriculum as code" methodology that treats curriculum materials as structured, version-controlled repositories editable by both instructors and LLM-based coding agents. When curriculum is authored as code following clear conventions, agents can generate personalized content on demand, enabling scalable differentiated instruction that was previously impractical. Our CS1 case study demonstrates that this approach makes true differentiated instruction achievable within the constraints of real classroom teaching.

1 Introduction

CS1 instructors routinely face cohorts where students arrive with vastly different readiness levels. The diversity arises from differences in prior programming experience, mathematical maturity, formal reasoning abilities, and learning preferences. Some students demonstrate their mastery of Python syntax because they have been coding since middle school. Others are confounded by

the first `for` loop. Some grasp algorithmic thinking immediately, while others require multiple explanations and/or varied examples before achieving understanding. This is not a flaw in our students. It is the reality of differences in cognitive abilities, prior experiences, and educational backgrounds.

An effective response to this diversity is differentiated instruction, where each student receives a truly tailored learning experience instead of a one-size-fits-all model. As Tomlinson describes in her foundational work, differentiated instruction is "a philosophy of teaching rooted in deep respect for students, acknowledgment of their differences, and the drive to help all students thrive" [17]. When done well, differentiation promotes inclusivity and equity by (1) addressing diversity in cognitive abilities, prior experiences, and learning profiles, (2) enhancing engagement and motivation by connecting to student interests and backgrounds, and (3) improving academic outcomes through appropriately scaffolded instruction [16]. These goals align with Universal Design for Learning (UDL) principles of providing multiple means of representation, engagement, and expression [1].

However, true differentiated instruction is nearly impossible to deliver at scale. For a typical CS1 class, an instructor implementing differentiated instruction might need to (among other tasks): (a) Create personalized learning guides for students struggling with functions; (b) Generate project variants that reflect diverse student interests, backgrounds, experiences, and identities; (c) Provide detailed, individualized feedback on every coding assignment; and (d) Adapt examples to match different backgrounds and interests. This represents only a fraction of what genuine differentiation demands. The time investment required for true differentiation is unsustainable for any educator.

Thus, we make compromises. We create one set of assignments and *hope* they work for most students. We provide individual feedback when possible, but the time required limits how detailed or frequent it can be. We use the same examples semester after semester. We know some students are not getting what they need, but manually creating differentiated materials for each student is unrealistic given time constraints and the limits of human creativity.

PathMX (Path Markdown eXtension) addresses this challenge. PathMX is a "curriculum as code" methodology that treats curriculum materials as structured, version-controlled repositories editable by both instructors and LLM-based coding agents. Existing platforms optimize for human authoring or interactive delivery; none provide conventions specifically designed for agent-based curriculum generation. When curriculum is authored as code following clear conventions, agents can generate personalized content on demand. This includes creating individualized student guides in minutes, generating project variants tailored to different interests, drafting detailed code feedback for instructor review, and adapting materials to specific learning needs. As depicted

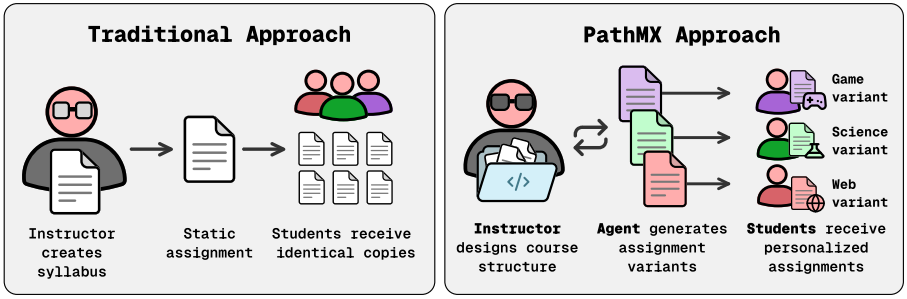

Figure 1: Traditional approaches deliver identical materials to all students; PathMX generates personalized variants through structured repositories and agent-based content generation.

in Figure 1, the instructor stays in control of pedagogical decisions while agents handle the time-consuming work of content generation and personalization.

We demonstrate this approach through a real classroom deployment: a CS1 course at Furman University in Fall 2025. Using PathMX, we created personalized learning guides for individual students, generated themed project variants, provided detailed code review feedback, and rapidly iterated on curriculum improvements, all while maintaining version control and a unified curriculum repository. Tasks that would have required many hours of manual authoring and customization were often completed in minutes with agent assistance.

Our contributions described in this paper include:

- The PathMX methodology: markdown conventions, metadata structures, and file organization that enable agent-based curriculum authoring.
- Practical workflows for creating personalized and differentiated curriculum at scale, demonstrated through our CS1 case study.
- Lessons learned from real classroom deployment, including both benefits realized and challenges encountered.

Collectively, this work establishes agent-based curriculum authoring as a viable path toward scalable differentiated instruction.

2 Related Work

Several platforms enable instructors to create interactive computing textbooks. Runestone Interactive [12] pioneered the open-source interactive textbook model, supporting embedded code execution, visualizations, and auto-graded

exercises. The platform [4] now serves thousands of daily learners across many textbooks authored in reStructuredText. OpenDSA [15] introduced a modular, community-driven approach for algorithm visualization textbooks, where instructors can customize content selections for their courses. LiaScript [2] represents the closest technical analog to PathMX, extending standard Markdown with interactive elements and using a browser-based compiler that requires no server infrastructure.

The application of software engineering practices to curriculum management has gained traction in the past few years. Rodriguez et al. [13] defined "courseware as code" principles at the U.S. Army Cyber School, where all educational content is stored in machine-readable markup in GitLab repositories. RepoBee [9] provides a command-line interface for managing student Git repositories at scale, enabling batch repository generation and automated assignment distribution. Hsing and Gennarelli [6] found that GitHub use in CS courses predicted better learning outcomes in a large-scale study of students.

Recent work has explored using large language models for educational content creation in computing courses. Sarsa et al. [14] demonstrated that LLMs could generate programming exercises with problem descriptions and solutions. However, only approximately 31% of generated exercises had sample solutions that passed their accompanying automated tests. This was primarily due to issues with test quality rather than solution correctness. MacNeil et al. [11] deployed GPT-generated code explanations in a web development textbook, finding them more helpful and accurate than student-created explanations. Leinonen et al. [10] confirmed that LLM explanations were rated as more accurate and easier to understand than those created by students.

Adaptive learning systems adjust content difficulty during student interaction. Ericson et al. [3] studied adaptive Parsons problems that dynamically adjust difficulty based on student performance, both within individual problems and between successive problems. Their study found that students solving adaptive Parsons problems completed practice exercises more efficiently than those writing equivalent code, while achieving similar learning gains. Fonseca et al. [5] explored differentiated instruction in a PHP programming course using ongoing progress monitoring to adjust content based on student performance. While these systems successfully personalize learning experiences, they require pre-authored content pools and adapt only within existing materials, leaving the challenge of creating differentiated content in the first place.

PathMX synthesizes elements from these traditions while enabling instructors to generate differentiated curriculum at scale. Existing platforms excel at delivering interactive content to students or supporting human collaboration on curriculum, but lack the structured conventions needed for agent-based editing. LLM tools can generate individual exercises or explanations, but cannot

maintain consistency across an entire curriculum repository. Adaptive systems personalize content during student interaction, but require pre-authored content pools and offer no solution to the authoring bottleneck. Similarly, UDL [1] advocates for curriculum designed from the beginning with multiple means of representation, engagement, and expression. However, implementing these principles at scale faces the same authoring bottleneck as differentiated instruction. Even in computing education contexts where UDL has been successfully applied [7], creating multiple pathways requires substantial upfront investment that grows with class size and content diversity.

PathMX addresses these limitations by treating curriculum as version-controlled repositories with explicit file organization and metadata conventions. These conventions make curriculum both human-editable and agent-processable. This enables LLM agents to generate differentiated content while instructors maintain control through templates, style guides, and review workflows. Rather than adapting content during student interaction, PathMX generates differentiated variations in advance, enabling instructor review and preserving agency over pedagogical decisions.

3 PathMX: Curriculum as Code

PathMX (Path Markdown eXtension) is a set of conventions for authoring human and agent-friendly curriculum repositories. Hyperlinks between documents create paths through the curriculum that can be followed by humans and agents alike. The three primary principles of PathMX are depicted in Figure 2 and are described as follows:

1. Curriculum is represented as Markdown files with minimal metadata in a single version-controlled repository
2. Markdown files are "type hinted" to indicate their role within the curriculum domain
3. Hyperlinks are used to link related files, create learning sequences, and provide additional context for agent and human authors

Storing curriculum in a source-controlled repository provides automatic version control, history, and branching. These are essential capabilities for any collaborative authoring process, including collaboration with agents. Agents and humans make mistakes, and version control is critical for inspecting, comparing, and recovering from those mistakes.

Markdown serves as the foundation of PathMX for several reasons. It is both human-readable and machine-parseable, is the AI-native *lingua franca* for documentation and LLM-based tooling, and it is widely supported and easily

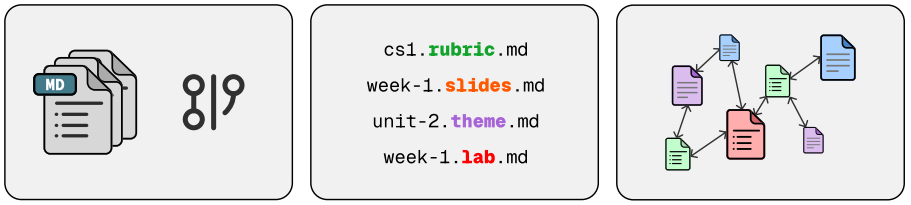

Figure 2: PathMX principles: version-controlled Markdown with type hints and hyperlinked knowledge graphs.

transformed into other formats. Modern coding models (e.g., GitHub Copilot, Gemini Code, etc.) are trained extensively on Markdown and are adept at reading, writing, and editing it without additional training or customization. In addition, many popular document authoring tools natively support Markdown, making it a natural choice for curriculum authors.

The repository also acts as an automatic context boundary (or "harness") for agents. Modern agents use standard file-based tools (e.g., `ls`, `find`, `grep`) to navigate and search within this boundary. Many agent systems allow repository-specific instructions to be provided via a file such as `AGENTS.md`, and expose tools for code exploration and editing. In addition, many modern agents maintain lightweight embeddings of repository contents, effectively creating a semantic index for efficient search and natural language retrieval.

Another important feature of Markdown is its support of hyperlinks. Agents can follow these links to navigate the curriculum and to understand the relationships between different parts of the repository. This is a powerful capability for bringing in additional context simply by following links. On the tooling side, as shown in Figure 3, this allows for a knowledge graph to emerge naturally from the curriculum content as it is built.

While simply adding and linking Markdown files in a repository provides benefits out-of-the-box, we found that applying domain-specific "type hints" produced additional inference gains. Because agent-based tools often browse a repository in directory-tree form, we add sub-extensions to markdown files by encoding their role in the filename. For example:

- `cs1.outcomes.md` indicates a file containing course learning outcomes
- `lecture-1.slides.md` indicates lecture slides
- `week-4.lab.md` indicates a lab assignment

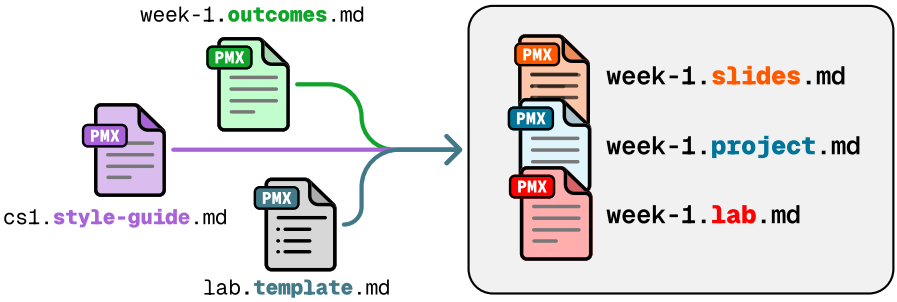

Figure 3: An example markdown file graph with links, type hints, and frontmatter illustrating automatic agent inference.

- `project.template.md` indicates a project template
- `week-4.project.spec.md` infers a project specification using the project template

See the companion repository[1] for an example specification file and the resulting output.

We found that agents were often able to infer the necessary context and next actions from file listings alone. This lightweight naming convention provides structure for agent inference while remaining flexible enough to accommodate new material types as the curriculum evolves. In some cases, additional metadata is necessary to guide agent inference and build tools. We use YAML frontmatter as a simple mechanism to attach metadata to Markdown files. This metadata can include learning outcomes, difficulty levels, prerequisites, tags, and other information that supports deterministic processing (e.g., build pipelines) and inference-based behaviors (e.g., personalized recommendations).

Hyperlinks are used to link in the necessary context for both agents and students. Agents will naturally follow links when prompted to perform a task. For example, linking a rubric to a project (e.g., `project-1.project.md` → `project-1.rubric.md`) allows the agent to automatically evaluate work against the rubric without additional instruction.

With these conventions in place, an instructor can scaffold a new course by creating a folder with a small set of Markdown files and high-level instructions to the agent (e.g., "Generate a lab assignment that introduces loops using examples from the cohort's interests"). Instructors may reuse existing materials as templates or author new materials from scratch. In our deployment,

[1] `https://github.com/pathmx/cs1-example`

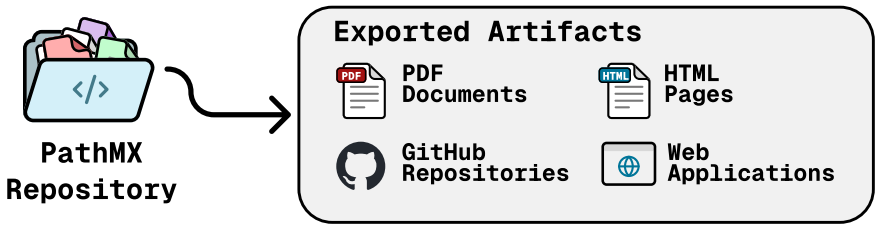

Figure 4: Sample export pipeline configuration used in our deployment.

agents performed best when expectations and standards were clearly specified in this initial scaffolding. Style guides, example project specifications, outcome descriptions, and type-hinted ontologies provided a robust starting point for agent-assisted authoring in the CS1 context.

Once the initial scaffolding exists, agents can generate a variety of curriculum artifacts, including personalized learning guides, project variants, code review comments, and other course materials. Instructors then review and refine these artifacts using standard code-review tools and workflows (e.g., pull requests, inline comments).

When materials are ready for student use, the same repository serves as the source of truth for export into multiple delivery formats, including PDF, HTML, GitHub repositories, and web applications (see Figure 4). These export pipelines are optional and can be adapted to the constraints of a particular course or institutional environment. Because Markdown is a widely supported source format, PathMX repositories can be integrated with existing tools and learning management systems with relatively little additional infrastructure.

4 CS1 Case Study

PathMX was developed in practice during a CS1 course teaching introductory Python programming to a cohort of 18 students. The class included a wide range of prior programming experience, and most were not computer science majors. The goal was to introduce fundamental programming concepts through project-based learning that mirrored real-world practices. Because of the diverse interests and backgrounds, the course was designed to be highly personalized and flexible. The instructor defined course outcomes at a high level and then tailored them to students' interests and level using the PathMX methodology described in Section 3.

Throughout the course, when students showed interest in particular tools or instructors identified gaps in understanding, we created projects and tutorials

to support their learning. We recorded student preferences in anonymized "persona" files (e.g., `student-<anon_id>.persona.md`) to avoid sharing personally identifiable information with the agent. This allows for robust personalization while complying with FERPA and other privacy regulations.

The PathMX methodology supported the lightweight collection of qualitative and quantitative information about students' aptitudes, interests, and learning goals. This information could be pasted or dictated into the agent tool, then automatically placed in the appropriate persona and cohort files. Information about the student's working environment, such as operating system, IDE, and prior programming experience, was also collected and used to generate more aligned and effective materials. Much of it was collected during lecture ice-breakers (for example, favorite artist, movie, or book), with more detailed information about goals and learning gaps gathered during office hours and direct interactions with the instructor. This kind of rapid transfer of qualitative data to a structured file would have been less tenable with traditional authoring tools.

Once enough information had been collected into personas, agents could perform cohort analysis by querying the repository. A taste profile for the cohort was created based on students' expressed interests and used to generate themed labs and practice exercises. Based on instructor observation, students showed higher interest and engagement with the themed curriculum than in prior sections of the same course that used more traditional static material. As an example, one student with prior Python experience worked on more advanced projects while keeping the same general theme and goals as the rest of the class. This allowed the student to be challenged at an appropriate level while still participating in shared discussions and activities.

Templates were used to scaffold new labs based on the cohort's taste profile and the course's topical flow. We wrote brief lab specification files (e.g., `lab-<number>.spec.md`) that described each lab's learning objectives, requirements, and constraints. The agent then used the template, the theme, and the taste profile to generate lab materials. We instructed agents to avoid letting thematic elements overshadow core learning objectives. For the CS1 course, the instructor used Deckset, a Markdown-based presentation tool, so lab materials, guides, and quizzes could be created directly from class lectures.

Mid-to-late term projects were presented in a "choose your own adventure" style (reflecting UDL's multiple means of engagement), where students selected a focus area and completed the project from that perspective. For example, a data science project was offered with Web, Game, or Scientific Computing focus areas. The same datasets were shared across tracks and the students could complete one or more based on their interests. Table 1 shows that students appreciated being able to align projects with their own interests and goals, with

Table 1: Student perceptions of CS1 using PathMX ($n = 17$).

Prior Experience	None: 41%, <1 yr: 47%, 1–3 yrs: 12%
Key Measures (1–5 scale, mean ± SD)	
Materials relevant to interests	4.00 ± 0.79
Project choice increased motivation	4.47 ± 0.72
Themed examples increased engagement	3.82 ± 0.88
Feedback specific and helpful	4.35 ± 1.00
Guides helped when struggling	4.24 ± 0.83

mean ratings of 4.0–4.5 (on a 5-point scale) for materials relevance, project choice motivation, feedback quality, and guide helpfulness.

Differentiated instruction also raises challenges with feedback. To provide high-quality and fair feedback across curriculum variants, we created feedback harnesses (e.g., `<student_id-project_id>.feedback.harness.md`). These harnesses were generated from students' code pulled into the repository via the GitHub CLI and converted to Markdown using a pre-established template. Each harness included commit history, code snapshots, project specification, a rubric, and agent-specific instructions. Agents generated accurate, specific feedback from these harnesses, and the same artifacts were helpful in human review as a record of student progress. In some cases, feedback was delivered as GitHub Issues with checklists and line-specific comments for each student.

The instructor remained central in this workflow and actively reviewed and iterated on the curriculum and feedback alongside agents. Agentic tools supported side-by-side collaboration and naturally created checkpoints that could be reviewed or rolled back as needed. Agent-based IDEs are continuing to improve these workflows and make a strong copilot for curriculum authoring. The version-controlled repository provided a history of the curriculum's evolution and captured snapshots of student work for later reference and analysis.

Overall, the PathMX methodology enabled rapid iteration and mid-semester improvements. Student feedback was incorporated into the repository, and agents generated new versions of the curriculum materials in response. This allowed the curriculum to be updated in real time and for students to receive timely feedback on their work. We found that this instructor-mediated on-demand generation was preferable to real-time student-facing agents that lacked editorial oversight or sufficient course-specific context.

5 Student Impact and Educational Integration

PathMX enabled just-in-time instruction and the deployment of curriculum materials to where students were already engaged. Students could access the curriculum in a variety of formats, including GitHub repositories, PDFs, and web applications. A custom integration with GitHub Classroom enabled bi-directional syncing of feedback-based curriculum materials with students' work. The custom integration itself was built with agentic assistance in the same repository as the curriculum materials.

Because the content was presented in the most appropriate medium for the students, a centralized Learning Management System (LMS) was not needed or used. Students were able to access the curriculum and feedback in the context of their own work. This allowed for much more direct and personalized instruction and feedback as the class progressed. In some cases, students would request additional materials for concepts they were struggling with. We generated personalized tutorials and guides for these concepts using the PathMX methodology, which helped them catch up with the class and get back on track. The line-by-line feedback was constructive for students to learn how to refactor and improve their code.

Several students expressed interest in more creative applications of coding, such as gaming or building out web interfaces. We were able to create project variants for these interests, and students were able to choose from a variety of paths to follow. The culminating project was the final project in which students defined their own project requirements with guidance from the instructor. These student-authored specifications were transformed into unique guides for each student. As the project progressed, snapshots of students' work were created at key checkpoints. Guides were then automatically updated to reflect progress, changes, and next steps for the project based on the students' current committed work.

Overall, students were able to engage with the curriculum in a way that was more aligned to their own interests and goals and calibrated to the proximal edge of their learning curve. They received direct, detailed, and actionable feedback throughout the course and the instructor perceived that students progressed more quickly and confidently than in prior offerings of the course.

6 Instructor Experience and Lessons Learned

The benefits of the PathMX methodology were evident in the instructor's experience. The methodology allowed for much more personalized instruction and feedback as the class progressed. In practice, creating personalized assignments and guides took under five minutes of agent interaction plus a few

minutes of instructor review, compared to an estimated 30–60 minutes of manual authoring. Because of the time savings, the instructor was able to focus on higher-level pedagogical decisions and personal attention to students rather than on routine content creation.

For instructors considering PathMX, we recommend the following based on our experience:

- Start with a small set of materials in a version-controlled repository.
- Build a style guide and clearly defined outcomes and scope early and iterate on them as the course progresses.
- Keep students' personal data private, secure, and anonymized for agent interactions.

Despite significant benefits, several challenges emerged. In early projects, when fewer examples were present, agents sometimes generated material that was beyond the students' current level of understanding. More explicit documentation about the scope of the course was created to avoid content drift. This underscores the importance of the instructor's role in vetting material before it is released to students.

Agents were often too helpful and would spell out the solution to the problem rather than guide the student to the solution. In practice, this required explicitly instructing agents not to reveal complete solutions, but instead to provide scaffolded guidance that led students through the problem-solving process without giving away answers.

We invite instructors to explore the PathMX methodology and tools to realize the benefits of personalized, differentiated, and scalable instruction in their own courses. We have provided a sample repository with instructions and guides for how to get started[2]. We also encourage instructors to share their experiences and feedback with us so we can continue to improve the methodology and tools.

7 Future Work

This work represents an initial exploration of agent-assisted curriculum development. Our observations come from a single semester deployment with 18 students. Larger-scale research studies are warranted to formally measure the impact of the PathMX methodology on student learning and instructor efficiency. We are currently designing a controlled study to evaluate PathMX across multiple CS contexts.

[2] `https://github.com/pathmx/cs1-example`

Additional areas of exploration include assistive automated grading and assessment, the use of PathMX agents for real-time feedback and tutoring, as well as automated skill mapping derived from student code and pre-assessment data. The PathMX methodology also lends itself to generating student-facing support materials that surface and correct common student misconceptions, building on prior work that systematically identifies misconceptions in introductory programming [8].

We are also investigating implications for agent-friendly Learning Object Repositories (LoRs), alternative learning object formats, adaptive learning pathways, and student-facing tutoring agents.

8 Conclusions

Authoring curriculum as structured, version-controlled code enables modern agent-based tools to deliver differentiated instruction at a scale that was previously impractical. PathMX provides a practical path forward, with conventions and workflows that work today while remaining flexible enough to evolve as the technology matures.

Our CS1 case study suggests that this approach makes differentiated instruction more attainable in actual practice. By enabling rapid generation of multiple representations and engagement pathways, PathMX operationalizes UDL principles at scale. In a single semester, we created personalized learning artifacts for each student, generated project variants tailored to different interests, and provided detailed, actionable code review feedback while preserving instructor control and oversight using familiar software engineering workflows. This allowed us to iterate on the curriculum throughout the semester and maintain consistency and quality across all materials. The same conventions that support agent inference, namely type hints, frontmatter metadata, and hyperlinks, also improved human navigation, reuse, and mid-term iteration.

At the same time, this work is exploratory and intentionally modest in scope. Questions about long-term outcomes, instructor workload, and student equity across diverse contexts remain open and motivate the future work described in the previous section. We argue that PathMX provides a practical way forward and a viable design space for next-generation computing education: one that keeps instructors in control but allows for truly scalable differentiated instruction that meets students in their own context.

References

[1] CAST. *Universal Design for Learning Guidelines version 3.0*. 2024. URL: `https://udlguidelines.cast.org`.

[2] André Dietrich. “LiaScript: a domain-specific-language for interactive online courses”. In: *Proceedings of the International Conference on e-Learning 2019*. Porto, Portugal, 2019, pp. 186–194.

[3] Barbara J. Ericson, James D. Foley, and Jochen Rick. “Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems”. In: *Proceedings of the 2018 ACM Conference on International Computing Education Research*. ICER ’18. Espoo, Finland: Association for Computing Machinery, 2018, pp. 60–68. ISBN: 9781450356282. DOI: 10.1145/3230977.3231000.

[4] Barbara J. Ericson and Bradley N. Miller. “Runestone: A Platform for Free, On-line, and Interactive Ebooks”. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE ’20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 1012–1018. ISBN: 9781450367936. DOI: 10.1145/3328778.3366950.

[5] Nuno Gil Fonseca, Luís Macedo, and António José Mendes. “Supporting Differentiated Instruction in Programming Courses through Permanent Progress Monitoring”. In: *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. SIGCSE ’18. Baltimore, Maryland, USA: Association for Computing Machinery, 2018, pp. 209–214. ISBN: 9781450351034. DOI: 10.1145/3159450.3159578.

[6] Courtney Hsing and Vanessa Gennarelli. “Using GitHub in the Classroom Predicts Student Learning Outcomes and Classroom Experiences: Findings from a Survey of Students and Teachers”. In: *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. SIGCSE ’19. Minneapolis, MN, USA: Association for Computing Machinery, 2019, pp. 672–678. ISBN: 9781450358903. DOI: 10.1145/3287324.3287460.

[7] Maya Israel et al. “Teaching Elementary Computer Science through Universal Design for Learning”. In: *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. SIGCSE ’20. Portland, OR, USA: Association for Computing Machinery, 2020, pp. 1220–1226. ISBN: 9781450367936. DOI: 10.1145/3328778.3366823.

[8] Lisa C. Kaczmarczyk et al. “Identifying Student Misconceptions of Programming”. In: *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. SIGCSE ’10. Milwaukee, Wisconsin, USA: Association for Computing Machinery, 2010, pp. 107–111. DOI: 10.1145/1734263.1734299.

[9] Simon Larsén and Richard Glassey. “RepoBee: Developing Tool Support for Courses using Git/GitHub”. In: *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. ITiCSE ’19. Aberdeen, Scotland Uk: Association for Computing Machin-

ery, 2019, pp. 534–540. ISBN: 9781450368957. DOI: 10.1145/3304221.3319784.

[10] Juho Leinonen et al. "Comparing Code Explanations Created by Students and Large Language Models". In: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. ITiCSE 2023. Turku, Finland: Association for Computing Machinery, 2023, pp. 124–130. ISBN: 9798400701382. DOI: 10.1145/3587102.3588785.

[11] Stephen MacNeil et al. "Experiences from Using Code Explanations Generated by Large Language Models in a Web Software Development E-Book". In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. SIGCSE 2023. Toronto ON, Canada: Association for Computing Machinery, 2023, pp. 931–937. ISBN: 9781450394314. DOI: 10.1145/3545945.3569785.

[12] Bradley N. Miller and David L. Ranum. "Beyond PDF and ePub: toward an interactive textbook". In: *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*. ITiCSE '12. Haifa, Israel: Association for Computing Machinery, 2012, pp. 150–155. ISBN: 9781450312462. DOI: 10.1145/2325296.2325335.

[13] Julianna Rodriguez et al. "Courseware as Code Setting a new bar for transparency and collaboration". In: *2018 IEEE Frontiers in Education Conference (FIE)*. 2018, pp. 1–4. DOI: 10.1109/FIE.2018.8658928.

[14] Sami Sarsa et al. "Automatic Generation of Programming Exercises and Code Explanations Using Large Language Models". In: *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*. ICER '22. Lugano and Virtual Event, Switzerland: Association for Computing Machinery, 2022, pp. 27–43. ISBN: 9781450391948. DOI: 10.1145/3501385.3543957.

[15] Clifford A. Shaffer et al. "OpenDSA: beginning a community active-eBook project". In: *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*. Koli Calling '11. Koli, Finland: Association for Computing Machinery, 2011, pp. 112–117. ISBN: 9781450310529. DOI: 10.1145/2094131.2094154.

[16] Carol Ann Tomlinson. *How to Differentiate Instruction in Academically Diverse Classrooms*. 3rd. Alexandria, VA: ASCD, 2017. ISBN: 9781416623335.

[17] Carol Ann Tomlinson. *The Differentiated Classroom: Responding to the Needs of All Learners*. 2nd. Alexandria, VA: ASCD, 2014. ISBN: 9781416618607.