Memorization to Categorization: Improving Students' Comprehension of SDLC Models

[1]K.Gomathi, [2]Gudapati Lakshmi Bhavani, [3]Khamruddin Syed
[1,2,3]KG Reddy College of Engineering and Technology
[1]Department of CSE(DS), [2]Department of ECE, [3]Department of CSE(AIML)
[1] gomathi.k@kgr.ac.in, [2]lakshmisrinivas.2010@kgr.ac.in, [3]syedkhamruddin@kgr.ac.in

***Abstract*—Sophomore-level students of engineering institutions of computer science engineering often face challenges in differentiating software development models due to the overlap between them, which leads to difficulties in recalling and categorizing. The effectiveness of pedagogical interventions for improving the conceptual understanding of Software Development lifecycle models is investigated in the study. A total of one hundred and twenty students were divided into experiment and control groups with varied instructions, which incorporated concept maps, comparative case exercises, and gamified learning activities. A pre- and post-test design was administered, and the data were analyzed using a paired t-test and an independent t-test. The post-test analysis shows significant improvement in the experimental group of students (M=78.4, SD = 5.9) compared to the control group (M=61.2, SD=7.4; t(118) = 13.27, p < 0.001). A one-way ANOVA confirmed a strong effect of instructional approach on performance, F(1,118) = 176.2, p < 0.001, and the intervention showed a significant effect (Cohen's d=0.92). Overall, this structured activity mentioned in the study improved students' ability to differentiate and categorize software development logic models, indicating the results, which are both practically and statistically meaningful.**

***Keywords*—Cognitive Load Theory; Categorization; Concept mapping; Gamification; Software Engineering**

ICTIEE* Track—*Emerging technologies and future skills
ICTIEE* Sub-Track—*Pedagogy for the Modern Classroom-Strategies for Engaging Students through AI

I. INTRODUCTION

ONE of the foundational courses in computer science engineering curriculum is software engineering. This course intends to develop conceptual understanding and industry-ready skills. However, many sophomore-level students find the fundamental ideas behind software development models, such as the Waterfall, Unified Process, Incremental, Spiral, and Agile development models, to be challenging, as many concepts overlap with each other. For those still developing their understanding of abstraction and knowledge classification, these models often appear confused, and this week's differentiation reduces retention, making it harder for students to grasp the concepts. These kinds of difficulties are widely reported in the field of computing education. Pessoni et al. (2015) emphasized that undergraduate engineering students frequently struggle to convert classroom instructions into abstract knowledge when foundational reasoning skills and scaffolded instructions are missing (Pessoni et al., 2015). Abstract concepts such as data structures and algorithms observe similar patterns where the cognitive demands and persistent misconceptions are obvious (Mtaho & Mselle, 2024). In courses like software engineering, this is viewed as rote memorization, which includes concepts of list of phases, pros and cons, and terminologies rather than the conceptual mapping needed to meaningfully differentiate the models. These problems addressed in this study are grounded in the Cognitive Load Theory (CLT). As working memory can handle only a few chunks of information, tasks that are added unnecessarily interfere with schema construction and the integration of new ideas into long-term memory (Winter et al., 2019; Syed et al., 2025). Traditional lectures often deliver the phases of software development phases with overlapping (e.g planning, modelling, construction, testing) without proper visual representations which pushes the students to learn the models as interchangeable and resulting into shallow learning. Structured instructions including concept mappings and gamified learning can address these issues reducing extraneous load, supporting schema formation and encouraging active engagement. Students also construct knowledge by doing, comparing, and reflecting, applying various software development models to real-time scenarios. They evaluate the models side by side and choose the appropriate ones. These structured pedagogies enhance the students learning by strengthening the conceptual clarity and promoting deep learning (Upchurch & Sims-Knight, 1997)

II. BACKGROUND STUDY

Students in computer science education face persistent challenges in categorization and abstraction. Not only the programming, but also the concepts related to modeling and systems thinking, have difficulties with abstract concepts and categorization. For instance, Silva et al. (2019) highlighted that students often struggle with mastering software modeling, especially when using UML diagrams. Many students find it challenging to understand software engineering models and methods. This difficulty is often attributed to the complexity of

Khamruddin Syed
KG Reddy College of Engineering and Technology Hyderabad
syedkhamruddin@kgr.ac.in

the material and the everyday use of teacher-centered instructional approaches (Silva et al., 2019). This observation is supported by other research indicating that students frequently struggle to differentiate between various software engineering concepts. This happens due to the poorly organized instructions which just include text in the form of bullet points or long paragraphs of content explanations (Mtaho & Mselle, 2024). This problem is grounded in Cognitive Load Theory (CLT), which emphasizes the importance of reducing unnecessary cognitive load while promoting the mental effort required to build knowledge structures, or schemas. In computer science, strategies such as visualization tools and concept maps have been found to ease cognitive load and enhance understanding of core concepts by creating long-term schemas (Yousoof et al., 2007; Winter et al., 2019). Same interventions can be used for the Software Engineering course concepts to help students to visualize the models using concept maps, tabular representations, etc. This will enable the course instructors in reducing the extraneous load on the learners and support them in building the mental frameworks needed to differentiate between models.

The limitations of traditional lecture-based teaching have prompted the broader adoption of active learning methods in computer science education. Project-based learning, peer instruction, and gamification have improved student engagement, motivation, and learning outcomes. For instance, Morais et al. (2021) found that problem-based learning significantly increased student engagement and success in courses like software engineering and information systems development, as students could experience the iterative nature of the development process firsthand. It is also evident as per the existing research that collaborative learning in concepts such as testing and integration improved cognitive performance in entire software engineering courses (Gopal and Cooper, 2001). Recent studies also support the integration of interactive and playful strategies. Studies conducted by past researchers conducted randomized control interventions to check the effectiveness of L-E-G-O series play activity (Lopez et al. , 2024; Shet et al., 2015). The findings also showed that students who participated in this interactive learning method understood SDLC models much better than those taught via traditional lectures. These results underscore the effectiveness of gamified and hands-on approaches for teaching complex, abstract topics like software development models.

The problem addressed in this study is grounded in both Constructivist Learning Theory (CLT) and Cognitive Load Theory. CLT focuses on reducing the unnecessary cognitive load demand on the student and allows schema construction, whereas Constructivism emphasizes active participation and the construction of knowledge over a period of time. An OpenSMALS developed by Silva et al. (2020) for teaching UML reduced the difficulties addressed above and improved the intended learning outcomes. By grounding in the mentioned theories and aligning with existing literature, this current study contributes by designing and administering a unique instructional model that employs concept mapping, gamified learning in software development life cycle models.

III. METHODOLOGY

In this study, a control design with a quasi-experimental approach is implemented. Pre and post-test data were collected the data to evaluate the effectiveness of the interventions. The quantitative data is collected through assignments for both the control and experimental groups. While accommodating the classroom setting, the casual relationships are assessed by using this design. Students from Computer Science and Engineering (AIML) have participated in the study. Participants were second-year undergraduate students enrolled in the core Software Engineering course. A total of 120 students participated, with 60 students assigned to the control group and 60 students assigned to the experimental group. Both groups were taught by the same instructor to ensure consistency of delivery and to minimize instructor-related variability. Students' demographic backgrounds were diverse, though all had completed prerequisite programming and introductory computing courses, ensuring a comparable baseline of prior knowledge.

The control group received traditional lecture-based instruction, focusing on descriptive presentations of SDLC models. The experimental group, however, was exposed to a blended instructional approach designed around three strategies:

1. Concept Mapping: Visual diagrams were used to highlight similarities and differences across models. Students were formed into groups of 3 and given keywords related to the SDLC models to compare based on parameters like iterations, increments, risk handling, requirements, etc. They were asked to draw the concept map in the form of a chart or a tabular form and were evaluated. The peer evaluation was done on the basis of completeness, clarity, correctness, and accuracy. This activity was conducted for 30 minutes.

2. Comparative Case exercises: Students analyzed real-world scenarios to identify the most appropriate SDLC model. The students of a group of 3 members were given an additional 30 minutes to identify a suitable model for a real-world scenario or application. Students did this using various dimensions, such as most suitable, trade-off reasons, and justifications. After this exercise, they presented information to the entire class, and the rest of the groups evaluated using the same dimensions.

3. Gamification: Students use gamification tools like H5P and Mentimeter to participate in gamified learning to drag and drop , match UML diagrams and also to attempt the quizzes.

These interventions were delivered across four consecutive weeks, aligning with the curriculum schedule for software engineering models. A test consisting of MCQs and descriptive questions was administered as both a before and after test. The reliability of the instrument was established using Cronbach's alpha ($\alpha = 0.82$), indicating high internal consistency.

1. Pre-test: administered one week before the instructional unit, assessing baseline understanding.

2. Post-test: administered immediately after the four-week intervention, measuring knowledge gains.

Both control and experimental groups completed the same assessments under identical conditions.

The study unfolded in three phases which is presented as a conceptual diagram Fig. 1:

1. Preparation Phase: Development of instructional materials, validation of test instruments.

2. Implementation Phase: Delivery of instruction to both groups, with the control group taught via conventional lectures and the blended approach for experimental group of students.

3. Evaluation Phase: administration of the post-test, followed by statistical analysis of results

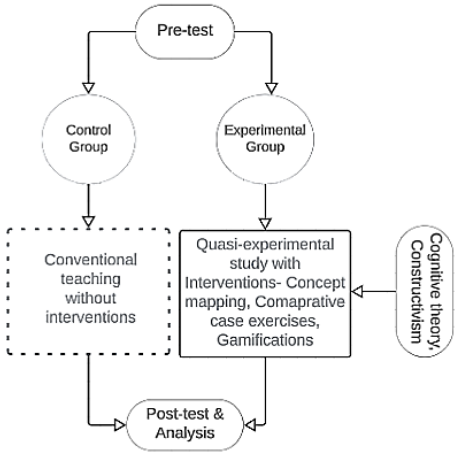

Fig. 1. Conceptual diagram

Attendance was monitored to ensure consistent exposure, and all students participated voluntarily, with informed consent obtained before the study.

Quantitative data were analyzed, where descriptive statistics such as mean, SD were first calculated to summarize performance. To test for significant differences:

1. Paired-sample t-tests: To compare pre- and post-test scores within the group.

2. Independent-sample t-tests: To compare post-test performance between the experimental and control groups.

3. One-way ANOVA: To confirm the effect of instructional approach on learning outcomes.

4. To determine the magnitude of the intervention's impact, the effect size (Cohen's d) was calculated.

A significance level of $p < 0.05$ was adopted for entire inferential tests. Results demonstrated statistically significant improvements in the experimental group, supporting the effectiveness of the intervention.

IV. Results

A. Descriptive Statistics

TABLE I shows the mean and standard deviations for both groups in pre- and post-tests. The pre-test scores indicated no major differences between the control group (M = 42.5, SD = 6.8) and the experimental group (M = 43.1, SD = 7.2), confirming baseline equivalence. After the instructional intervention, the experimental group showed a notable results (M = 78.4, SD = 5.9), whereas the control group demonstrated moderate gains (M = 61.2, SD = 7.4).

TABLE I
PRE- AND POST-TEST SCORES

Source	SS	df	MS	F	p-value
Between Groups	12850.4	1	12850.4	176.2	< 0.001
Within Groups	4320.7	118	36.6	–	–
Total	17171.1	119	–	–	–

B. Inferential Statistics

Paired-sample *t*-tests indicated significant learning gains within both groups: control ($t(59) = 15.4$, $p < 0.001$) and experimental ($t(59) = 28.9$, $p < 0.001$). The post-test results for the experimental group show higher performance than the control group ($t(118) = 13.27$, $p < 0.001$).

A one-way ANOVA confirmed that the instructional method had a statistically significant effect on student outcomes ($F(1,118) = 176.2$, $p < 0.001$). The effect size (Cohen's $d = 0.92$) indicated a strong practical impact of the intervention as shown in TABLE-II

TABLE II
ANOVA SUMMARY

Group	Pre - test Mean (SD)	Post - test Mean (SD)	N
Control	42.5 (6.8)	61.2 (7.4)	60
Experimental	43.1 (7.2)	78.4 (5.9)	60

These findings quantitatively establish that the blended instructional approach substantially improved students' ability to differentiate and apply software development models compared to traditional instruction. Fig. 2 shows the pre- and post-test comparison

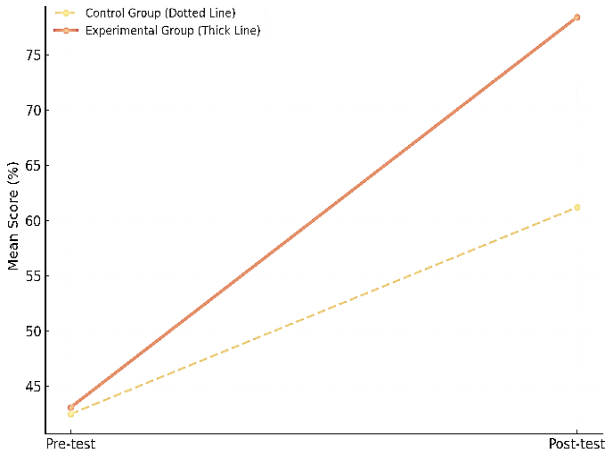

Fig. 2. Pre and Post test comparisons

V. DISCUSSION

The results shows that integrating concept maps, comparative case-based learning, and gamified activities significantly enhanced students' conceptual understanding of software engineering models. While both groups showed progress, the experimental group achieved substantially higher gains, providing empirical support for the use of active and visual instructional strategies. Project-based and gamified methods have previously been shown to increase motivation and success rates in software engineering contexts (Morais et al., 2021). Similarly, peer instruction and collaborative learning have improved outcomes in areas such as software testing and modeling (Gopal & Cooper, 2021). The current findings contribute to this growing body of literature by showing that even fundamental content, such as differentiating SDLC models, benefits from active learning designs. The study offers practical implications for educators in computer science and engineering. First, reliance on lecture-based approaches is insufficient for promoting deep learning of overlapping models. Instead, instruction should deliberately incorporate comparative visualization tools and scenario-based exercises to foster conceptual differentiation. Second, the strong effect size observed suggests that relatively small modifications to pedagogy can yield substantial improvements in learning outcomes, making these strategies feasible for adoption in resource-constrained educational contexts. These results cannot be generalized as this is limited to one institution. Additionally, the focus was on short-term knowledge gains measured immediately after the intervention; long-term retention was not evaluated. Future research should examine the durability of learning gains, explore scaling the intervention to larger classes, and investigate the integration of digital tools (e.g., interactive simulations, adaptive learning systems) to further enhance effectiveness, including mixed methods approach.

CONCLUSION

This study explored the current challenges experienced by second-year CSE-AIML students in distinguishing SDLC models. With this design for 120 students, the study investigated the relative effectiveness of traditional lecture-based instruction versus an integrated approach that included concept maps, comparative case-based exercises, and gamified activities. These findings suggest that the experimental group of students improved in comparison to the control group in specific tasks in the post-test (Cohen's d = 0.92). The findings also suggest that traditional lectures alone cannot resolve students' difficulties in understanding the overlapping models in a software engineering course. However, a blended instruction that includes active and visual learning aids helps students learn, which in turn helps develop their schema. This study is performed in a single institutional setting, so the generalization is limited. Further research on diversified settings with a longitudinal study is recommended.

REFERENCES

Gopal, B., & Cooper, S. (2021). Peer instruction in software testing and continuous integration. Proceedings of the 52nd ACM Technical Symposium on Computer Science Education.

Lopez-Fernandez, D., Tovar, E., Gordillo, A., Gayoso-Cabada, J., Badenes-Olmedo, C., & Cimmino, A. (2024). Comparing a LEGO Serious Play activity with a traditional lecture in software engineering education. IEEE Access, 12, 74045–74053.

Morais, P., Ferreira, M. J., & Veloso, B. (2021). Improving student engagement with project-based learning: A case study in software engineering. IEEE Revista Iberoamericana de Tecnologías del Aprendizaje, 16(1), 21–28.

Mtaho, A., & Mselle, L. J. (2024). Difficulties in learning the data structures and algorithms course: Literature review. The Journal of Informatics, 4(1).

Pessoni, V. V., Federson, F. M., & Vincenzi, A. (2015). Learning Difficulties in Computing Courses: Cognitive Processes Assessment Methods Research and Application. Proceedings of the Brazilian Symposium on Information Systems, 31–38.

Shet, A., Bhavani, P., Kumarasamy, N., Arumugam, K., Poongulali, S., Elumalai, S., & Swaminathan, S. (2015). Anemia, diet and therapeutic iron among children living with HIV: A prospective cohort study. BMC Pediatrics.

Silva, W., Gadelha, B. F., Steinmacher, I., & Conte, T. (2020). Towards an open repository for teaching software modeling applying active learning strategies. 2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), 162–172.

Silva, W., Steinmacher, I., & Conte, T. (2019). Students' and instructors' perceptions of five different active learning strategies used to teach software modeling. IEEE Access, 7, 184063–184077.

Syed, K., M, V., Kandakatla, R., & Narayanan, S. (2024). Inculcating Multidisciplinary Learning in Electrical Engineering Undergraduate Curriculum through

Problem-Based Learning. Journal of Engineering Education Transformations 38(is1), 210–215. https://doi.org/10.16920/jeet/2024/v38is1/24234

Upchurch, R., & Sims-Knight, J. (1997). Designing process-based software curriculum. Proceedings Tenth Conference on Software Engineering Education and Training, 28–38.

Winter, V., Friend, M., Matthews, M., Love, B., & Vasireddy, S. (2019). Using visualization to reduce the cognitive load of threshold concepts in computer programming. 2019 IEEE Frontiers in Education Conference (FIE), 1–9.

Yousoof, M., Sapiyan, M., & Kamaluddin, K. (2007). Reducing cognitive load in learning computer programming. World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 1(11), 4100–4103.