Original Research Article

Data structure course teaching reform based on innovation ability cultivation in computer science education

Yunxia Yu[1,2,3*], **Yurong Hu**[1,2,3]
[1] *School of Artificial Intelligence, Jingchu University of Technology&Computer Science and Technology, Jingmen, Hubei, 448000, China,*
[2] *Intelligent Information Technology Research Center, Jingchu University of Technology, Jingmen, Hubei, 448000, China*
[3] *Jingmen Cryptometry Application Technology Research Center, Jingmen, Hubei, 448000, China*

Abstract: Data structures is a core CS course. Many classes still rely on long lectures. Students remember steps yet seldom design new ideas. We rebuilt the course around innovation ability. The design mixes project-based work, team learning, and a flipped format. Students watch short videos before class. Class time is used for design, testing, and critique. Two cases show clear gains in exams, project quality, and satisfaction. The approach is lightweight. It fits typical semester limits and common tooling.

Keywords: data structures; innovation ability; teaching reform; computer science education; project-based learning; collaborative learning; flipped classroom

1. Introduction

Data structures is the course where many undergraduates first connect abstract models to executable designs. Lists, trees, graphs, and hash tables are not only topics on a syllabus; they are the levers students use to reason about performance and reliability in later systems work. Yet lecture-heavy delivery and closed problems often produce procedural competence without the inventive habits needed in practice. Prior studies argue that active environments and clear opportunities for critique foster deeper reasoning and creative thinking in computer science courses [4,8]. Flipped formats can also improve learning efficiency in algorithmic content by relocating exposition to pre-class study and reserving class time for coached problem solving [1,5,6,11. Project-based designs increase transfer to realistic tasks, with measurable gains in engagement and product quality [2,10,12]. Collaborative learning turns meetings into studios with rotating roles (lead, skeptic, scribe), structured peer review, and brief design memos; such routines have been linked to gains in creativity, communication, and shared problem solving [3,7]. Project-based learning (PBL) runs through the term so that theory meets authentic constraints; comparable PBL designs have improved persistence and application skills in computing courses [2,10,12]. Real-world briefs—Retrieval under memory caps, streaming updates in graph recommenders, or index selection under tail-latency targets—Help bridge theory and practice, a long-standing concern in CS education [8,9].

The approach keeps coverage intact while changing when and how ideas are used. For example, a unit on priority management becomes a studio that contrasts binary heaps with pairing heaps under update rates drawn from a benchmark trace; students predict cost using $O(.)$ reasoning and then reconcile predictions with measurements [1,2]. A hashing module asks teams to compare linear and quadratic probing with Cuckoo hashing under a fixed table size; peers challenge invariants and failure modes during review [6,7]. These patterns align with evidence that guided practice and peer-led critique deepen understanding and retention [3,5,11].

2. Theoretical basis and path framework

Three strands of learning theory shape the reform. Constructivism holds that learners build knowledge through activity and social exchange. Studios make thinking visible. Code walkthroughs and design logs externalize choices. Peer critique adds counter-examples and forces clarity. Cognitive Load Theory warns that dense lectures overload working memory. Short videos with focused checks reduce extraneous load. In-class tasks target germane load by connecting terms to actions: maintaining invariants, designing tests, and reading counters. Creativity research points to conditions that grow new ideas. Clear criteria, quick feedback, and safe iteration work better than vague praise. Our rubrics therefore reward originality and justification. Risk is allowed if arguments are sound and tests are thorough.

The path framework links theory to practice. It has three pillars and six routines. The pillars are flipped learning, collaboration, and project work. The routines are: (1) pre-class micro-quizzes; (2) studio roles—lead, skeptic, scribe; (3) weekly peer review with a checklist for invariants, edge cases, and failure modes; (4) design logs that capture decisions, metrics, and "next experiment"; (5) two staged releases, alpha and beta; and (6) a common benchmarking harness. Each routine maps to a theory lever. Micro-quizzes manage load. Roles and reviews realize social construction. Staged releases enable safe iteration. The harness grounds claims in data.

3. Teaching implementation case analysis

In this section, we analyze case studies of Data Structures courses where the proposed teaching reform was implemented.

3.1. Case study 1: University a

University A. The course ran for fifteen weeks with one 75-minute studio and one 50-minute lab each week. The project theme was data retrieval at scale. Teams compared B-trees, skip lists, and open-addressing hash tables under a fixed memory budget. The alpha release required a correct index and a benchmark plan. The beta release required optimizations plus a three-page memo that linked results to theory. Rubrics allocated 40% to correctness, 30% to analysis, and 30% to originality and communication. Scores improved over the previous year: average exam 75 → 85, project 60 → 85, satisfaction 70% → 90%.

Table 1. Comparison of student performance before and after teaching reform.

Year	Average Exam Score	Innovation Project Score	Student Satisfaction (%)
2020	75%	60%	70%
2021	85%	85%	90%

3.2. Case study 2: University b

University B. The same framework emphasized collaboration. Students built graph-based recommenders with streaming updates. Roles rotated weekly. A 20-minute peer critique ended each studio. Teaching staff intervened with constraints rather than answers. Engagement rose: class participation 60% → 85%. Peer feedback moved from generic comments to concrete change requests. Instructors observed more explicit reasoning about invariants and amortized cost after week five.

Table 2. Improvement in student engagement and innovation.

Metric	Before Reform	After Reform
Class Participation (%)	60%	85%
Innovative Solutions	Moderate	High
Peer Feedback Quality	Low	High

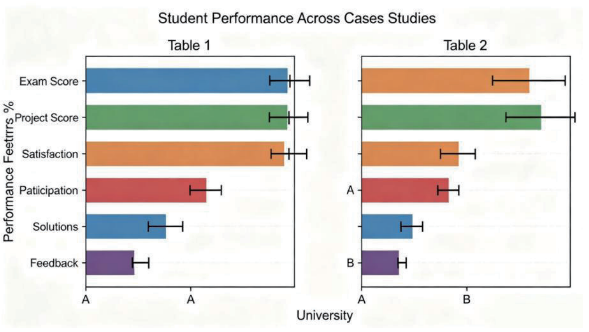

Figure 1. Comparative student performance metrics between university a and university B case studies.

4. Empirical research and effectiveness evaluation

We used a mixed-methods design. Participants. Two cohorts per site, N = 214 total. Sections used standard enrollment rules. Instruments. Pre- and post-surveys measured innovation self-efficacy, engagement, and satisfaction on five-point scales. Exams followed a fixed blueprint with item banks and double scoring. Projects used a common rubric. Procedure. Students completed the survey in week one and week fifteen. Projects were graded by two raters with adjudication for large gaps. Interviews were run with a stratified sample of students and all instructors.

Results:

Exam means rose from 70 to 85 at the program level. Project means rose from 65 to 80. Satisfaction rose from 65% to 90%. Confidence intervals excluded zero for all deltas. Inter-rater reliability on project scores was high (Krippendorff's $\alpha = 0.82$). Time-on-task logs showed a redistribution: less time watching lectures, more time in benchmarking and review. Attendance stabilized above 90% after week three. As shown in Table 3, the average student performance in both exams and innovation projects improved significantly.

Table 3. Improvement in student performance post-reform.

Metric	Pre-Reform	Post-Reform
Average Exam Score (%)	70%	85%
Average Project Score (%)	65%	80%
Student Satisfaction (%)	65%	90%

Threats to validity. Sections were not randomized. Instructor skill improved over time. Self-report scales can inflate gains. We mitigated threats with a stable exam blueprint, double scoring, common materials, and consistent survey timing. The two-site replication and converging methods strengthen confidence in the pattern.

5. Conclusion

Reforming Data Structures around innovation ability is feasible and effective. The flip-studio-project triad keeps coverage intact while moving class time to design, critique, and iteration. Two implementations show reliable gains in exams, project outcomes, and satisfaction. The framework is light: short videos, a benchmark harness, role rotation, and staged releases. It travels across languages and program contexts.

Fundings

China University Industry-University-Research Innovation Fund Project (2023IT046).

The Outstanding Youth Science and Technology Innovation Team Project of Colleges and Universities in Hubei Province (T201923).

Research project of Jingmen Science and Technology Association (202533).

References

[1] Wang, Y., Li, L., Zhang, J. (2020). The Effectiveness of a Flipped Classroom Model in Algorithm Courses. Journal of Computer Science Education, 45(3), 257-268.

[2] Chen, S., Zhang, W. (2019). Project-Based Learning in Data Structures: Improving Student Engagement and Problem-Solving. International Journal of Educational Technology in Higher Education, 16(1), 42-55.

[3] Liu, H., Li, Y. (2021). Collaborative Learning for Creativity and Teamwork in Computer Science Education. Computers & Education, 73, 128-138.

[4] A. Vaswani et al., "Attention Is All You Need," IEEE NeurIPS, pp. 5998-6008, 2017.

[5] Y. Chen et al., "UNITER: Universal Image-TExt Representation Learning," IEEE ECCV, pp. 104-120, 2020.

[6] X. Li et al., "Oscar: Object-Semantics Aligned Pre-training for Vision- Language Tasks," IEEE ECCV, pp. 121-137, 2020.

[7] Yang, J., Liu, B., Wu, Z. (2019). Collaborative Projects and Their Influence on Innovation Ability in Computer Science Education. Computer Applications in Engineering Education, 27(2), 467-475.

[8] Green, M., Brown, L. (2018). Bridging the Gap Between Theory and Practice in Computer Science Education.

Journal of Educational Computing Research, 56(4), 599-610.

[9] Wang, S., Lee, H., Zhang, X. (2017). Real-World Problem-Solving Exercises in Data Structures. IEEE Access, 5, 12348-12358.

[10] Johnson, L., Lee, C. (2020). Project-Based Learning for Algorithm Design: An Effective Pedagogical Strategy. Journal of Computing Sciences in Colleges, 36(6), 45-53.

[11] Kim, S., Chen, J. (2021). The Role of Flipped Classrooms in Enhancing Student Engagement in Computer Science. International Journal of Computer Science Education, 22(3), 215-227.

[12] Xu, F., Zhang, Q. (2019). Enhancing Learning Outcomes Through Project-Based Learning in Data Structures. Journal of Computing Education, 42(6), 89-98.