Software Architecture Teaching: A Systematic Review

Ugyen Lhatshok
u3265091@uni.canberra.edu.au
University of Canberra

Pema Gyamtsho
u3267717@uni.canberra.au
University of Canberra

Thinley Dema
u3255665@uni.canberra.edu.au
University of Canberra

Thinley Rabgay
u3267425@uni.canberra.au
University of Canberra

Zhijia Ren
u3262965@uni.canberra.edu.au
University of Canberra

1. ABSTRACT

Software Architecture (SA) education is critical in equipping students with complex software design skills required by the industry. However, software architecture teaching faces distinct challenges due to its abstract and conceptual nature, high-level thinking and critical reasoning requirements, and the mismatch between SA education and industrial needs. This systematic review analyses 82 peer-reviewed studies published between 2010 and 2024 to identify and examine current teaching strategies, their assessment methods, learning outcomes and challenges. Ten teaching themes were identified, with Project-Based Learning (PBL), Real-World-Based Learning, and Collaborative Learning being the most frequently adopted strategies. Summative assessment practices are the dominant assessment approaches. However, our analysis highlights the additional value from experiential assessments, such as Architectural Kata, that align with industry needs. Learning outcomes are evaluated based on Bloom's taxonomy, and these themes achieve higher-order learning outcomes ("Apply" and "Understand"), but not the advanced levels ("Create" and "Evaluate"). Despite the achievements, significant challenges exist, including abstract and conceptual difficulties, time and workload constraints for students, heavy workloads, a lack of practical experience, and insufficient educational training for instructors. Our review offers an interrelated analysis of SA teaching strategies, particularly from the students' perspective, based on which we call for greater collaboration among all stakeholders, including the adoption of diverse teaching strategies by institutions, selection of suitable projects to balance workload and industrial complexity by instructors and maintenance of engagement from students. Our review helps educators, curriculum designers, and industry stakeholders enhance SA education's relevance, engagement, and effectiveness.

Keywords: *Software architecture, Teaching strategies, Education, Assessment Methods, Hands-on experiences*

2. INTRODUCTION

Software Architecture (SA) refers to designing the architecture, including its interfaces and interactions among modules and elements used in operating and setting a software or system [1]. This makes software architecture a fundamental aspect of software engineering, serving as the blueprint that dictates a software system's structure, interactions, and quality attributes. It plays a pivotal role in ensuring system performance, maintainability, scalability, and security, making it a critical field of study for software engineers and architects [2].

Despite its importance, software architecture teaching faces distinct challenges. Unlike other software engineering courses that commonly focus on coding and implementation, SA is inherently abstract and conceptual, requiring students to have high-level design thinking, system thinking, critical reasoning, and trade-off analysis. These skills are complex to teach or assess. In addition, SA rarely has a "current" solution, but it requires students to justify the most appropriate design [3-5]. Another key challenge is the gap between theory and practice, which means that software architecture educators must equip students with sufficient conceptual knowledge and hands-on experience to meet industrial expectations [6-8]. This further requires SA educators to have good educational skills and training, enough real-world experience, and the capacity to work under tight schedules and tough loads, all increasing difficulties in providing high-quality SA education [9-11].

Given these complexities, this study aims to review the existing literature on software architecture education systematically. Our research is motivated by the persistent gap between academic practices and industry expectations, and the limited attention paid to students' learning challenges in existing mapping studies. Our research objective is to conduct a more integrated understanding of how SA teaching strategies assess their students and their outcomes and equip them with the required industrial skills and knowledge, as well as the challenges, particularly from the student perspective.

Based on 82 peer-reviewed studies, this review identifies ten major teaching themes and analyses their alignment with industrial expectations, challenges of implementation and learning outcomes using Bloom's taxonomy. This makes our research different from prior mapping studies on SA education. Firstly, our study explores the interactive relationships among strategies, assessments, outcomes and educational challenges. Secondly, it focuses on the students' perspective, offering new insights into how SA education can equip students with sufficient skills for industry needs.

This study has both literary and practical significance. Firstly, our study summarised the most influential teaching strategies currently adopted by various teaching institutions based on prior literature into teaching themes. In addition to the synthesis, our study evaluates these themes comprehensively based on their ability to offer hands-on practices based on industrial needs, assessment methods, outcomes and challenges. All these offer additional insights into SA education. Further, our study critically analyses these themes from the students' perspective, which provides reasonable indications to practical educators and stakeholders, including curriculum designers, software developers, tutors, etc. It helps them to know the challenges and expectations from students, directing them to appropriate adjustments to their current teaching strategies and curriculum development.

3. RELATED WORK

Systematic review refers to synthesising secondary data in a detailed manner that integrates all the primary studies for a review, providing a critical synthesis [12]. It is also widely used to evaluate and interpret all available research relevant to a particular research question, topic area or phenomenon of interest, thus ensuring fair evaluation by employing a trustworthy, rigorous and auditable methodology [13]. As a multifaceted subject, Software Architecture Education has been extensively investigated through various case studies and teaching methods. However, systematic analysis in this field remains

limited. This section will thus delve into the work of researchers who have conducted a systematic review of Software Architecture (SA) Education.

Rodrigues [14] conducted one of the inaugural systematic reviews targeting software architecture education. They assessed 37 primary studies, categorising teaching methods based on course design, active learning features, employed tools, and project sizes. Their findings suggest that, while many programs included larger projects and collaborative components, traditional classroom settings predominantly governed SA education. Although the focus was not on student-reported outcomes but on instructor facilitation, it accentuates the necessity of active learning, underscoring collaborative learning and instructor support as fundamental elements.

Pantoja Yépez, et al. [15] systematically reviews 56 studies from 2011-2021 and explores effective teaching strategies, skills development and alignment between SA education and industry needs. The main contribution of this work was identifying the methods used in educating software architecture students that were aligned with the software industry's needs. Additionally, the study compared curriculum contents in software development and architecture courses and identified recurring topics such as architecture patterns, quality attributes, and architectural views. Sets of skills that students of software architecture should develop were also identified, such as leadership and negotiation. The challenges in software architecture training were also discussed, such as instructors' lack of experience in actual projects, the abstract and fuzzy nature of software architecture, and the difficulty of involving clients and industry experts.

Likewise, the study [16] provides a comprehensive mapping of software engineering education, in which the study found that Agile methodologies dominate software engineering education, with Project-Based Learning (PBL) being the most widely used teaching strategy. While software engineering education increasingly incorporates industry trends, collaboration between academia and industry is still limited, leading to gaps in real-world experience for students. The study also recognises the lack of standardisation in the assessment frameworks, making it challenging to evaluate student competencies effectively.

After analysing 60 primary studies, the study [17] on a comprehensive systematic mapping of software engineering competencies identified two primary focus areas: personal and organisational competencies. The authors further categorised 49 essential competencies in 11 themes, revealing the dominance of soft over hard skills and highlighting gaps in competencies required for industry readiness. The paper also emphasised the need for updated integrated approaches to assessing and developing competencies. The study thus calls for further investigation into evolving roles, industry-aligned competencies, and stakeholder-inclusive frameworks to ensure future software professionals are well-equipped.

In 2022, Oliveira et al. [18] conducted a systematic mapping study of SA Education, analysing 50 studies published between 1992 and 2020. They explored how software architecture (SA) has been taught, focusing on primary subjects such as quality attributes, ATAM, and architectural styles. The study also investigated the cognitive levels targeted by learning goals. While they noted the incorporation of various techniques, such as game-based learning, project-based learning, and collaborative tools, the review identified a lack of exploration into contemporary paradigms in architecture, like microservices. Their study offers a current and comprehensive overview of trends in software architecture education. However,

further research is needed to understand how teaching practices align with evolving industry demands and student experiences.

Although several systematic reviews have been conducted, to our knowledge, a comprehensive review specifically evaluating how SA education research has been synthesised from the students' perspective is still lacking. This review aims to address this gap by exploring how learners engage with, benefit from, and encounter challenges in SA educational settings, offering a novel perspective that centres student experiences in pedagogical analysis.

4. RESEARCH METHODOLOGY

The study employs a systematic literature review (SLR) methodology, a structured and rigorous approach to identifying, evaluating, and synthesising existing research on software architecture teaching. The SLR follows the guidelines proposed by Kitchenham and Charters [19] for systematic reviews in software engineering with three fundamental phases: Planning, Execution and Review. In this paper, the planning phase is carried out to define research objectives and search protocol. The execution phase is implemented during the search for the article for review, and the review phase is presented separately in the Results section. This method ensures transparency and reproducibility and minimises bias in the selection and analysis of studies. The following are the methods we followed for this study.

4.1. Research Questions

The research questions (RQ) for this systematic review were framed using the SPIDER framework [19] to address the gaps and challenges in SA education. The goal was to explore the current state of teaching strategies, practical experiences or hands-on skills, assessment methods for teaching SA, outcomes from the teaching strategies, and challenges faced by educators and students in higher education settings. Despite diverse training strategies in teaching experiences and several mapping studies conducted in SA education, research gaps exist in the lack of a critical analysis of the interrelationship among various SA teaching strategies, assessment approaches, outcome evaluations, as well as challenges faced by multiple stakeholders. In addition, the limited attention paid to students' integrated understanding of teaching strategies and learning challenges in existing mapping studies further enhances the necessity of research.

TABLE 1: APPLICATION OF SPIDER FRAMEWORK IN THE STUDY

S- Sample	Relevant Articles, Secondary Data
P – Phenomenon of Interest	Teaching Strategies, Outcome, Assessment approaches, industrial experiences and challenges
D – Design	Systematic Review
E – Evaluation	Synthesise teaching themes, assessment approaches, categories and evaluate outcomes based on Bloom's Taxonomy and challenges faced by multiple stakeholders. Critically analysing interrelationships among SA teaching themes, assessment approaches, outcomes and challenges, based on which recommendations are provided.
R – Research Type	Qualitative, Mixed-Method Approach

The research questions formulated for this study are as follows:

RQ1: What are the different teaching strategies used in software architecture education?

RQ2: How are students provided hands-on practical experiences related to software architecture teaching?

RQ3: What are the various assessment methods used to assess their students?

RQ4: What are the outcomes of software architecture teaching strategies?

RQ5: What challenges do students and educators face with the software architecture teaching approach?

4.2. Search Strategy

A detailed search strategy was designed to identify relevant database sources. Ten databases were initially identified to search for relevant studies on software architecture teaching. These databases comprise ACM Digital Library, DBLP, ERIC, IEEE, Google Scholar, Science Direct, Scopus, Springer, and Web of Science. However, ResearchGate had to be removed because it allows authors to update their published and unpublished works, which is not authentic for this systematic review.

After identifying the databases, search keywords and search strings were derived from research questions. A common search string was developed for all reviewers in their respective database using keywords and Boolean operatorss. We have defined the following search strings for our research questions and used Boolean operators (AND, OR) to combine various concepts [20]. We used truncation (*) for variations of words that are useful for finding singular and plural forms of words or different endings {Leeds, N.A. #208}.

("software architecture" OR "software design" OR "architectural patterns" OR "system design") AND ("teaching methods" OR "educational strategies" OR "pedagogical approaches" OR "active learning" OR "problem-based learning" OR "case studies" OR "project-based learning" OR "practical experience" OR "hands-on training" OR "assessment techniques" OR "evaluation methods")

4.3. Inclusion and Exclusion Criteria

The criteria were set to select the relevant studies for this report.

4.3.1. Inclusion Criteria

- Studies must be from the perspectives of students or educators in the domain of software architecture.
- The phenomenon of interest must be the teaching of software architecture, the challenges of teaching it, or the results of that teaching.
- Studies must provide an evaluation of teaching strategies used, challenges faced by educators, student performance, or industry alignment in software architecture.
- Studies must be within the specified research type.
- Peer-reviewed articles, conference papers, case studies, and book chapters.
- Articles published between 2010 and 2024.

4.3.2. Exclusion Criteria

- Studies that only discuss software architecture without addressing teaching strategies.
- Studies that focus on general software engineering education without specific relevance to architecture.
- Studies that are not available in full text.
- Studies focused on industry training rather than academic teaching.
- Non-English publications.
- Opinion pieces or non-peer-reviewed articles.
- Articles published before 2010 and after 2024.

4.4. Execution Stage

A total of 58,027 studies were identified through database searches and grey literature. Conference proceedings, workshop papers, and technical reports were included to capture recent and practical insights. We used MS Excel to do a basic screening of articles using inclusion and exclusion criteria, which selected 1,322 studies. The study used a review tool called Covidence to screen out duplicates and get consensus from different authors to avoid biases. Titles and abstracts were screened to remove irrelevant studies, resulting in 230 potentially relevant studies. The remaining 122 studies were reviewed in full text to assess their relevance based on the inclusion and exclusion criteria. This resulted in 82 studies being selected for data extraction. Data was extracted using a standardised form, including information on teaching strategies, practical experiences, assessment methods, outcomes, and challenges.

The selected studies were evaluated for quality using a checklist based on criteria such as research design, validity, and relevance to the research questions. After a thorough evaluation for quality, a thematic synthesis approach was used to categorise findings into themes aligned with the research questions. For example, teaching strategies were grouped into project-based learning, case studies, and flipped classrooms.

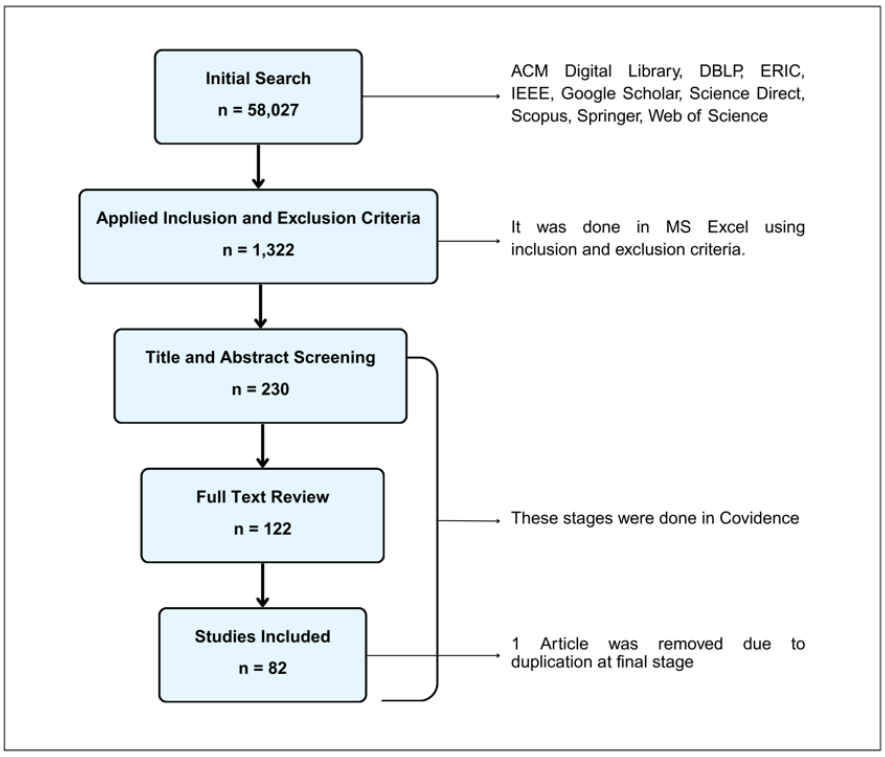


Fig. 1: PRISMA diagram of Study Selection for Systematic Review

The systematic review methodology, with its rigorous search strategy, selection criteria, and execution process, ensures that the findings are reliable and relevant to academia and industry. This approach provides a solid foundation for identifying best practices and areas for improvement in software architecture teaching.

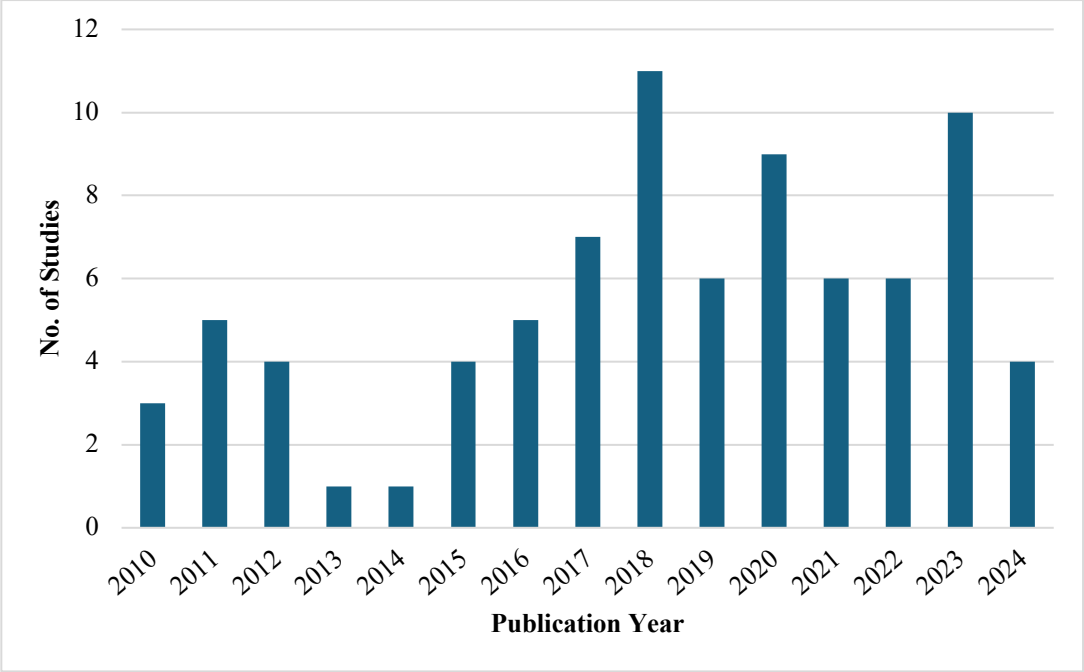


Figure 2: Number of studies by publication year

5. RESULTS ANALYSIS

This section will discuss the findings of each research question.

5.1. RQ1: What are the different teaching strategies used in software architecture education?

Through inductive thematic analysis of 64 papers reviewed, the study developed 10 themes to address diverse yet complementary pedagogical needs, reflecting the evolving landscape of SA education. Consequently, thematic categorisation helped synthesise the literature meaningfully while respecting the nuanced diversity of teaching strategies used across various institutions, course levels, and educational frameworks. Additionally, these 10 themes provide a framework to assess how effectively current practices address architectural education's technical and professional components. Therefore, it is essential to note that some reviewed studies incorporate multiple teaching strategies that may overlap with the identified themes.

TABLE 2: NUMBER OF TEACHING STRATEGIES IN DIFFERENT THEMES

Themes	Number of studies	Reference
Project-Based Learning Approach	14	[22], [23], [24], [25], [26], [5], [27], [10], [28], [29], [30], [31], [32], [33]
Real World-Based Learning	12	[34], [35], [6], [36], [37], [38], [28], [29], [39], [40], [8], [41]
Collaborative Learning	10	[42], [26], [5], [43], [28], [44], [30], [45], [46], [47]
Game-Based Learning	9	[48], [20], [3], [28], [49], [50], [51], [52], [53]
Personalised Learning	8	[54], [55], [56], [57], [58], [38], [59], [60]
Abstraction and Modelling	8	[61], [62], [63], [9], [64], [5], [65], [66]
Traditional Approach	6	[35], [11], [29], [45], [49], [47]
Blended Learning	5	[67], [35], [7], [49], [1]
Software System Learning	5	[68], [69], [70], [71], [72]
Agile-Based Learning	4	[73], [74], [75], [76]

This study identified three prominent themes, namely Project-based learning, Real-world-based learning and Collaborative learning, with a substantial number of papers reporting strategies categorised under these themes as seen in *Table 2*. The Project-based learning approach theme with the highest teaching strategies (n=14) emphasises sustained, artefact-driven assignments that replicate the complexity and expectations of real-world software development. The project-based learning strategy under this theme, students work on open-source or real-world projects, fostering skills in analysis, design, and evaluation of architectural solutions [2] while promoting teamwork [51]. Students face challenges that enhance their problem-solving abilities and practical skills through medium complexity projects [56,52], often culminating in a group-based project exam [36]. This strategy supports architecture-centric pedagogy via research-driven, collaborative learning [75] aligned with industry demands, especially in mobile and embedded systems development. It addresses the challenges of teaching embedded software to students with limited electronics backgrounds by integrating hands-on projects with open-source tools [35].

Other teaching strategies, such as product-based learning, require students to complete open-ended assignments that resemble real-world deliverables, which are rigorously graded [30]. Platform-based lightweight projects introduce cloud environments like Google App Engine, offering early exposure to scalable architecture practices [17]. Mobile project-based learning creates an interactive and

collaborative learning environment [39]. End-to-end project-based teaching strategy integrates business, architecture, and process instruction using real-world frameworks, flipped delivery, and active learning tools (games, projects), encouraging strategic and holistic software design thinking [16]. Projects are developed iteratively with peer feedback, quality attribute analysis, and real-client-inspired problem-solving [15].

Real-world-based learning (n=12), the reviewed papers' teaching strategies focus extensively on practical, authentic scenarios closely mirroring actual industry practices. The teaching strategies in these themes, such as Problem-Based Learning, allow adult learners to focus on challenges and the strategy's effectiveness without any preset goals and expected outcomes [42]. Students address architectural problems, and instructors play minimal roles in this strategy [29]. It is supported by Neural Pathway Based Learning (NPL) strategies, which combine diverse cognitive stimulation such as quizzes, animations, games, puzzles and repetition [43]. The Architectural Kata structure enhances real-world simulation by guiding students through design challenges, peer review, and voting phases [57]. It also serves as a learning activity for students to hone the skills required for their assignments, with the teacher functioning as a moderator [52]. Case-Based Learning introduces bullet and mini cases to develop analytical thinking using real-world scenarios [59]. This strategy involves a real-life scenario that allows students to analyse, apply taught concepts and mitigate potential adjustments [49]. It also has workshop-based courses with standard structures such as preset goals and expected outcomes [42]. Microservices-based problem-solving [24] involves collaborative tasks aligned with modern software design practices. Learning by doing allows students to transform 2D shapes into 3D models through hands-on experience [50]. Learners perform one or a few architectural activities based on knowledge acquired before or during the experience [51]. Moreover, Interdisciplinary Learning incorporates industry collaboration, allowing students to experience diverse perspectives on designs [63], while Role-Play Learning fosters an appreciation for architectural compromises through professional role-playing [64]. All these principles promote a holistic learning experience that fully equips students with skills relevant to real-world applications.

Within the Collaborative Learning theme (n=10), teaching strategies emphasise peer interaction, co-construction of knowledge, and shared decision-making. Unlike other themes focusing on context (e.g., real-world scenarios or project outputs), collaborative learning emphasises the learning process through structured teamwork and reflective peer engagement. Strategies such as team-based learning centre on teamwork and peer evaluation through group activities [56], while collaborative decision-making involves students taking on various architectural roles- senior, junior, or cognitive architects- to simulate real-world group dynamics and enhance critical design thinking for better quality decisions and architectures [35,40]. The Computer-supported Collaborative Learning (CSCL) strategy further facilitates interaction in virtual environments, supporting distributed teams in solving complex tasks [6]. Models like the Collaborative Studio, Transdisciplinary Encounter, and Real Team Collaboration (where students alternate between individual and group work) promote deeper critical thinking and self-regulated learning [53]. Open-source collaboration through platforms like GitHub introduces layered team dynamics based on varying skill levels, mirroring real development ecosystems [58]. Learning with real clients or industry experts and apprenticeship-style mentorship adds authenticity to the educational experience by exposing students to real-world constraints, professional expectations, and industry-grade

decision-making processes [30,66]. These approaches collectively distinguish collaborative learning as a critical pedagogical strategy that cultivates architectural thinking through active peer engagement, distributed decision-making, and reflective role-based participation in team-oriented environments.

Under the Game-Based Learning theme (n=9), selected studies demonstrate how interactive, game-driven strategies enhance student understanding of software architecture concepts. These strategies incorporate structured gameplay, simulations, and competitive elements to replicate architectural decision-making engagingly. Educational games DecidArch [61] and DecidArch v2 [78] that simulate collaborative architectural planning can take on roles, negotiate trade-offs and reconcile stakeholder concerns. Other implementations involve card-based games and role-playing, recreating industry constraints, enabling students to examine design principles and system interactions in a safe and motivating context [69,77]. The studies also include using tools like Kahoot! to test conceptual understanding in flipped classroom settings, turning assessments into interactive, formative learning experiences [60]. Across the strategies, game-based learning was found to enhance conceptual understanding, motivation, and peer collaboration by embedding students in simulated architectural processes that reflect real-world scenarios. In contrast to real-world-based learning, which exposes students to realistic professional scenarios, game-based learning connects students by providing an interactive, structured environment for learning through play, ultimately making complex concepts easier to grasp and more memorable.

Comparatively, the Personalised Learning theme was reported only in (n=8) studies, emphasising teaching strategies that promote student autonomy, self-regulation, and custom learning experiences in SA. Suh's strategies include a pattern-based approach that offers flexible modular frameworks for exploring architectural concepts tailored to their unique learning requirements [31] and the Student Ownership of Learning (SOL) model which promotes self-directed learning by encouraging students to utilise their experiences and create personal meaning, for deeper engagement and conceptual depth [34]. Active experimentation is further enhanced through hands-on design and experiential learning techniques [50,54]. This approach has been applied to adaptive learning, aligning content delivery with individual student profiles, thereby improving performance and accessibility [38]. Additionally, scaffolded requirements-to-architecture strategy aids students through complexities from requirements analysis to architectural design, building their confidence incrementally [22].

Likewise, the Abstraction and modelling theme(n=8), the reviewed studies focus on cultivating architectural thinking through formal modelling techniques, system representation, and abstraction-driven instruction. Strategies such as model-driven teaching [21] and model-driven [25] engineering (MDE) enable students to structured abstraction practices aligned with professional architectural modelling workflows. These approaches develop conceptual clarity by focusing on system-level reasoning, transformation logic, and layered design. Other strategies, like abstract modelling and its design pattern decomposition, guide students in analysing functional components, enhance their understanding of system design, and prepare them for real-world software development opportunities [32]. Similarly, early design-centric education introduces structured modelling tools early in the curriculum to foster architectural thinking from foundational programming stages [80]. Queueing models [26] and simulation-based instruction [46] bring performance evaluation, visualisation, and system behaviour modelling into the classroom, strengthening analytical skills.

Only (n=6) was reported under the Traditional Approach theme, where the teaching strategies were characterised by traditional, instructor-led methods focusing on structured content delivery. These include classroom-based lectures where educators share domain knowledge directly with students [29], conceptual explanations provided during lab sessions using slides, web pages, and code extracts [60], and guest lectures designed to familiarise students with current industry trends and challenges [67]. Developing structured teaching manuals that outline learning objectives, desired outcomes, and methodologies supports consistent instruction across various topics [52]. These strategies depend significantly on students' ongoing engagement and receptiveness. When implemented effectively, teaching strategies like constructive alignment [45], where all learning activities connect directly to desired outcomes, can promote a deeper understanding of concepts.

The blended learning theme (n=5) signifies a transformation in how instruction is delivered in contemporary education. Integrating asynchronous digital tools with real-time classroom engagement enhances accessibility through Zoom, MS Teams, and WhatsApp platforms. In contrast, asynchronous techniques include recorded lectures and additional resources from platforms such as YouTube, NPTEL, SWAYAM, Edx, and Coursera [29,55]. This structure enables flipped classroom models [14, 29, 60, 62], where students engage with video content independently and apply their knowledge during face-to-face sessions focused on design analysis and quality attributes. Additionally, the use of Massive Open Online Courses (MOOCS) and Small Private Online Courses (SPOCS) [29] supports self-paced exploration and broad content access. Some strategies also incorporate intelligent tutoring systems to personalise instruction based on student progress.

The Software System Learning theme(n=5) is grounded in applied understanding, focusing on how architectural decisions play out within integrated, interactive, and operational software environments. The reviewed studies emphasise teaching strategies that help students understand software architecture through direct interaction with full-scale systems and systemic behaviours. Strategies such as the Usability-Supporting Architecture Patterns (USAPS) approach teach students to identify and address usability concerns early in the architecture of real software systems, highlighting human-centred design in system-wide contexts [37]. Similarly, the Software Product Line Architecture strategy introduces students to architecture as a strategic asset, teaching them to manage variation and reuse across families of products [44]. AR/VR and immersive visualisation tools enhance this systems-level perspective by allowing students to explore architectural constructs in 3D or augmented environments [82]. Tools like Archinotes support project coordination and provide real-time performance insights, helping students and instructors monitor progress and adapt learning activities [68].

Notably, only (n=4) of the reviewed papers addressed the Agile-based learning theme. Strategies under this theme mirror development methodologies commonly used in industry [7, 9] and are also a popular empirical process control model to manage complex activities [41]. Incorporating sprints, scrums, and retrospective feedback into instructional design fosters iterative thinking and a culture of continuous improvement. This strategy encourages students' adaptability to change, teamwork, and transparency in processes, all fundamental principles of contemporary SA [48]. When embedded within project-based learning, Agile-based strategies enhance educational experience by aligning academic tasks with realistic development workflows and encouraging industry-relevant tools and practices in student-generated solutions [7, 9].

5.2. RQ2: How are students provided hands-on practical experiences related to software architecture teaching?

Of the 82 studies reviewed, 73 were found relevant to RQ2 with distinct thematic approaches. These studies are grouped into seven overarching themes to categorise various strategies from the systematic review of software architecture-related studies. These strategies are employed in software architecture pedagogy to foster practical skills and experiences. The findings of this RQ supplement the other questions in fulfilling the objectives of finding how critical hands-on practical experiences are to the teaching and learning of software architecture. While *Table 3* shows the list of studies included in each thematic category, here is the detailed analysis of RQ2's findings.

TABLE 3: NUMBER OF STUDIES IN EACH THEME

Theme	Number of Studies	Reference
Project-based Practical Experiences	26	[3, 11, 24, 30, 34, 36, 44, 45, 49, 55-62, 64, 67, 70, 73-75, 77-79]
Real-world-based Experiences	25	[5, 6, 20, 23, 26, 28, 29, 32, 35, 38, 41, 42, 47, 50, 52, 54, 65, 68, 71, 76, 80-84]
Tool and Platform-based Skills	8	[25, 33, 37, 48, 66, 69, 72]
Collaboration	6	[1, 9, 10, 22, 27, 85]
Analytical Skills	8	[31, 37, 39, 43, 53, 63, 86, 87]

The foundational approaches are rooted in experiential learning, dominated by the project-based practical experiences (n=26) through diverse project formats such as Agile and capstone projects [73], iterative design assignments through flipped classroom models [67], interactive decision-making games [3], specific modelling tasks [64], and gamified exercises [57]. Additionally, the closely related real-world-based experiences theme (n=25) emphasises a prevalent strategy where students engage with projects to mirror authentic industry challenges. Findings indicate that educators utilise diverse methods, from developing complete software systems like a 'Theatre Management System' [80] or mobile robots to undertaking specific, contextualised tasks like usability problem-solving [68] and managing client interactions [76]. While these two themes underscore a strong commitment to practical application through project work, they are complemented by themes with a more targeted focus, albeit less frequently represented in the studies. Tool and platform-based skills (n=7), for instance, contrast with the broader project themes by concentrating explicitly on equipping students with proficiency in specific technologies crucial to the field. The studies reported employing various tasks such as version control systems, cloud platforms [48], modelling tools [66], and code generation tools [33] that enable students to gain practical familiarity with the technological landscape of the software architecture.

Supplementing project execution and technical skills, the collaboration theme (n=6) highlights strategies designed to cultivate essential teamwork and communication abilities, reflecting the inherently collaborative nature of professional practice. Studies used tools such as structuring students into start-up-like teams, facilitating collaboration with real customers [22], utilising shared tools for version control and planning [85], and designing group projects [9]. Furthermore, the development of crucial cognitive capabilities is addressed through the analytical skills theme (n=8), focusing on practical experiences that enhance critical thinking, evaluation, and justification of architectural decisions. The studies employed

tasks involving the comprehension of architectural documentation, analytical models [63], engagement with decision-modelling tools [43], analysis of realistic problems, evaluation of existing systems [31], and decision-making games [53].

5.3. RQ3: What are the various assessment methods used to assess their students?

The findings of the assessment methods from 60 studies from 82 reviewed articles have been categorised into four themes based on their nature and purpose of the assessment. The four thematic categorisations emerged from the three foundational concepts of assessments: Assessment for Learning (AfL), Assessment of Learning (AoL) and Assessment as Learning (AaL). Assessment for Learning [88] involves the information for improved performance and further actions to improve learning. Assessment of Learning [89] is judging the performance and measuring outcomes after the learning activity. Assessment as Learning [90] involves students in self-assessment and self-directed learning to enhance the learning process.

This paper relates formative assessment strategies, summative assessment practices, and experiential assessment techniques to AfL. The fourth theme, Technology Assessment Tools, supplements the other three assessment methods since it is incorporated into online tools or platforms for assessment purposes that can provide instant feedback and grading.

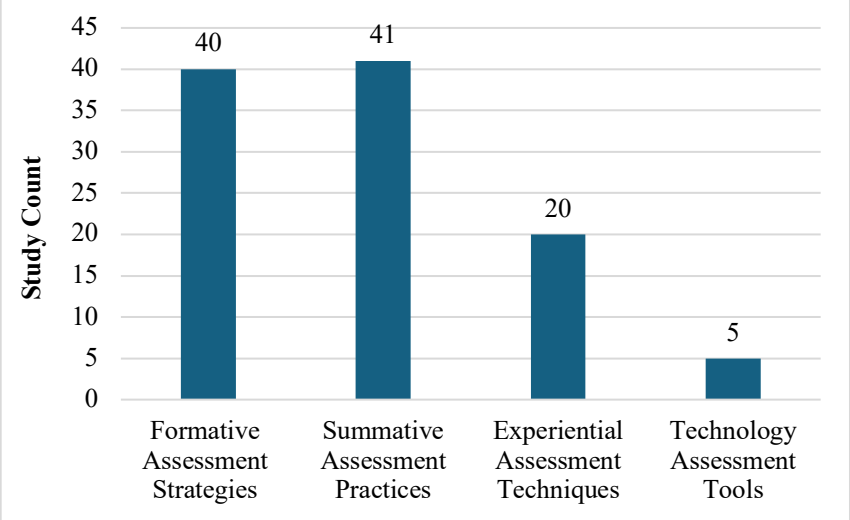


Figure 3: A chart showing the frequency of occurrence of studies in each assessment theme

Figure 3 shows the frequency of assessment methods by the themes extracted from the reviewed articles, in which summative and formative assessment strategies were reported to be widely used. Most papers mentioned using more than two assessment approaches to assess students' learning, while some papers did not mention the use of assessment methods at all. This study found that five different technology tools can enhance the assessment method.

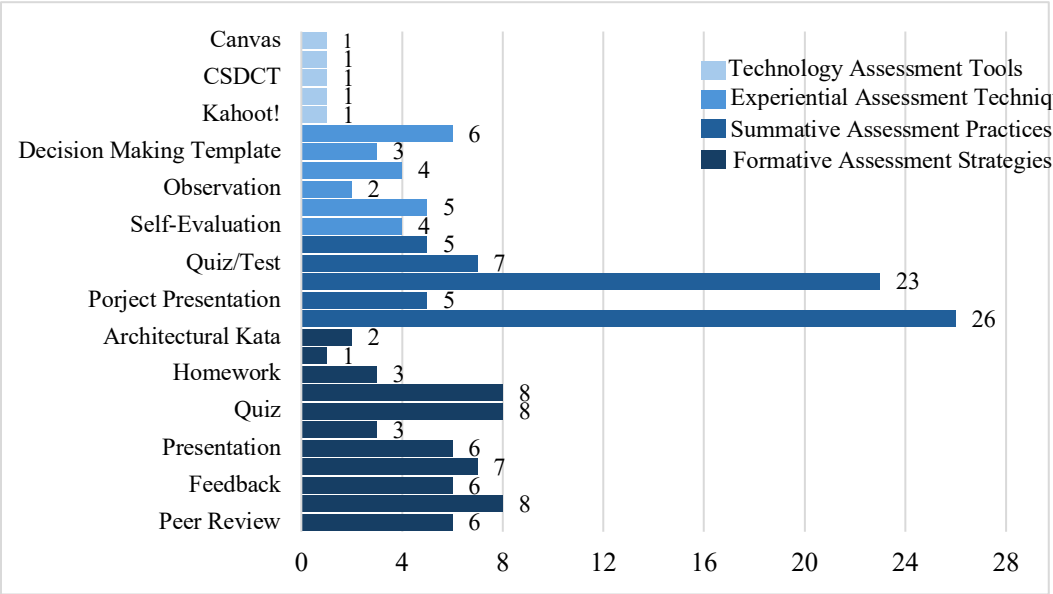


Figure 4: Assessment methods by theme

For the formative assessment, the frequently used methods were discussion, exercises and quizzes, while the least used were inter-group evaluation (n=1) and Architectural Kata (n=2). It has been found that they implement the pre-class quiz before coming to the class to check their level of understanding of specific topics, as well as a pre-assessment of the course or content. They provided the reading materials and let the students attempt the quiz so that the educators could assess them. There are also other formative assessment techniques such as presentation (n=6) where they do oral presentation on midterm progress, the class participation (n=7) assessed through engagement and participation log, give peer review (n=6) by the students for collaborative learning and improvements, and feedback (n=6) are usually given by the assessors (teachers or other stakeholders).

The examinations and project form of assessment were widely used from the summative assessment practices, the top two assessment tools used in the reviewed articles. Examination form of assessment is a widely used assessment strategy in teaching software architecture, with 23 of the reviewed articles mentioning it. They conducted the examinations in oral and written form at the beginning and end of the semester, as well as closed-book [34] and group-based project exams [27]. The project-based assessment (n=26) is widely used as a project, such as a capstone project, writing a book chapter, practical projects, a real-world project problem and even an individual project. They were evaluated at the end of the session, and the project presentation (n=5) was also integrated to present their findings. Other methods, such as quizzes or tests (n=7) and assignments (n=5), were also used for grading the learner's performance. Quizzes, tests and assignments were assessed in the classroom, which took place after the lesson for a short duration.

The reflective writing journal or portfolio (n=6) was frequently used for the experiential assessment strategies, followed by self-reflection (n=5). In the portfolio, the learners were required to maintain the documentation of the individual contributions, process, methodology, decision-making process and other related questions. At the end of each session or sprint, the reflection is done on the team project and their learning journey. Self-evaluation was done through self-evaluation at the end of each session [55],

individual assessment at each sprint [75] and self-evaluation survey [76]. The other assessment techniques mentioned in the reviewed articles were as well as observation (n=2), prototyping (n=4) and decision-making template (n=3), which helped the learners to log and reflect on their reasoning for their decisions.

Six technology-based assessment tools were used to enhance the assessment. They are platforms used to supplement the summative and formative assessment strategies. Canvas [58] and Kahoot! [49] were used to conduct quizzes, EasyChair [45] for peer review, and CSDCT [41] for instant feedback. 3D visualisation technologies [72] with AR and VR help to bridge the gap between theory and practice.

The detailed assessment techniques in the papers under each theme are mentioned in *Table 4.*

TABLE 4: NUMBER OF STUDIES UNDER EACH ASSESSMENT THEME

Theme	Assessment Methods	Reference	Total Number of Studies
Formative Assessment Strategies	Peer Review, Discussion, Participation, Presentation, Weekly Assignment, Quiz, Exercise, Homework, Inter-Group Evaluation, Architectural Kata	[1, 5-7, 9, 11, 22-24, 26, 29, 31, 32, 34, 36-39, 43, 45, 48, 49, 52, 55, 57, 59, 61, 63-65, 67, 76, 77, 79-82, 84, 85, 87]	40
Summative Assessment Practices	Project, Project Presentation, Examinations, Quiz/Test, Assignments,	[4, 5, 7, 9, 20, 22-27, 31, 32, 34, 36, 38, 42, 45, 48, 49, 55, 57-59, 62-65, 71, 74-76, 78, 80-85, 87, 91]	41
Experiential Assessment Techniques	Self-Evaluation, Reflection, Observation, Prototype, Decision Taking Template, Portfolio,	[6, 11, 22, 23, 33, 34, 38, 45, 48, 50, 53, 55, 65, 67, 70, 73, 75, 76, 85, 86]	20
Technology Assessment Tools	Kahoot!, EasyChair, Client-Server Design Critic Tool, 3D Visualisation Tools, Canvas.	[41, 45, 49, 58, 72]	5

5.4. RQ4: What are the outcomes of software architecture teaching strategies?

A total of 64 articles were analysed, and in this study, we have considered the learning outcome as the measurable statement that the student has learned or gained from the studies conducted. So, we have adopted Bloom's taxonomy [92] for better comprehension and assessment. Bloom's taxonomy is a comprehensive framework that will help identify and classify studies according to levels of complexity and specificity, as seen in *Figure 5*.

The highest level of Bloom's taxonomy is the Create, and only n=3 studies were reported under this level. Similarly, n=3 studies were classified under Evaluate, followed by n=4 papers for the Analysis. Most papers (n=29) were classified under Apply, followed by studies under Understand (n=27), as displayed in *Figure 6*, while Remember had none, which can be positively perceived. The categorisation thus reveals that all the papers with the identified software architecture teaching strategies contribute to the development of foundational and practical competencies among students, with varying degrees of emphasis depending on their alignment with specific Bloom's Taxonomy levels.

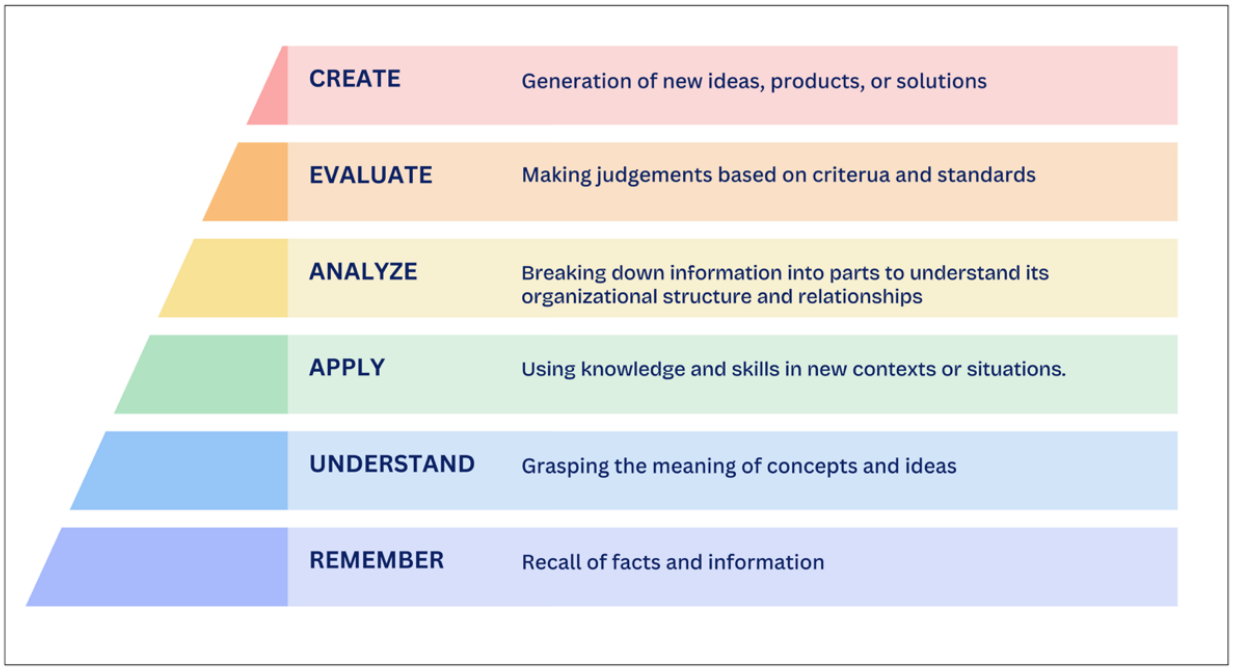


Figure 5: Learning outcomes aligned with Bloom's Taxonomy

The reported learning outcome aligned with the Create level of Bloom's taxonomy included statements such as "student created more complex software architecture [20]" and "solving real-world problems [20]." These outcomes suggest that such strategies targeting these levels encourage innovation and bridge the gap between theoretical understanding and practical application.

At the Evaluate level, the reported learning outcomes included "strongly favouring a strategy [23]," "Enhanced reasoning skills [50]," and "Developed critical thinking [1]." Such outcomes reflect the student's ability to make informed judgements, assess complex scenarios, and justify preferences, which are the key components of the level.

For the Analyse, reported outcomes were "Improved ability to reason and analyse [67]", "Enhance analytical skills [39]" "Learnt to make decisions and prioritising[39]." These highlight the role of strategies in fostering students' capacity to deconstruct problems, identify and make informed choices.

The Apply level was the most represented among the studies, with commonly reported learning outcomes including "Increased productivity [42]," "Improved Knowledge/learning [26, 35, 56, 74]," "Positive performance [48, 61]," "Adequately demonstrating [24]," "Applying understanding across subject area [55]," "Fosters early engagement [57]," "Enhanced motivation and engagement [10]," "Application of software architecture principles [6]," "Improved confidence [38]." Collectively, these findings highlight the effectiveness of application-focused strategies in facilitating the transfer of theoretical knowledge into a practical, real-world context.

Lastly, for the Understand level, commonly reported outcomes included "Improved understanding [47, 54, 71]," "Perceived value [73]," "Complements teaching [3]," "Demonstrate proficiency and improved

student engagement [58, 64],” “Gained Knowledge [29],” and “higher satisfaction [60].” These outcomes encourage the importance of foundational comprehension in software architecture education, promoting conceptual clarity and contributing to a meaningful learning experience.

TABLE 5: CATEGORISATION OF TEACHING STRATEGIES ACCORDING TO BLOOM'S TAXONOMY.

Blooms Level of Taxonomy	No of studies	Categories
Create	3	Game based learning [20], Problem based learning [34, 36] [20, 34, 36].
Evaluate	3	Project based learning [23], Experiential learning [65], Flipped classroom [1], Game based learning [50] [23] [65] [1] [50].
Analyse	4	Flipped classroom [67], Design centric learning [87], Case-based learning [39]. [39, 67, 87].
Apply	29	Project based learning [10, 22, 24, 25, 30, 31], Agile [74, 76], Game based learning [48, 50], Model Based Learning [11, 56, 61, 63, 68], Case based learning [6, 37, 41], Experiential learning [26, 38, 57, 59], Collaborative learning [42, 45, 54], Design centric learning [87],Others [7, 55].
Understand	25	Project based learning [27, 30, 32, 54], Agile [75], Game based learning [3, 53], Model Based Learning [52, 62, 64, 71], Experiential learning [8, 58, 60], Collaborative learning [5, 43, 44, 54, 73], Design centric learning [66, 70], Flipped classroom [1, 49], Others [11, 44, 47].
Remember	N/A	N/A

Building on the synthesis of learning outcomes by Bloom's taxonomy levels, a closer examination of individual teaching strategies as shown in *Table 5* reveals that Problem Based Learning with the highest levels of Blooms taxonomy (Create n=2) followed by Game-Based Learning, appearing in both the Create (n=1) and Evaluate (n=1) categories, highlighting its potential to foster advanced cognitive outcomes. In contrast, Project-Based Learning demonstrated broad applicability across the taxonomy, making it one of the most versatile and widely adopted strategies, as seen in *Figure 6*. It was most prevalent in the Apply level (n=7), followed by Understand (n=5), and appeared at the Evaluate level (n=1), indicating its widespread use in promoting both foundational and higher-order learning. Similarly, the Flipped Classroom approach was observed across multiple levels, with instances at Evaluate (n=1), Analyse (n=1), and Understand (n=2), suggesting its flexibility in addressing varied cognitive demands.

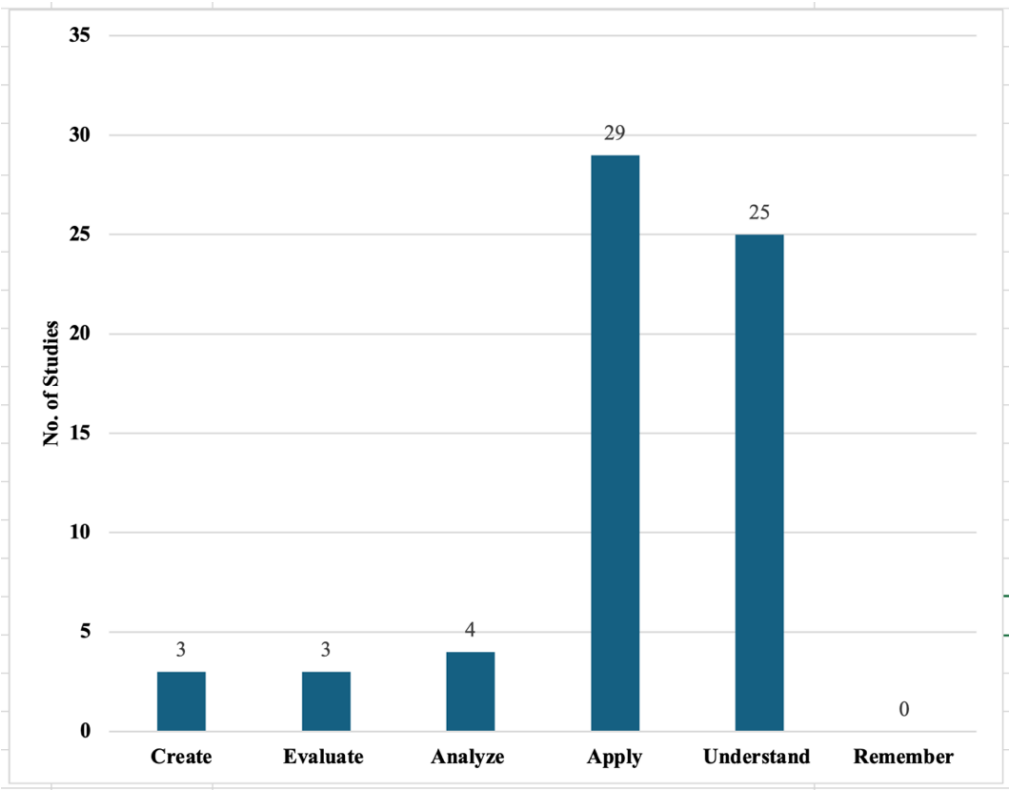
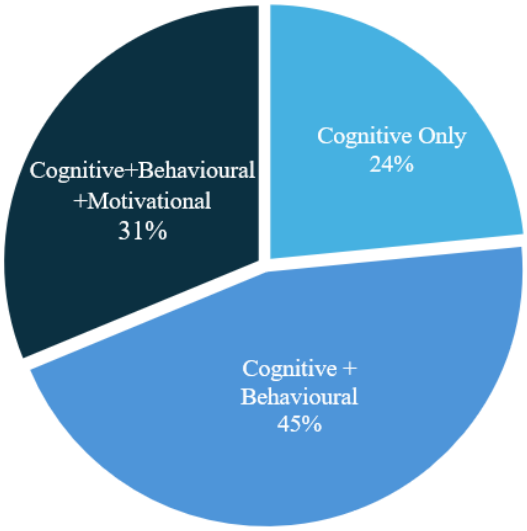


Figure 6: Distribution of studies across levels of Bloom's Taxonomy

Model-Based Learning showed an even distribution across Apply (n=5) and Understand (n=5), indicating that students engaged with this strategy effectively bridged conceptual comprehension and practical application. Agile methodologies were predominantly observed at the lower cognitive levels, Understand (n=2) and Apply (n=2), paralleling the trend seen with Collaborative Learning, which was also concentrated at Apply (n=4) and Understand (n=4). These findings suggest that while some strategies are more specialised in targeting higher-order thinking, others remain essential for reinforcing foundational competencies."

In addition to classification by Bloom's Taxonomy, the nature of the learning outcome was also studied through the lens of outcome types, namely cognitive, behavioural and motivational dimensions. This categorisation provides added nuance to understand how students engaged with the learning process across different levels of cognitive complexity. Depending on the specific outcome statement reported in each study, learning outcomes were classified as belonging to one or multiple dimensions as depicted in *Figure 5*. This layered analysis allowed a deeper interpretation of what students learned and how they responded to the instructional strategies employed."



Cognitive outcomes *refer to the development of knowledge, intellectual skills, and mental processes.* ***Behavioural outcomes*** *reflect observable actions or performance that result from learning.* ***Motivational outcomes*** *relate to changes in students' attitudes, engagement, confidence, and willingness to participate in learning.*

Figure 7: Distribution of learning outcomes by cognitive level.

The studies reveal that most teaching strategies in software architecture education are associated with multi-dimensional learning outcomes beyond just cognitive gains. Specifically, Cognitive & Behavioural outcomes with 45 per cent as shown in *Figure* 7: Distribution of learning outcomes by cognitive level.

7, highlighting a strong emphasis on practical engagement that fosters skill development in decision-making, teamwork, and system design. Thirty-one per cent of the total studies had outcomes across all three domains, Cognitive, Behavioural, and Motivational suggesting the effectiveness of integrated and immersive approaches, these strategies not only support conceptual understanding but also encourage learner motivation and active participation, both of which are essential for sustained engagement and professional readiness in software architecture roles. In contrast, only 24 per cent of the studies focused solely on Cognitive outcomes.

TABLE 6: NUMBER OF STUDIES FOCUSED ON MULTI-DIMENSIONAL LEARNING OUTCOMES

Outcome Type	Total studies	Reference
Cognitive Only	15	[3, 8, 9, 22, 23, 33, 50, 54, 56, 59, 64, 66, 67, 70, 73]
Cognitive + Behavioural	29	[5-7, 11, 20, 24, 25, 29-32, 37, 41, 47, 49, 52, 53, 57, 58, 60, 61, 63, 65, 69, 71, 74-76, 87]
Cognitive + Behavioural + Motivational	20	[1, 10, 26-28, 30, 34, 36, 38, 39, 42-45, 48, 55, 57, 62, 68, 76]

5.5. RQ5: What challenges do students and educators face with the software architecture teaching approach?

This question aims to identify various challenges faced by multiple stakeholders, based on which stakeholders could develop strategies to address these challenges. *Table 7* summarises the preliminary challenges encountered by different stakeholders in software architecture education, including general challenges faced by the industry and specific difficulties experienced by educators (e.g., university lecturers and tutors) and students. Each challenge is summarised in *Table 7.*

TABLE 7: CHALLENGES IN SOFTWARE ARCHITECTURE EDUCATION TO STAKEHOLDERS

No.	Challenge Name	Reference	Stakeholder Group	Number of Studies
1	Abstractness and Conceptual Difficulty	[2-6, 23, 24, 35, 42, 43, 47, 48, 61-65, 67, 73, 75, 78, 81, 93, 94]	Student	24
2	Time and Workload Constraints	[9-11, 20, 22, 30, 37, 38, 42, 48, 50, 53, 59, 72, 87]	Student	17
3	Instructor Workload & Expertise	[1, 2, 4, 11, 25, 33, 35, 44, 48, 49, 57-59, 62, 82, 85]	Educator	16
4	Technical and Tool-related Issues	[1, 7, 25, 32-34, 42, 49, 62, 65, 69, 70, 73, 80, 85]	General	15
5	Communication and Collaboration	[22, 29, 30, 34, 47, 53, 59, 68, 75, 77, 80-82, 91]	Student	14
6	Student Engagement and Motivation	[3, 8, 24, 27, 34, 43, 55, 58, 61, 69, 77, 81, 82, 93]	Student	14
7	Student Skill Gap	[1, 4, 5, 10, 22, 25, 33, 37, 56, 59, 73, 78, 80, 87]	Student	14
8	Scalability and Resource Constraints	[20, 30, 32, 35, 38, 42, 45, 46, 48, 55, 65, 70, 95]	General	13
9	Curriculum and Content Gaps	[6-8, 10, 36, 56, 72, 83, 95]	Educator	9
10	Assessment and Evaluation	[1, 2, 22, 26, 68, 82, 83, 93]	Educator	8
11	Lack of industry experts involved	[24, 35, 68, 83, 95]	Educator	5
12	Language and Cultural Barriers	[44]	Student	1
13	Plagiarism	[26]	Student	1

Abstractness and conceptual difficulty in understanding software architecture are the most frequently cited challenges (n=24). This challenge arises from the subject's inherent nature, where students are expected to understand and apply high-level architecture concepts, including architectural patterns, quality attributes, trade-offs, and systematic principles [94].

Some studies define software architecture as a "wicked problem" due to its open-ended nature, indicating no absolute right or wrong solution in designing software architecture. Still, architects must apply critical thinking, negotiation and decision justification to weigh trade-offs and find the most suitable solutions [2, 3, 94]. In addition, it becomes more complex when software architecture design transitions from programming-level details to high-level architectural thinking, resulting in cognitive overload for learners to struggle to visualise the architecture and its influence on the overall system [2, 47, 64]. Furthermore, design thinking, which is based on creative and user-centred approaches, and system

thinking, which ensures designs work coherently in real-world systems, are also key issues usually discussed in our selected studies. Despite its effectiveness in addressing cognitive overload challenges, students cannot fully understand and apply it appropriately due to its abstract nature [47, 64].

Relating to the above challenge of the subject's intricacy at a higher level, the second most discussed challenge (n=17) is time and workload constraints to finish the architectural project. This challenge refers to tight academic schedules, heavy workloads, and limited time to complete complex projects, which usually require students to finish multiple tasks, including judgment on trade-offs, documentation, modelling tools, and decision rationale.

Many studies report difficulties managing the heavy workload associated with architecture projects, especially for those projects requiring collaboration, hands-on prototyping, and extensive documentation within tight academic schedules, usually within one semester [11, 30]. For instance, large case-based and real-world projects, although valuable, are time-consuming because these projects require applying design thinking and system thinking consistently [30]. Hence, the tight schedule will reduce students' incentive and engagement, resulting in surface-level learning. In agile or simulation-based learning projects, students encounter even higher time constraints. These projects require students to make quick decisions based on specific backgrounds, coordinate in teams, and manage time efficiently. However, having students do structured reflection or deep technical discussions [3, 53] within a limited schedule is almost impossible.

The instructor's workload will increase when the subject becomes difficult for the students to understand. Therefore, instructor workload and expertise are the third most reported challenge (n=16) faced by educators tasked with teaching software architecture. Prior studies, in summary, address three significant challenges for instructors: high teaching loads, lack of training as an educator and lack of real-world experience. Many studies note that instructors often experience high teaching loads, similar to what students face in learning software architecture. This is because teaching strategies, like real-world-based or open-source project learning, also require a high workload from instructors, especially when they require frequent student feedback, ongoing assessment, and adaptation to team-based dynamics [11, 48]. In addition, a shortage of instructors with real-world architectural experience is another challenge, which will reduce their ability to provide students with in-depth analysis and explanations of the rationale behind architecture, including architectural evolution, scalability considerations, or trade-off analysis [33, 57]. Another common concern is the lack of structured training or pedagogical frameworks to support instructors in teaching architecture. This challenge relates to an instructor's teaching skills, including handling a large classroom, teaching students with different backgrounds and skill levels, and ensuring classroom attention [1, 59].

In contrast to the issues featured above, the technical and tool-related issues (n=15) relate to stakeholders in the software architecture industry. It refers to the complexity, immaturity, or lack of integration of the tools and platforms used to teach architectural concepts, particularly when used in similar real-world situations or to support collaborative learning. For example, one major problem is the steep learning curve associated with using architectural modelling tools, such as PlantUML, SonarQube, or various UML-based platforms [33, 62, 85].

Students often find difficulties in setting up, configuring and understanding these tools. When students spend too much time getting familiar with these tools, they cannot concentrate on integrating them into their architectural design. Tool immaturity and the absence of user-friendly interfaces are key reasons for the ineffectiveness. In addition, the lack of industrial standards and design could be another reason [33, 62]. Moreover, the functionality limitations of these tools might also reduce students' learning outcomes. As software architecture requires multiple integration layers, including data, business logic, and presentation, many existing platforms are limited in design-level analysis [85]. These technical and tool limitations will further influence educators' teaching results, as they will first have to understand these drawbacks and integrate appropriate tools into their courses, together with technical support, which will further improve their teaching load [34].

As reported in many reviewed studies (n=14), communication and collaboration are most prominent in team-based and hybrid learning environments since designing software architecture requires technical competence and practical interpersonal and transferable skills. Many studies highlight the difficulties of teamwork in developing software architecture, primarily when the projects are conducted in distributed or hybrid settings, with both in-person and online learners [22, 77, 80]. Common issues include miscommunication, inadequate participation, members with different backgrounds, and task allocation, which could negatively influence team performance and learning outcomes. The lack of collaborative tools, such as teamwork tools, could also give rise to collaboration problems due to a lack of familiarity, platform inconsistency, or technical limitations [77].

The student engagement and motivation challenge (n=14) refers to students' low engagement and motivation when learning software architecture subjects, as students perceive the materials covered as theoretical, detached from practical applications, or abstract [8, 69, 77]. Some studies also point out the role of teaching methods in promoting or hindering motivation. For example, traditional teaching methods could lead to passive learning, reducing students' motivation, while interactive methods, such as gamification, role-playing, and project-based learning, could increase interest and motivation.

Suppose students do not adequately understand and engage in software architecture. In that case, a pertinent issue is the student skill gap (n=14), which is the mismatch between their prior knowledge and the skills required to engage effectively with architectural tasks. This challenge typically arises when students perform poorly in their previous learning, including poor modelling, language handling, inadequate understanding of theoretical frameworks, and design thinking. It indicates that students are underprepared for software architecture courses [4, 33].

Encouraging students to engage and gain skills requires resources to scale the learning. Therefore, scalability and resource constraints (n=13) are the difficulties educators face in teaching software architecture at scale, as many teaching methods work well for small classes but are difficult to scale to large student cohorts [42, 95]. Issues related to this challenge include limited access to specialised tools and teaching support. Specifically, limiting access to hardware resources and infrastructure, such as cloud-based platforms or collaborative design simulators, is a key constraint. Further, the lack of support from educators, such as personalised tutoring, is also a constraint [30, 45].

If there are no efforts to address the abovementioned challenges, curriculum and content gaps (n=9) in software architecture could widen further. This would indicate the structural misalignments between

what is taught in software architecture courses and the skills, knowledge, and industry expectations that graduates are expected to meet. These mismatches include using outdated materials, lacking real-world exposure, and inadequate coverage of modern architectural topics [6, 30, 83]. For example, some research mentioned that the lack of teaching frameworks in software architecture reduces the reusability of the designed architecture [56]. Furthermore, the selected projects often fail to balance complexity and efficiency in teaching, making them either too complex to finish or too simple to reflect real-world situations [36].

Like any higher learning subject, assessment and evaluation are also challenging (n=8) in software architecture teaching. There are difficulties in designing, implementing, and grading assessments that accurately reflect student learning in software architecture courses. Due to the software architecture's abstract and complex nature, conventional assessment methods, such as exams and quizzes, are inadequate to assess student performance [26, 93]. This is why different assessment methods are widely adopted, as discussed in RQ3. However, the subjectivity in evaluating architectural designs, the changing nature of real-world practices and high workloads for instructors always reduce the validity and reliability of the evaluations [11].

Unlike other challenges, the lack of industry experts involved (n=5) reflects a broader disconnect between academic education and real-world architectural practice. Due to their tight schedules, industry experts might be reluctant to join software architecture education [35, 68]. Further, the language and Cultural Barriers challenge (n=1) refers to the challenges in class communication and handling misunderstandings due to students' different backgrounds [44]. In contrast, plagiarism (n=1) refers to academic integrity issues in architecture education [26].

6. DISCUSSION

In this section, the findings are examined to offer deeper understanding of the evolving pedagogy of Software Architecture (SA). They are critically compared with existing knowledge and highlighting emerging trends, gaps, and areas for further development.

6.1. Alignment of teaching strategies with industry needs.

The RQ1 finding illustrates the dominance of student-centred strategies, particularly project-based, real-world, and collaborative learning, in Software architecture education. These may be attributed to their authenticity, engagement, and alignment with post-study work environments. Among these, Project-Based Learning (PBL) emerged as a widely effective strategy supporting learning outcomes across the Understand, Apply, and Evaluate, which falls under the higher hierarchy of Bloom's taxonomy as seen in *Figure 4*. This is likely because PBL offers an authentic learning environment, encouraging students to grapple with real-world problems, iterate on designs, and collaborate within teams, all of which contribute to architectural practice, which is what awaits them in the professional world.

While PBL emerges as widely adopted, its effectiveness also appears to rely heavily on the level of student engagement, like the Game-based learning (GBL), which demonstrates interaction, competition and cooperation, essential for motivating student learning and improving knowledge retention. GBL was

also associated with the highest cognitive level, *Create* and exhibited multidimensional impacts spanning cognitive, behavioural and motivational domains. This may be attributed to the immersive nature of games and simulations, which allows students to explore architectural decisions in a secure and interactive environment. As digital learning continues to evolve, GBL appears increasingly promising. However, current literature reveals that it remains underexplored, with relatively few empirical studies substantiating long-term effectiveness. Despite this, the positive outcomes reported in the existing few studies highlight its potential, suggesting a valuable opportunity for further research to validate and expand upon these initial findings.

The lesson that we gained from this literature is that for the students, the most significant educational benefits are not achieved through passive observation and instruction, but the lessons that we gain through active engagement, such as building, discussing, reflecting on feedback and making informed decisions. This was evident in our findings, as none of the reviewed studies reported learning outcomes at the lower level of Bloom's taxonomy (i.e. Remember) indicating that most strategies, regardless of type, supported higher-order thinking and produced meaningful cognitive gains.

While traditional teaching is vital in establishing foundational knowledge, it should complement practical, collaborative, and personalised approaches. These principles align well with Agile methodology, which in this study is underexplored due to a limited number of relevant papers but we consider the strategy to hold significant potential. Emerging trends suggest that Agile-based approaches are gaining traction and are likely to become more prominent in SA education. This strategy promises to transform architecture education by portraying it as a dynamic, iterative process beyond replicating scrums and sprints. Agile learning can shift architectural education from a static design framework to a dynamic, continuous improvement model mimicking real-world scenarios. Therefore, if more effort and research are dedicated to uncovering its potential, it could significantly strengthen SA education.

Overall, the future of SA Education should aim not only to ensure conceptual understanding, but also to empower students to think critically, design creatively, and make informed architecture decisions in practice.

6.2. Enhancing Practical learning and Assessment in Software Architecture Education.

The growing emphasis on project-based and real-world practical experiences in the studies reviewed indicates a shift towards a more purposeful practice that aligns with industry needs. These approaches offer students a more engaging and context-driven learning experience through authentic design challenges. This alignment with professional practice not only strengthens learning but also lends itself well to evaluation through summative assessment, which is currently the most widely adopted approach. The assessment consists of examinations and project-based evaluations, such as capstone projects, which provide clear, quantifiable measures of student achievement and offer educators tangible benchmarks for evaluating content depth and final deliverables. Such assessments effectively capture learning outcomes by gauging student engagement and performance.

To fully harness the advantages of practical learning, we recommend employing experiential assessment methods that combine formative and summative techniques. This methods promtes a holistic perspective on student development by considering students' content knowledge alongside their collaboration,

reflection, assessment, and ability to transfer knowledge across various contexts. Such multifaceted approaches are particularly crucial in learning software architecture, where non-technical skills are viewed as necessary as technical skills. Moreover, experiential assessment is more effective in capturing the complexity and depth of learning, especially when evaluating practical strategies, such as knowledge gained in real-world and project-based experiences.

A notably rich but often overlooked theme emerged from practical experieoes, is the Tool and Platform based skills. This approach risks producing graduates lacking deep proficiency in essential industry tools, unable to perform rigorous architectural analysis and justification, and unprepared for the vital negotiation and teamwork skills inherent in the architect's role. Consequently, while students gain project exposure, they may lack the specific, refined expertise required for immediate effectiveness. This potential gap highlights a need to re-evaluate pedagogical strategies, potentially integrating more explicit, focused instruction in tooling, analytical methods, and collaborative dynamics to ensure graduates are comprehensively prepared for the complexities of professional software architecture practice. In our view, this gap may stem from the assumption that students naturally acquire essential competencies through projects, when in reality, these skills often remain underdeveloped. To address this, experiential assessment techniques can be employed to uncover individual skill gaps and support the development of tailored learning plans, fostering personal growth and deeper engagement in the field.

While existing assessment practices benefit future projects, we consider Architectural Kata as an innovative and relatively underutilised framework. This method mimics real-world design challenges, prompting students to make and justify their architectural choices and prototyping decisions. Moreover, it encompasses all four assessment types and nurtures vital career skills, including problem-solving, critical thinking, and design communication. The limited incorporation of Architectural Kata in established educational curricula presents a unique opportunity for innovative assessment development.

This review emphasises the growing reliance on practical, industry-aligned learning in software architecture education. While summative assessments like capstone projects are widely used, integrating experiential assessment can better capture student growth, particularly in overlooked areas like tool proficiency and analytical reasoning. Ultimately, assessment should not only evaluate learning but connect theory to practice and enable students to develop into competent, industry-ready architects.

6.3. Challenges in Teaching and Learning Software Architecture

In examining the challenges associated with software architecture education, our discussion will be primarily informed by *Figure 8* and *Figure 9*. The analysis of challenges and teaching strategies reveals two additional patterns. The first is the challenges faced by each teaching theme. As shown in *Figure 8*, Abstractness & Conceptual Difficulty is the most common challenge faced by almost all teaching themes, except personalised learning and software system learning. This might be because personalised learning often adapts to individual learning paces and styles, which could mitigate the cognitive overload. Further, Software System Learning usually emphasises practical tools, visualisations, and system-level implementation, reducing cognitive overload. On the other hand, abstraction and modelling face significant challenges in Abstractness and Conceptuality due to their focus on formal modelling techniques and abstraction-driven instruction.

Project-based learning encounters almost all of the identified challenges, with the exception of Language and cultural barriers. This may be attributed to the focus on English- language studies, which did not sufficiently capture these particular issue. Furthermore, the most mentioned challenge for this theme is the student skill and experience gap, indicating the significant gap between learning theoretical frameworks and applying them in projects.

Furthermore, scalability and resource constraints represent another common challenge across nearly all teaching themes. This reveals the need to balance cost and effectiveness when designing instructional approaches. For instance, although all teaching themes reviewed in our study reported positive outcomes in teaching software architecture, their practical implementation may be limited due to associated costs. An example of it is the collaborative learning that effectively improves student collaboration with various stakeholders. Still, the applicable fees, such as incorporating relevant platforms, obtaining real-world resources, and learning from industrial experts, are very high and might only be suitable for small classes.

Besides, time and workload constraints are another common challenge to almost all teaching themes, except agile-based and blended learning. Due to the complexity of the project or game design, game-based learning and project-based learning are the two themes commonly facing this challenge. Therefore, this also creates the demand to balance the complexity and simplicity of the project or game to ensure the tasks are tough but achievable, as also addressed in the challenge of curriculum design.

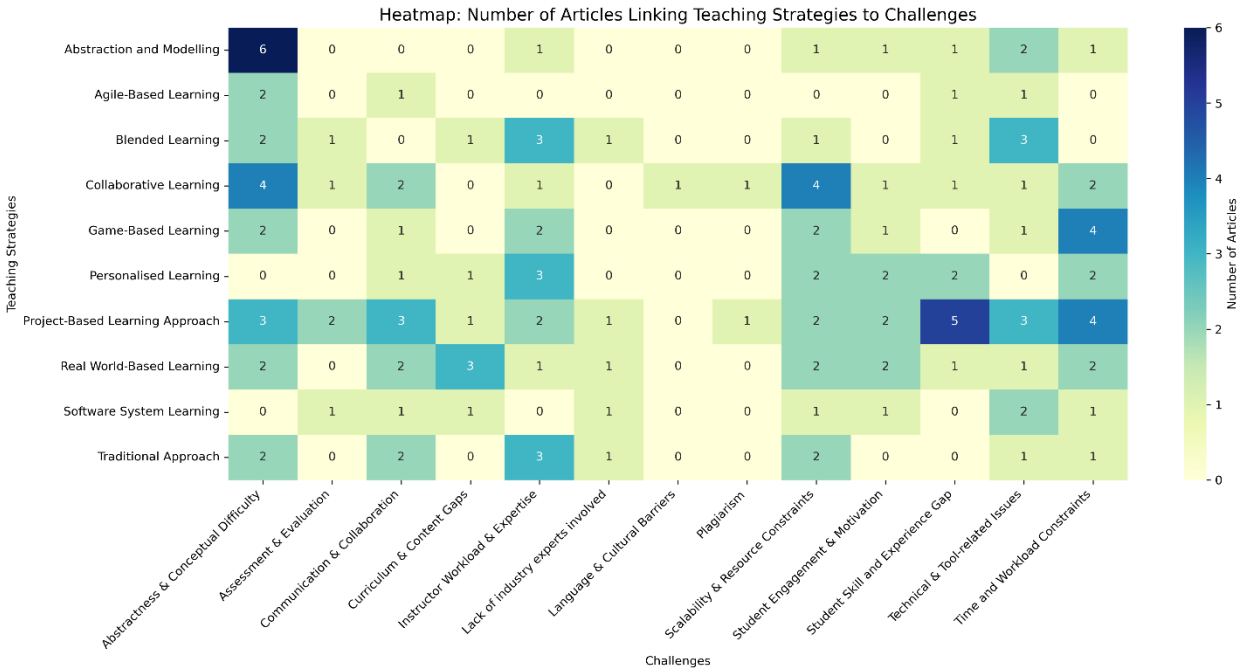


Figure 8: Heatmap displaying the linkage between teaching strategies and challenges

Another issue is that these challenges are not isolated; they are deeply interrelated and often reinforce one another, creating a complex web of barriers to effective learning and teaching in software architecture. These interactions are illustrated by *Figure 8*. Accordingly, as one of the most frequently cited challenges, Abstractness and Conceptual Difficulty are foundational challenges that could lead to many other challenges. For example, when students struggle to conceptualise architectural thinking, their motivation (Challenge 6) and ability to collaborate (Challenge 5) can diminish, especially in group settings where

advanced understanding is required. Further, Curriculum and Content Gaps (Challenge 9) may also intensify many other challenges, such as Assessment and Evaluation (Challenge 10) and incentives for industry experts to join education (Challenge 11). In summary, these interactions are complex, as demonstrated in *Figure 8*, which collectively influence the implementation of various teaching strategies and themes.

From the students' perspective, educators' strategies to handle these challenges are critically important. With Abstractness and Conceptual Difficulty and Time and Workload Constraints as the most influential barriers, students expect instructors to select appropriate, relatable examples that balance complexity and simplicity. This balance helps students grasp abstract architectural concepts without being overwhelmed. Moreover, efficient teaching methods, such as scaffolded learning, visual modelling tools, and real-world case studies, can bridge the gap between theory and application. In high-workload environments, students value structured content delivery and timely feedback that could enhance their deep understanding of the concepts in software architecture. All these require instructors to be mindful of workload and time constraints and be able to select the most suitable teaching themes to not only enhance conceptual understanding but also foster the skills and mindset needed for real-world SA practice to ensure their engagement.

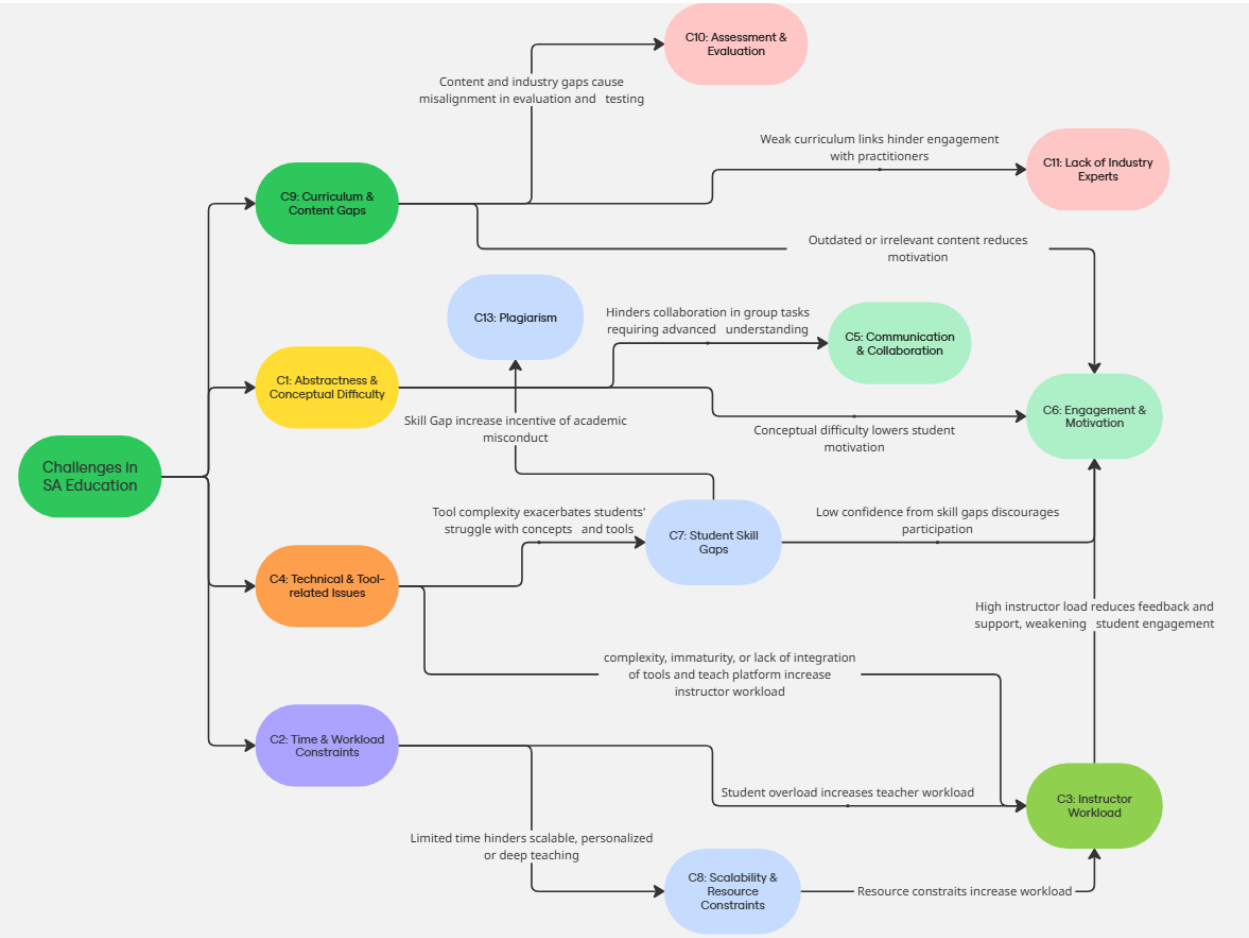


Figure 9: Interconnection of challenges

7. THREATS TO VALIDITY

This systematic review was conducted using rigorous methods. However, three threats to validity must be acknowledged here.

- There is a risk of selection bias in the screening and inclusion process. Although multiple reviewers reviewed and cross-validated the study inclusion, subjective judgment may still influence the decision. Moreover, the use of search strings may also influence it. Therefore, the search protocol was developed, and Covidence was used to minimise the bias through collaborative review.
- Data extraction was based on predefined thematic categories for teaching strategies, assessment types, outcomes and challenges. However, variations in terminology and interpretation across studies may have introduced inconsistencies. To reduce the bias in data extraction, each reviewer extracted the data for all research questions from the paper we downloaded during the search strategy. In the next step, each reviewer reviewed each research question, validating it with the articles. Moreover, the thorough discussion was conducted to ensure thematic reliability.
- The synthesis was based on thematic analysis and classification. While this approach supports interpretive depth, it may be limited by the heterogeneity of study contexts and reporting quality, potentially affecting the generalisability and consistency of the insights presented. To reduce this threat, each researcher proofread the draft report several times and corrected the misinformed sections.

8. FUTURE WORK

In the future scope, we would like to propose the following work:

- **Longitudinal Study on Findings**
  This paper has mentioned a few teaching strategies and assessment methods that improved learning performance and prepared students for the job market with skills. Yet, no studies have been done on teaching strategies, assessment methods, and their professional impact. Therefore, a study can be conducted to validate our findings.
- **Empirical Study of Agile Methodology**
  Agile methodology was found promising, but it was found to be rarely used in SA teaching. Future researchers could conduct an experimental study on student engagement and skill development using the agile method.
- **Case Study on the Use of Architectural Kata as Assessment Tools**
  Most papers do not capture the use of Architectural Kata as an assessment method in SA teaching. It is most effective in assessing students' learning. Therefore, the case study can be conducted to evaluate the effectiveness of Architectural Kata specifically in SA teaching.

9. CONCLUSION

This study is a systematic mapping to investigate various SA teaching strategies based on their capacity to provide hands-on practical experiences, assessment methods, outcomes, and challenges. Based on the review of 82 peer-reviewed papers from 2010 to 2024, our research synthesises their teaching methods into 10 teaching themes, dominated by student-centred strategies, with Project-Based Learning (PBL)

emerging as the most frequently employed strategy, followed by Real-World Based Learning and Collaborative Learning approaches. These themes are closely aligned with industrial needs due to their focus on hands-on experiences, design thinking, critical reasoning, teamwork and application of architectural frameworks.

Our review revealed that most teaching themes achieve higher-order learning outcomes, dominated by the "Apply" and "Understand" levels of Bloom's taxonomy. However, teaching themes targeting the advanced levels of Bloom's taxonomy ("Create" and "Evaluate") remain underdeveloped, indicating a strong demand for further innovation in teaching methodologies. Summative assessment practices dominate assessment approaches, such as project evaluations and examinations. However, experiential assessments, such as Architectural Kata, are recommended to better align with industry needs to evaluate students' performance from broader dimensions, including technical ability and transferable skills.

Despite these achievements, significant challenges exist in teaching SA. Students face challenges of abstract and conceptual difficulties, time and workload constraints, engagement difficulties, and skill gaps. Instructors struggle with heavy workloads, a lack of practical experience, and insufficient educational training. In addition, Technical and tool-related Issues, scalability and resource Constraints require feasibility assessments when designing teaching strategies.

Addressing these challenges requires collaboration among all stakeholders in SA education. Educational institutions must seek to adopt diverse teaching strategies, enhance experiential learning, and incorporate innovative assessment methods. Instructors should be mindful of workload and time constraints and utilise appropriate teaching strategies to foster the skills and mindset needed for real-world SA practice to ensure their engagement.

In conclusion, SA education focuses on industry-relevant and student-centred pedagogies. In addition to the current Project-Based Learning (PBL) approaches, further empirical research might focus more on longitudinal studies and experimental designs, including testing the effectiveness of Agile-based learning and the use of Architectural Kata as Assessment Tools.

10. DECLARATIONS

We declare that this work is purely of our own and not someone's work, nor is it a published document to the best of our knowledge and belief. In addition, we certify that no part of this work will be used by someone to submit on our behalf in the future.

We permit the digital version of our work (no edited version) to be shared by the unit convenor, sponsor and mentor for review for publication in case. We will edit the final version for the submission upon feedback from the sponsor and mentor.

We acknowledge the support received from our sponsor and mentor for this project.

11. REFERENCES

[1] A. C. Gonçalves, V. V. G. Neto, D. J. Ferreira, and U. F. Silva, "Flipped Classroom Applied to Software Architecture Teaching," 2020. [Online]. Available: https://doi.org/10.1109/FIE44824.2020.9274255.

[2] M. Galster and S. Angelov, "What makes teaching software architecture difficult?," 2016. [Online]. Available: https://doi.org/10.1145/2889160.2889187.

[3] H. Cervantes, S. Haziyev, O. Hrytsay, and R. Kazman, "Smart Decisions: An Architectural Design Game," *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C),* pp. 327-335, 2016, doi: https://doi.org/10.1145/2889160.2889184.

[4] R. Kazman *et al.*, "A Better Way to Teach Software Architecture," *Software Architecture: Research Roadmaps from the Community,* pp. 101-110, 2023, doi: 10.1007/978-3-031-36847-9_6.

[5] M. U. Rafique, A. M. Mohammed, S. Li, A. T. Khan, and S. Kadry, "Integrating Open-Source Tools for Embedded Software Teaching: A Case Study," 2019.

[6] O. E. Lieh and Y. Irawan, "Teaching Adult Learners on Software Architecture Design Skills," 2018. [Online]. Available: https://doi.org/10.1109/FIE.2018.8658714.

[7] A. Varma and M. S. Jafri, "COVID-19 responsive teaching of undergraduate architecture programs in India: learnings for post-pandemic education," *Archnet-Ijar International Journal of Architectural Research,* vol. 15, no. 1, pp. 189-202, 2021, doi: 10.1108/arch-10-2020-0234.

[8] L. M. Castro, "Role-playing software architecture styles," *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C),* pp. 171-174, 2023, doi: 10.1109/ICSA-C57050.2023.00045.

[9] M. Dorodchi, N. Dehbozorgi, M. Fallahian, and S. Pouriyeh, "Teaching Software Engineering Using Abstraction through Modeling," vol. 20, no. 4, pp. 515-532, 2021, doi: https://doi.org/10.1145/3502718.3524758.

[10] L. F. Al-Qora'n, A. Jawarneh, and J. T. Nganji, "Toward Creating Software Architects Using Mobile Project-Based Learning Model (Mobile-PBL) for Teaching Software Architecture," *Multimodal Technol. Interact.,* vol. 7, no. 3, p. 31, 2023, doi: 10.3390/MTI7030031.

[11] A. Cain and M. A. Babar, "Reflections on applying constructive alignment with formative feedback for teaching introductory programming and software architecture," 2016. [Online]. Available: https://doi.org/10.1145/2889160.2889185.

[12] C. M. Theile and A. L. Beall, "Conducting a Systematic Review of the Literature," *Journal of Dental Hygiene,* vol. 98, no. 2, 2024.

[13] D. Budgen and P. Brereton, "Performing systematic literature reviews in software engineering," presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006. [Online]. Available: https://doi.org/10.1145/1134285.1134500.

[14] C. S. C. Rodrigues and C. M. L. Werner, "Software Architecture Teaching : A Systematic Review," 2008.

[15] W. L. Pantoja Yépez, J. A. Hurtado Alegría, A. Bandi, and A. W. Kiwelekar, "Training software architects suiting software industry needs: A literature review," *Education and Information Technologies,* vol. 29, no. 9, pp. 10931-10994, 2024/06/01 2024, doi: 10.1007/s10639-023-12149-x.

[16] O. Cico, L. Jaccheri, A. Nguyen-Duc, and H. Zhang, "Exploring the intersection between software industry and Software Engineering education - A systematic mapping of Software Engineering Trends," *Journal of Systems and Software,* vol. 172, p. 110736, 2021/02/01/ 2021, doi: https://doi.org/10.1016/j.jss.2020.110736.

[17] N. Assyne, H. Ghanbari, and M. Pulkkinen, "The state of research on software engineering competencies: A systematic mapping study," *Journal of Systems and Software,* vol. 185, p. 111183, 2022/03/01/ 2022, doi: https://doi.org/10.1016/j.jss.2021.111183.

[18] B. Oliveira, L. Garcés, K. Lyra, D. Santos, S. Isotani, and E. Nakagawa, "An Overview of Software Architecture Education," 08, 2022.

[19] A. Cooke, D. Smith, and A. Booth, "Beyond PICO:The SPIDER Tool for Qualitative Evidence Synthesis," *Qualitative Health Research,* vol. 22, no. 10, pp. 1435-1443, 2012, doi: 10.1177/1049732312452938.

[20] W. Alf Inge, "Extensive Evaluation of Using a Game Project in a Software Architecture Course," *ACM Trans. Comput. Educ.,* vol. 11, no. 1, p. Article 5, 2011, doi: 10.1145/1921607.1921612.

[21] U. o. Leeds. "Literature searching explained." University of Leeds. https://library.leeds.ac.uk/info/1404/literature-searching/14/literature-searching-explained/4 (accessed March 2, 2025, 2025).

[22] F. Fabian, H. Arto, L. Matti, K. Kati, Y. Sezin, and M. Hanna, "Designing and implementing an environment for software start-up education: Patterns and anti-patterns," *Journal of Systems and Software,* vol. 146, pp. 1-13, 2018, doi: https://doi.org/10.1016/j.jss.2018.08.060.

[23] C. R. Rupakheti and S. V. Chenoweth, "Teaching Software Architecture to Undergraduate Students: An Experience Report," 2015. [Online]. Available: https://doi.org/10.1109/ICSE.2015.177.

[24] M. Herold, J. Bolinger, R. Ramnath, T. Bihari, and J. Ramanathan, "Providing end-to-end perspectives in software engineering," *2011 Frontiers in Education Conference (FIE),* pp. S4B-1, 2011, doi: 10.1109/FIE.2011.6142927.

[25] L. Zheng, "Using public and free platform-as-a-service (PaaS) based lightweight projects for software architecture education," Seoul, South Korea, 2020. [Online]. Available: https://doi.org/10.1145/3377814.3381704.

[26] A. Giovanni Gonzalez, "Using Apprenticeship and Product Based Learning to Improve Programming Outcomes in Introductory Computer Science Courses," 2024.

[27] B. Dahl and A. Kolmos, "Students' Attitudes towards Group-Based Project Exams in Two Engineering Programmes," vol. 3, no. 2, pp. 62-79, 2015.

[28] B. R. N. Oliveira, L. Garcés, K. T. Lyra, D. S. Santos, S. Isotani, and E. Y. Nakagawa, "An Overview of Software Architecture Education," *Anais do Congresso Ibero-Americano em Engenharia de Software (CIbSE),* pp. 76-90%@ 0000-0000, 2022-06-13 2022, doi: 10.5753/cibse.2022.20964.

[29] W. L. P. Yepez, J. A. H. Alegria, and A. Kiweleker, "Aligning Software Architecture Training with Software Industry Requirements," *International Journal of Software Engineering and Knowledge Engineering,* vol. 33, no. 03, pp. 435-460, 2023/03/01 2023, doi: 10.1142/S0218194023500031.

[30] M. Vidoni, J. M. Montagna, and A. Vecchietti, "Project and team-based strategies for teaching software architecture," 2018.

[31] L. Zhang, Y. Li, and N. Ge, "Exploration on Theoretical and Practical Projects of Software Architecture Course," *2020 15th International Conference on Computer Science & Education (ICCSE),* pp. 391-395, 2020, doi: 10.1109/ICCSE49874.2020.9201748.

[32] C. A. Mattmann, N. Medvidovic, S. Malek, G. Edwards, and S. Banerjee, "A Middleware Platform for Providing Mobile and Embedded Computing Instruction to Software Engineering Students," *IEEE Transactions on Education,* vol. 55, no. 3, pp. 425-435, 2012, doi: 10.1109/TE.2012.2182998.

[33] B. Pando and J. Castillo, "PlantUMLGen: A tool for teaching Model Driven Development," *2022 17th Iberian Conference on Information Systems and Technologies (CISTI),* pp. 1-6, 2022, doi: 10.23919/CISTI54924.2022.9820348.

[34] Y. M. Lau, C. M. Koh, and L. Jiang, "Teaching Software Development for Real-World Problems Using a Microservice-Based Collaborative Problem-Solving Approach," *2024 IEEE/ACM 46th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET),* pp. 22-33, 2024, doi: 10.1145/3639474.3640064.

[35] Y. Wilson Libardo Pantoja, A. Julio Ariel Hurtado, B. Ajay, and W. K. Arvind, "Training Software Architects Suiting Software Industry Needs: A Literature Review," vol. 29, no. 9, pp. 10931-10994, 2024.

[36] M. Ramachandran and R. Sedeeq, "Learning Environment for Problem-based Learning in Teaching Software Components and Service-oriented Architecture," 2017. [Online]. Available: https://doi.org/10.5220/0006257702490255.

[37] E. L. Ouh, B. K. S. Gan, Y. Irawan, Ieee, A. S. E. E. E. R. Ieee, and I. E. S. I. C. S. Methods Div, "Did our Course Design on Software Architecture meet our Student's Learning Expectations?," *IEEE Frontiers in Education Conference (FIE),* 2020. [Online]. Available: <Go to ISI>://WOS:000646660800146.

[38] D. El-Mahdy, "Learning by Doing: Integrating Shape Grammar as a Visual Coding Tool in Architectural Curricula," *Nexus Network Journal,* vol. 24, no. 3, pp. 701-716, 2022/09/01 2022, doi: 10.1007/s00004-022-00608-w.

[39] E. L. Ouh and Y. Irawan, "Applying Case-Based Learning for a Postgraduate Software Architecture Course," presented at the Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education, Aberdeen, Scotland Uk, 2019. [Online]. Available: https://doi.org/10.1145/3304221.3319737.

[40] P. O. Antonino, A. Morgenstern, and T. Kuhn, "Embedded-Software Architects: It's Not Only about the Software," *IEEE Software,* vol. 33, no. 6, pp. 56-62, 2016, doi: 10.1109/MS.2016.142.

[41] N. A. A. Jamal and N. M. Ali, "Comparative critiquing and example-based approach for learning client-server design," *2017 IEEE Conference on e-Learning, e-Management and e-Services (IC3e),* pp. 30-35, 2017, doi: 10.1109/IC3e.2017.8409234.

[42] C. Santi and X. Fatos, "CLPL: Providing software infrastructure for the systematic and effective construction of complex collaborative learning systems," *Journal of Systems and Software,* vol. 83, no. 11, pp. 2083-2097, 2010, doi: https://doi.org/10.1016/j.jss.2010.06.013.

[43] R. Capilla, O. Zimmermann, C. Carrillo, and H. Astudillo, "Teaching Students Software Architecture Decision Making," 2020. [Online]. Available: https://doi.org/10.1007/978-3-030-58923-3_16.

[44] F. Abdullah *et al.*, "Case study on perspicacity of collaborative learning experiences," *IOP Conference Series: Materials Science and Engineering,* vol. 291, no. 1, p. 012006, 2017/12/01 2017, doi: 10.1088/1757-899X/291/1/012006.

[45] A. v. Deursen *et al.*, "A Collaborative Approach to Teaching Software Architecture," 2017. [Online]. Available: https://doi.org/10.1145/3017680.3017737.

[46] G. Zhang and M. Rong, "Exploration and practice of software engineering core curriculum construction," *2011 6th International Conference on Computer Science & Education (ICCSE),* pp. 703-705, 2011, doi: 10.1109/ICCSE.2011.6028734.

[47] M. Cao and Z. Cao, "Teaching data structures and Software Architecture while constructing curriculum platform," *2011 6th International Conference on Computer Science & Education (ICCSE),* pp. 1433-1437, 2011, doi: 10.1109/ICCSE.2011.6028899.

[48] M. A. Manal and M. M. Ana, "Gamification in software engineering education: A systematic mapping," *Journal of Systems and Software,* vol. 141, pp. 131-150, 2018, doi: https://doi.org/10.1016/j.jss.2018.03.065.

[49] J. A. Parejo, J. Troya, S. Segura, A. del-Río-Ortega, A. Gámez-Díaz, and A. E. Márquez-Chamorro, "Flipping laboratory sessions: an experience in computer science," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje,* vol. 15, no. 3, pp. 183-191, 2020.

[50] P. Lago, J. F. Cai, R. C. de Boer, P. Kruchten, and R. Verdecchia, "DecidArch: Playing cards as software architects," in *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)*, 2019: Hawaii International Conference on System Sciences (HICSS), pp. 7815-7824.

[51] C. M. Aldenhoven and R. S. Engelschall, "The beauty of software architecture," *2023 IEEE 20th International Conference on Software Architecture (ICSA),* pp. 117-128, 2023, doi: 10.1109/ICSA56044.2023.00019.

[52] C. H. Montenegro, H. Astudillo, and M. C. G. Álvarez, "ATAM-RPG: A role-playing game to teach architecture trade-off analysis method (ATAM)," *2017 XLIII Latin American Computer Conference (CLEI),* pp. 1-9, 2017, doi: 10.1109/CLEI.2017.8226416.

[53] R. C. d. Boer, P. Lago, R. Verdecchia, and P. Kruchten, "DecidArch V2: An Improved Game to Teach Architecture Design Decision Making," *2019 IEEE International Conference on Software Architecture Companion (ICSA-C),* pp. 153-157, 2019, doi: 10.1109/ICSA-C.2019.00034.

[54] M. Campo, A. Amandi, and J. C. Biset, "A Software Architecture Perspective about Moodle Flexibility for Supporting Empirical Research of Teaching Theories," vol. 26, no. 1, pp. 817-842, 2021.

[55] M. Feldgen and O. Clua, "Teaching effective requirements engineering for large-scale software development with scaffolding," *2014 IEEE Frontiers in Education Conference (FIE) Proceedings,* pp. 1-8, 2014, doi: 10.1109/FIE.2014.7044176.

[56] S. Chimalakonda and K. V. Nori, "A Patterns Based Approach for the Design of Educational Technologies," vol. 31, no. 4, pp. 2114-2133, 2023.

[57] L. M. Castro, M. Hidalgo, and H. Astudillo, "Effective Teaching Strategies for Large Classes: A Case Study in Software Architecture Education," 2024. [Online]. Available: https://doi.org/10.1145/3702163.3702414.

[58] C. Corritore and B. Love, "Redesigning an Introductory Programming Course to Facilitate Effective Student Learning: A Case Study," vol. 19, pp. 91-136, 2020.

[59] O. E. Lieh and Y. Irawan, "Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course," *2018 IEEE Frontiers in Education Conference (FIE),* pp. 1-9, 2018, doi: 10.1109/FIE.2018.8659200.

[60] O. E. Lieh and Y. Irawan, "Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course," in *2018 IEEE Frontiers in Education Conference (FIE)*, 3-6 Oct. 2018 2018, pp. 1-9, doi: 10.1109/FIE.2018.8659200.

[61] L. Kazi, "The Role of Modelling in Business Software Development: Case Study of Teaching and Industrial Practice in Zrenjanin, Serbia," *2019 29th International Conference on Computer Theory and Applications (ICCTA),* pp. 14-23, 2019, doi: 10.1109/ICCTA48790.2019.9478817.

[62] C. Federico *et al.*, "How do we teach modelling and model-driven engineering? a survey," Copenhagen, Denmark, 2018. [Online]. Available: https://doi.org/10.1145/3270112.3270129.

[63] V. Liubchenko, N. Shakhovska, and M. O. Medykovskyy, "Queueing Modeling in the Course in Software Architecture Design," pp. 550-560, 2019. [Online]. Available: https://doi.org/10.1007/978-3-030-01069-0_39.

[64] P. J. Thomas, "Characterizing Student Proficiency in Software Modeling in Terms of Functions, Structures, and Behaviors," 2021, doi: https://doi.org/10.1145/3458039.

[65] D. Tappan, "A quasi-network-based fly-by-wire simulation architecture for teaching software engineering," 2015. [Online]. Available: https://doi.org/10.1109/FIE.2015.7344345.

[66] B. R. Gacitúa, M. Diéguez, and E. Vidal, "Forming software architects in early stages: From craft to engineering," *2017 36th International Conference of the Chilean Computer Science Society (SCCC),* pp. 1-8, 2017, doi: 10.1109/SCCC.2017.8405130.

[67] G. Agerwala and L. Bass, "Teaching Software Architecture Design - Building Intuition," 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10669881.

[68] E. Golden, "Early-Stage Software Design for Usability," Ph.D., Carnegie Mellon University, United States -- Pennsylvania, 3438450, 2010. [Online]. Available: https://www.proquest.com/dissertations-theses/early-stage-software-design-usability/docview/845727792/se-2

[69] A. S. Marcolino and E. F. Barbosa, "Towards a Software Product Line Architecture to Build M-learning Applications for the Teaching of Programming," 2017. [Online]. Available: https://hdl.handle.net/10125/41922.

[70] C. S. C. Rodrigues, "VisAr3D: an approach to software architecture teaching based on virtual and augmented reality," 2010. [Online]. Available: https://doi.org/10.1145/1810295.1810388.

[71] J. S. Urrego and D. Correal, "Archinotes: A tool for assisting software architecture courses," *2013 26th International Conference on Software Engineering Education and Training (CSEE&T),* pp. 80-88, 2013, doi: 10.1109/CSEET.2013.6595239.

[72] C. S. C. Rodrigues and C. M. L. Werner, "Making the comprehension of software architecture attractive," *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T),* pp. 416-420, 2011, doi: 10.1109/CSEET.2011.5876116.

[73] S. Angelov and P. d. Beer, "Designing and applying an approach to software architecting in agile projects in education," *Journal of Systems and Software,* vol. 127, pp. 78-90, 2017, doi: https://doi.org/10.1016/j.jss.2017.01.029.

[74] S. Ezequiel, R. Guillermo, S. Álvaro, and C. Marcelo, "Towards better Scrum learning using learning styles," *Journal of Systems and Software,* vol. 111, pp. 242-253, 2016, doi: https://doi.org/10.1016/j.jss.2015.10.022.

[75] G. Wedemann, "Scrum as a Method of Teaching Software Architecture," 2018. [Online]. Available: https://doi.org/10.1145/3209087.3209096.

[76] G. Katipoglu, S. Utku, I. Mijailovic, E. Mekic, D. Avdic, and P. Milic, "Action Research Approach to Analysis of Teaching of Blockchain Web 3.0 Application Based on MACH Architecture," *Applied Sciences-Basel,* vol. 14, no. 23, 2024, doi: 10.3390/app142311158.

[77] F. D. Giraldo *et al.*, "Applying a distributed CSCL activity for teaching software architecture," *International Conference on Information Society (i-Society 2011),* pp. 208-214, 2011, doi: 10.1109/i-Society18435.2011.5978540.

[78] V. Niculescu, C. Şerban, and A. Vescan, "Does Cyclic Learning have Positive Impact on Teaching Object-Oriented Programming?," *2019 IEEE Frontiers in Education Conference (FIE),* pp. 1-9, 2019, doi: 10.1109/FIE43999.2019.9028600.

[79] N. El-Bathy, C. Gloster, G. Azar, M. El-Bathy, G. Stein, and R. Stevenson, "Intelligent teaching models for STEM related careers using a service-oriented architecture and management science," *2012 IEEE International Conference on Electro/Information Technology,* pp. 1-7, 2012, doi: 10.1109/EIT.2012.6220720.

[80] C.-V. Edgar, C.-B. Carlos, and D. Dhimitraq, "Application of Project-Based Learning to a Software Engineering course in a hybrid class environment," *Information and Software Technology,* vol. 158, p. 107189, 2023, doi: https://doi.org/10.1016/j.infsof.2023.107189.

[81] A. M. Moreno, M.-I. Sanchez-Segura, F. Medina-Dominguez, and L. Carvajal, "Balancing software engineering education and industrial needs," *Journal of Systems and Software,* vol. 85, no. 7, pp. 1607-1620, 2012/07/01/ 2012, doi: https://doi.org/10.1016/j.jss.2012.01.060.

[82] C. R. Rupakheti, M. Hays, S. Mohan, S. Chenoweth, and A. Stouder, "On a pursuit for perfecting an undergraduate requirements engineering course," *Journal of Systems and Software,* vol. 144, pp. 366-381, 2018/10/01/ 2018, doi: https://doi.org/10.1016/j.jss.2018.07.008.

[83] V. Garousi, G. Giray, E. Tüzün, C. Catal, and M. Felderer, "Aligning software engineering education with industrial needs: A meta-analysis," *Journal of Systems and Software,* vol. 156, pp. 65-83, 2019/10/01/ 2019, doi: https://doi.org/10.1016/j.jss.2019.06.044.

[84] L. Zhang, J. W. Niu, and Ieee, "A Comprehensive Experiment Approach to Enhancing Computer Engineering Ability," *IEEE Frontiers in Education Conference (FIE),* 2022, doi: 10.1109/fie56618.2022.9962680.

[85] R. Claudia and Francesca, "Collaborative and teamwork software development in an undergraduate software engineering course," *Journal of Systems and Software,* vol. 144, pp. 409-422, 2018, doi: https://doi.org/10.1016/j.jss.2018.07.010.

[86] N. A. Ernst and M. P. Robillard, "A study of documentation for software architecture," *Empirical Software Engineering,* vol. 28, no. 5, p. 122, 2023, doi: https://doi.org/10.1007/s10664-023-10347-2.

[87] U. Nasir, "Using Architectural Kata in Software Architecture Course: An Experience Report," presented at the Proceedings of the 5th European Conference on Software Engineering Education, Seeon/Bavaria, Germany, 2023. [Online]. Available: https://doi.org/10.1145/3593663.3593694.

[88] D. Wiliam, "What is assessment for learning?," *Studies in Educational Evaluation,* vol. 37, no. 1, pp. 3-14, 2011/03/01/ 2011, doi: https://doi.org/10.1016/j.stueduc.2011.03.001.

[89] T. Crooks, "Assessment for learning in the accountability era: New Zealand," *Studies in Educational Evaluation,* vol. 37, no. 1, pp. 71-77, 2011/03/01/ 2011, doi: https://doi.org/10.1016/j.stueduc.2011.03.002.

[90] L. M. Earl, *Assessment as learning: Using classroom assessment to maximize student learning*. Corwin press, 2012.

[91] X. Liu, Y. Zhao, J. Yuan, W. Rao, L. Lu, and F. Huang, "Experimental Teaching Reform to Embedded System Curriculum," in *2020 15th International Conference on Computer Science & Education (ICCSE)*, 18-22 Aug. 2020 2020, pp. 737-742, doi: 10.1109/ICCSE49874.2020.9201757.

[92] N. E. Adams, "Bloom's taxonomy of cognitive learning objectives," (in eng), *J Med Libr Assoc,* vol. 103, no. 3, pp. 152-3, Jul 2015, doi: 10.3163/1536-5050.103.3.010.

[93] Y. Tao, G. Liu, J. Mottok, R. Hackenberg, and G. Hagel, "Ranking task activity in teaching software engineering," *2016 IEEE Global Engineering Education Conference (EDUCON),* pp. 1023-1027, 2016, doi: 10.1109/EDUCON.2016.7474678.

[94] U. Nasir and M. Laiq, "Threshold Concepts and Skills in Software Architecture: Instructors' Perspectives," *2022 29th Asia-Pacific Software Engineering Conference (APSEC),* pp. 547-553, 2022, doi: 10.1109/APSEC57359.2022.00076.

[95] C. Orges, J. Letizia, N.-D. Anh, and Z. He, "Exploring the intersection between software industry and Software Engineering education - A systematic mapping of Software Engineering Trends," *Journal of Systems and Software,* vol. 172, p. 110736, 2021, doi: https://doi.org/10.1016/j.jss.2020.110736.