




# Training software architects suiting software industry needs: A literature review

Wilson Libardo Pantoja Yépez<sup>1</sup>  · Julio Ariel Hurtado Alegría<sup>1</sup> · Ajay Bandi<sup>2</sup> · Arvind W. Kiwelekar<sup>3</sup>

Received: 15 November 2022 / Accepted: 16 August 2023 / Published online: 16 October 2023  
© The Author(s) 2023

## Abstract

The ability to define, evaluate, and implement software architectures is a fundamental skill for software engineers. However, teaching software architecture can be challenging as it requires students to be involved in real-context projects with high degrees of complexity. This involves making trade-off decisions among several quality attributes. Furthermore, the academic perception of software architecture differs from the industrial viewpoint. To address this issue, a study was conducted to identify and analyze the strategies, challenges, and course experiences used for teaching software architectures. The study analyzed 56 articles reporting on teaching experiences focused specifically on software architectures or focused on software engineering in general but discussing software architecture. The main contributions of this work include identifying strategies used in educating software architecture students aligned with the needs of the software industry. These strategies include short design projects, large development projects, and projects with actual clients. 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. This study also recognizes the set of skills that students of software architecture should develop during training, such as leadership and negotiation. The challenges in software architecture training were discussed, such as instructors' lack of experience in actual projects, the abstract and fuzzy nature of software architectures, and the difficulty of involving clients and industry experts. Evaluation methods commonly used in training software architects, such as surveys, pre-test/post-test, and quality metrics on architectural artifacts, were identified and described. Overall, this study guides researchers and educators in improving their software architecture courses by incorporating strategies reported by the literature review. These strate-

---

Julio Ariel Hurtado Alegría and Ajay Bandi are both contributed equally to this work.

---

✉ Wilson Libardo Pantoja Yépez  
wpantoja@unicauca.edu.co

gies can bring architecture courses closer to the needs and conditions of the software industry.

**Keywords** Software architecture · Training · Systematic mapping · Industry

## 1 Introduction

Software architecture is a significant topic in Software Engineering curricula, as evidenced by many academic publications (Zhang et al., 2020; Van Deursen et al., 2017). Architectural design is frequently integrated into other software engineering courses in most computer science and related programs (Li, 2020). However, teaching software architecture design is challenging due to its multi-view definition, practical requirements such as coding, and the use of complex problems (Galster & Angelov, 2016). The architect's role is multifaceted, demanding experience in designing and developing real systems, the ability to analyze and quickly solve problems while detecting the root cause and making critical decisions for the project based on context and communication skills (Van Deursen et al., 2017).

Teaching software architecture is complex, requiring a genuine context, teamwork, projects with sufficient complexity, and ongoing student support and guidance (Angelov & de Beer, 2017). It is particularly challenging for teachers to instruct undergraduate students on abstract concepts such as principles, heuristics, and patterns crucial to software architecture design, particularly since they lack experience in the software industry (Lieh & Irawan, 2018). Teaching architecture concepts goes beyond conventional courses where the teacher guides an area of knowledge from books, lectures, and exercises. The abstract and extensive nature of software architecture design (Li, 2020; Zhang et al., 2020) presents a challenge in teaching software architecture because abstract concepts are difficult to comprehend. Practical learning involves complex projects, real clients, and diverse software quality attributes that compete with each other.

Perceptions differ between colleges and companies regarding software architecture practices and the responsibilities and skills required of architects. Trainers need to balance numerous factors, such as quality, scope, depth, applicability, soft and technical skills, and individual and collaborative learning to ensure effective teaching in the classroom (Palacin-Silva et al., 2017; Abad et al., 2019; Angelov & de Beer, 2017; Wei et al., 2020). It is necessary to analyze past experiences with teaching and learning software architecture activities to understand the gap between academic insight and industrial needs.

This paper presents a systematic mapping study aimed at resolving issues related to the teaching of software architecture. We consider course experiences and teaching strategies that allow us to align teaching practices with the competencies expected of software architects in the industry. These strategies can bring architecture courses closer to the needs and conditions of the software industry, improving graduate students' employability, facilitating their transition from academic institutes to industry, and achieving better learning outcomes. Without such strategies, fresh graduates may

need to undergo re-training, resulting in training costs and delays in hiring new graduates by prospective employers.

Our work makes a significant contribution to the field of research by identifying the teaching strategies employed in various experiences reported in the literature. We explore how these strategies intersect with the competencies achieved by students, the course contents studied, the challenges encountered during the training process, and the methods used to confirm each experience. By analyzing and cross-referencing this information, we gain valuable insights into existing research gaps.

## 2 Frequent software architecture concepts from the industry

This section explores various architectural concepts that are used in industrial practices and software architecture courses. These concepts include the definition of software architecture, quality attributes, architecture patterns, and architectural tactics. These ideas have been identified and defined based on their frequent appearance in the literature. Recognizing and understanding these concepts is crucial when designing software architecture training programs.

### 2.1 Definition of software architecture

The concept of software architecture is continually evolving, and it is essential to understand the various definitions of software architecture reported in the literature. A modern definition of software architecture might involve a set of fundamental decisions concerning the structure of the software system, which guides the design and construction of the system (Jansen & Bosch, 2005; Fowler, 2003; De Boer & Van Vliet, 2009). This structure includes the organization of the components, the way they communicate, and the distribution of responsibilities among them (Bass et al., 2012). Software architecture also addresses issues related to the quality of non-functional attributes, such as the system's performance, scalability, security, usability, and maintainability (Richards & Ford, 2020).

### 2.2 Quality attributes

Quality attributes refer to the specific traits a software product satisfies, with each attribute being associated with specific metrics that define the quality levels for a software product (Sabry, 2015). Quality attributes include security, reliability, performance, and interoperability, which arise from complex business rules and quality concerns (Dobrica & Niemela, 2002). Managing these quality attributes appropriately is crucial since inappropriate handling poses a significant business risk. The architect needs to consider the potential conflicts between quality attributes and solve them through trade-offs.

## 2.3 Architecture patterns

Architecture patterns refer to common solution structures to similar design problems (Harrison & Avgeriou, 2010). Each pattern describes a general software system structure or high-level behavior that should satisfy a product's functionalities, qualities, and constraints. These patterns are chosen based on early design decisions, such as the satisfaction of functional requirements, non-functional requirements, and system constraints (Harrison & Avgeriou, 2010).

## 2.4 Architectural tactics

Architectural tactics refer to high-level abstractions that capture decisions made to achieve quality goals in software architecture. These tactics can be design-time, such as *hiding information* to improve modifiability, or runtime, such as *managing concurrency* to improve performance (Harrison & Avgeriou, 2010). Architectural tactics have varying impacts on software design, and a developer can initiate a tactic through a particular architectural pattern.

The role of the software architect involves a range of technical skills, including software design and development, as well as a deep understanding of systems and network concepts. The architect needs analytical skills to understand the problem and diagnose potential root causes quickly. Additionally, leadership skills are crucial to making decisions crucial to the project based on the context and environment (Lieh & Irawan, 2019).

## 3 Related work

While there have been secondary studies on the alignment of software engineering education and the software industry, the focus on software architecture education has been limited. However, a few studies have addressed this topic in detail. These studies are discussed below.

1. Garousi et al. (2020) conducted a review of studies investigating the challenges faced by software engineering graduates in their early careers due to a misalignment between the skills developed in college education and the needs of the software industry. The authors found that graduates often struggle to apply their knowledge to practical situations and lack essential skills required by the industry, such as communication, teamwork, and critical thinking skills.
2. da Cunha et al. (2018) conducted a systematic mapping study of Brazilian literature on studies related to games, tools, teaching methodologies, and the integration between disciplines to support education in software engineering. The study aimed to assess the status of advancements within the teaching-learning process of software engineering. The findings of this study provide valuable insights into the current state of software engineering education.
3. Brito et al. (2018) conducted an updated systematic mapping study on free projects in software engineering education to bridge the gap between academia and the

software industry. The study aimed to identify the current state of the pedagogical use of open-source software projects for teaching software engineering. The authors found a growing trend in the use of open-source software projects for teaching software engineering, but they also noted that this area of research has yet to reach maturity.

4. Groeneveld et al. (2019) conducted a literature review focusing on the importance of soft skills as a fundamental component to help students become good software developers. The study aimed to identify the primary soft skills essential for success in software development. The authors identified four primary soft skills: self-reflection, conflict resolution, communication, and teamwork. This study makes a valuable contribution to enhancing software engineering curricula by offering fresh perspectives on the connections between noncognitive abilities and teaching elements.
5. Qadir and Usman (2011) conducted a systematic mapping study to synthesize software engineering programs and curricula from various colleges worldwide. The study aimed to provide a broad overview of the software engineering education landscape. This systematic mapping study facilitated a comprehensive overview of the field by consolidating documented endeavors about the software engineering curriculum.

Two relevant studies focus on software architecture education reviews. The first study, proposed by Rodrigues and Werner (2009), aims to identify initiatives used in software architecture education. The study examines examples, techniques, software development tools and editors, books, and methodologies related to software architecture teaching. The study's findings highlight the prevalent use of design patterns across multiple educational initiatives.

Similarly, Oliveira et al. (2022) explore concepts associated with software architecture and integrate real-world project requirements, such as agile and continuous delivery, into software architecture education. While these studies share similarities with ours, our study reveals findings regarding strategies to align software architecture between educational and industry contexts.

These contributions offer researchers and educators guidance to improve their software architecture courses by incorporating strategies reported by the literature review. These strategies can bring architecture courses closer to the needs and conditions of the software industry.

## 4 Literature review process

### 4.1 Method

This study follows the mapping study method proposed by Petersen et al. (2008), which describes how to conduct a systematic mapping study in software engineering and provides guidelines. The method systematically organizes current knowledge into categories, classification, or taxonomy formats and summarizes its results related to the research questions. This approach requires less effort while providing a more coarse-grained overview of the field.

The essential process steps of the systematic mapping study are as follows (see Fig. 1): (1) defining the research questions, (2) searching for primary studies, (3) reviewing full-text articles based on inclusion criteria/exclusion, (4) classifying documents, and (5) extracting and mapping collected data. Each stage produces a result used as input for the following steps.

The following subsections present the process for preparing the mapping study. By following these steps, we can systematically organize and analyze the current knowledge in software engineering education and identify strategies to align software architecture education with industry needs.

## 4.2 Research questions

The research questions in this study aim to identify gaps in software architecture training research, including the training process followed by colleges, their learning strategies, and content. The research questions are as follows:

1. RQ1: How do software architecture courses incorporate training techniques and teaching methodologies to meet the needs of the software industry?
2. RQ2: What software architecture content is included in college courses?
3. RQ3: What are the key skills that a student in software architecture should develop during their training?
4. RQ4: What are the challenges that students encounter during software architecture training?
5. RQ5: What methods were used to validate the training experiences and achieve the proposed objectives?

Table 1 presents the questions and their motivation. By addressing these research questions, we can select, analyze, and categorize the information found in the study area. This approach allows us to identify areas where improvements can be made in software architecture education to better prepare students for success in the software industry.

## 4.3 Search strategy

A fundamental aspect of conducting a systematic literature mapping study is ensuring that research questions are well-constructed. The PICOC method helps describe

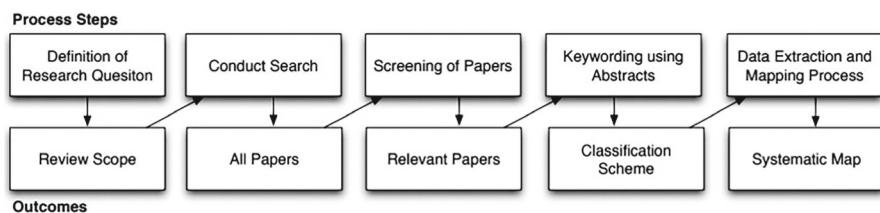


Fig. 1 Systematic mapping process proposed by Petersen et al. (2008)

**Table 1** Research questions used for systematic mapping

Research questions	Motivation	Potential answers
RQ1	Determine the strategies used to train software architects.	Collaborative work, open-source projects, case-based learning, games, project-based learning, case studies, etc.
RQ2	Determine the topics and learning outcomes addressed by the courses found.	Course contents and learning outcomes
RQ3	Determine the most essential technical and soft skills a software architect requires to perform well in the industrial sector.	Leadership, critical thinking, communication and teamwork, design thinking, and abstraction, among others.
RQ4	Determine the problems students face that are reported by the literature in software architecture.	Lack of experience and complexity.
RQ5	Evaluate how the proposed course was validated for its relevance.	Case study, experiment, Action Research, etc.

the five elements of a searchable question and stands for Population, Intervention, Comparison, Outcome, and Context.

To collect data from bibliographic databases systematically, we used the following search string based on the PICOC criteria (Souza et al., 2018):

(“EDUCATIONAL NEEDS” OR “KNOWLEDGE NEEDS” OR “DESIRED SKILLS” OR “ESSENTIAL COMPETENCIES” OR “KNOWLEDGE REQUIREMENTS” OR “SKILL REQUIREMENTS” OR “TRAINING” OR “TEACHING”) AND (“SOFTWARE ARCHITECTURES” OR “SOFTWARE ARCHITECTURE” OR “SOFTWARE DESIGN”) AND (“INDUSTRIAL SOFTWARE” OR “SOFTWARE INDUSTRY”) AND “COURSE”

We applied this query to the title, abstract, and keywords of six bibliographic databases: ACM Digital Library, EBSCO Services, IEEE Digital Library, Scopus, ScienceDirect, and Google Scholar (Ronchieri & Canaparo, 2019). Although EBSCO is not a technology-related database, we included it to incorporate relevant articles from reliable sources. We adjusted the query to the syntax of each search engine and applied it to studies published from January 2011 to March 2021.

4.4 Selection criteria for primary studies

Following the guidelines proposed by Petersen et al. (2008), we employed the subsequent inclusion and exclusion criteria to identify the most relevant studies for our research and answer the research questions.

**INCLUSION CRITERIA:** The inclusion criteria employed to identify relevant studies for our research and answer the research questions were as follows:

1. The papers are written in English, concentrating on software architects’ education to meet the industry’s requirements.

2. The papers are full-text publications in peer-reviewed journals, conferences, or symposiums.
3. The papers are based on empirical data and not solely on the authors' opinions.
4. The full text of the paper is accessible to readers from a reliable source.

#### EXCLUSION CRITERIA:

1. The primary study does not present a solution that contributes to software architects training that aligns with the needs of the industry.
2. Studies not available in full text.
3. Papers that display non-peer-reviewed material.
4. Secondary or tertiary studies that report and analyze the results of other investigations.
5. Duplicate publications by the same author.

We considered a study if it met each inclusion criterion and did not meet any of the exclusion criteria.

## 4.5 Execution stage

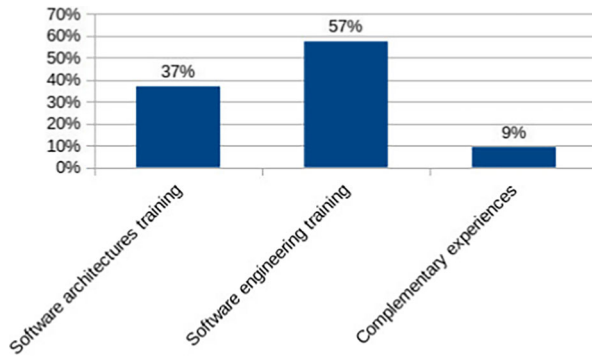
In phase 1 of our study, we applied the search string to five bibliographic databases and obtained 673 results. We then applied the inclusion and exclusion criteria to the abstracts, keywords, and titles of the articles, resulting in 78 “candidate primary studies” after eliminating duplicates. In phase 2, we further applied the selection criteria and identified 56 primary articles. In phase 3, we extracted data, characterized the studies, and answered the proposed research questions. We did not consider secondary studies in our information extraction, and a researcher expert in architecture issues used a second criterion in cases of doubt during the selection process.

Between 2011 and 2021, 56 experiences with software architecture training were reported. These experiences aimed to bridge the gap between college education and the needs of the software industry. We categorized these experiences into three groups: (i) experiences focused solely on software architecture training (37%); (ii) experiences focused on software engineering training that included software architectures in a global theme and the software development cycle, including analysis, design, implementation, testing, and deployment (57%); and (iii) complementary experiences that used strategies to support software patterns and architecture training (9%). Figure 2 depicts the number of experiences in each of these three groups.

Appendix A shows the primary articles.

The study identified four levels of software architecture training: undergraduate, postgraduate, continuing education, and industry training. Undergraduate training refers to college programs in computer science, software engineering, and related fields. Postgraduate training refers to master's programs while continuing education refers to university-based teaching that aims to link with the environment through training and educational programs. The goal is to educate individuals who wish to deepen





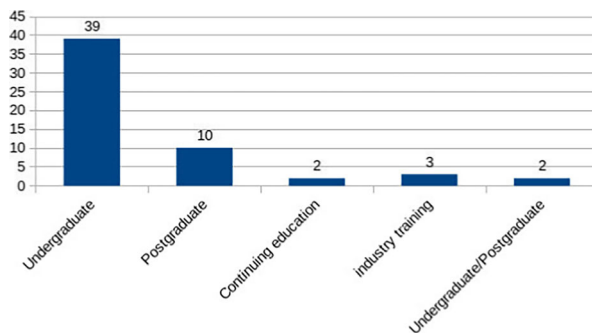
**Fig. 2** Experiences supporting issues of software architecture training

their knowledge and skills and keep up to date with the latest developments in their field. Industry training refers to training provided by companies to their employees through private courses aimed at improving their skills. Figure 3 depicts the distribution of experiences across each level of training, with most incidents reported in the literature corresponding to undergraduate courses (39 experiences, 70%).

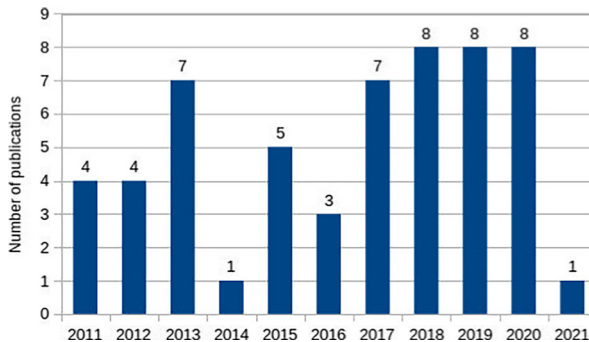
## 5 Results and analysis of results

Fig. 4 presents the annual distribution of publications about software architecture training in line with the needs of the software industry. In recent years, there has been a significant increase in publications, with the highest number occurring between 2018 and 2020 (8 per year, totaling 25, representing 44.4%). This trend indicates that teaching software architecture is an area of interest to the research community. 2014 had the smallest number of publications, with only one paper published. 2021, there is only one publication because the data collection was conducted until March.

Overall, our intention is to answer the research questions posed earlier.



**Fig. 3** Report of experiences at each level of training



**Fig. 4** Article publications per year focusing on the training of software architects

### 5.1 RQ1: How do software architecture courses incorporate training techniques and teaching methodologies to meet the needs of the software industry?

The literature review reveals various teaching strategies to align software architecture training courses with the needs of the software industry. These techniques can be classified into two categories: those used for training software architects and those used for teaching and learning methodologies. The former refers to strategies for recreating the industry's working environment in the classroom, while the latter refers to strategies teachers use to teach software architecture courses. Tables 2 and 3 provide a synthesis of these strategies.

In terms of training techniques, several strategies have been identified.

One approach to software architecture training is small project-based training. This approach involves developing one- or two-week-long projects focusing on a specific quality attribute, such as improving user interface usability, enhancing system performance, or strengthening security. This approach emphasizes practical exercises using modern web services and software frameworks rather than attempting to make superficial improvements on junior projects (Rupakhetti & Chenoweth, 2015). Some experiences have demonstrated educational strategies that use lightweight projects with public and free resources for collaborative and experimental learning.

These courses are designed using an iterative approach that involves planning, executing, evaluating, and improving. Projects are available on cloud-based platforms such as GoogleAppEngine, IBM Cloud Lite, and Dropbox. From a faculty perspective, public access platforms make it easier to evaluate student projects by allowing users to examine the runtime of project systems or components online [36]. The course progresses from designing a large system with its requirements and discussing design choices to a set of smaller weekly coding exercises that involve adding new features to the system using a specific design pattern. This approach provides students with a practical and hands-on learning experience (Chatley & Field, 2017).

In certain experiences, students were provided with the source code to analyze and evaluate the architectural design (Li, 2019). In other instances, projects utilized a simulated or actual client with a genuine need (Mohan et al., 2012). Some courses

**Table 2** Techniques in training software architects aligned with the needs of the software industry

Strategy Training techniques	References
Small project-based training	Rupakheti and Chenoweth (2015); Li (2020); Chatley and Field (2017); Li (2019)
Large project-based training	Mohan et al. (2012); Palacin-Silva et al. (2017) Wang (2011); May et al. (2018); Luukkainen et al. (2012); Hjelsvold and Mishra (2019) Zhang et al. (2020); Abdool and Pooransingh (2014); Sun et al. (2011); Péraire (2019) Matthies et al. (2018); Angelov and de Beer (2017); Chauhan et al. (2019); Giraldo et al. (2011) Khakurel and Porras (2020)
Open-Source Projects Training	Pinto et al. (2019); Silva et al. (2019); Hjelsvold and Mishra (2019); Van Deursen et al. (2017) Hu et al. (2018); Wei et al. (2020); Weerawarana et al. (2012); Meneely and Lucidi (2013)
Real clients or industry experts	Rupakheti and Chenoweth (2015); Abdool and Pooransingh (2014) Weerawarana et al. (2012)
Conferences featuring guests from the software industry sector	Wang (2011); de Beer and Angelov (2015); Zhang et al. (2020); Van Deursen et al. (2017) Li (2019); Ji and Song (2015); Collins (2020)
Field visits	Collins (2020)
Instructors and guest contributors in software engineering from the industry sector	Hu et al. (2018) Johns-Boast and Flint (2013)
Developing software under conditions faced in the industry	Mohan et al. (2012)
Global software development	Hjelsvold and Mishra (2019)
Knowledge and skills demanded in the industry	Ward et al. (2015); Joy and Renumol (2018); Sun et al. (2011)
Job advertisements to align curricula with industry needs.	Ward et al. (2015)
Industry-applied tools	Paez (2017)
Current state-of-the-art software architecture s	Zhang et al. (2020); Ji and Song (2015)
Design thinking	Palacin-Silva et al. (2017)

incorporated Design Thinking methods and agile practices into the project life cycle (Palacin-Silva et al., 2017).

**Large project-based training** involves involves medium and high-complexity projects with multiple requirements that focus on software architecture. This approach allows students to learn software architecture through a complete project, enabling them to see the results of their architectural design in a final product [68]. Software architecture and engineering courses are usually built on project-based learning and collaborative paradigms that simulate industrial environments.

**Table 3** Teaching methodologies in training software architects in line with the needs of the software industry

Strategy	References
Teaching methodologies	
Professors teach traditional classes in software architecture education	Wang (2011)
The flipped classroom approach in software architecture education	Lieh Ouh et al. (2020); Péraire (2019) Palacin-Silva et al. (2017); Péraire (2019)
Problem-based learning	Joy and Renumol (2018); Kiwelekar (2021); Paez (2017)
Case-based learning	Lieh and Irawan (2019); Kiwelekar (2021); Khakurel and Porras (2020) Lieh Ouh et al. (2020); Lieh and Irawan (2019); Li (2019) Ji and Song (2015); Capilla et al. (2020); May et al. (2018)
Online learning	Wendt et al. (2016); Schmidt and McCormick (2013)
Intelligent tutorial system	Lieh and Irawan (2019)

Some studies incorporate agile team development practices (Luukkainen et al., 2012), while others combine Global Software Engineering with Open-Source Development Projects (Hjelsvold & Mishra, 2019). These projects enable students to apply theoretical knowledge of software architecture to real-world systems (Zhang et al., 2020). Students learn to design software systems that meet stakeholder needs and provide a positive user experience by working on actual challenges (Sun et al., 2011; Péraire, 2019; Khakurel & Porras, 2020). This approach requires students to communicate and collaborate effectively within teams. Some courses even provide junior research assistants who serve as tutors in team meetings, offering advice and support (Matthies et al., 2018). Students also learn about creating architecture, the architect's role, and the documentation that should be used to support software architecture (Angelov & de Beer, 2017).

**Open-source projects training** enables students to interact with existing systems, factual issues, and actual software development teams to build high-quality working software. This approach provides students with a unique opportunity to gain experience only present in real-world scenarios, which can increase their abilities and self-confidence (Pinto et al., 2019). By working on open-source projects, students can make extensions, correct bugs, or analyze the architecture of existing systems. Listed below, we provide different alternatives to work with open-source projects.

- Experiences in teaching object-oriented modeling with UML (Unified Modeling Language) using open-source projects are reported by Silva et al. (2019).
- Students work with Moodle mobile and Moodle plug-in issues (Hjelsvold & Mishra, 2019).

- Students analyze open-source systems and learn from analyses conducted by other teams. Additionally, students create book chapters based on the architecture documentation of their system (Van Deursen et al., 2017).
- Open-source projects provide students with the opportunity to gain experience from common mistakes made by beginner developers, such as poor naming of variables and methods, comments between lines, lengthy methods, switch structures instead of polymorphism, no unit tests, and duplicated code (Hu et al., 2018).
- Some proposals suggest the creation of public code repositories containing actual projects from popular industry sectors. These repositories would serve as a reference for instructors to enhance the software architecture learning experience (Wei et al., 2020).
- Other initiatives propose that students develop a component, extension, or application within or on top of an existing open-source software project (Weerawarana et al., 2012).

Some software architecture courses **involve real clients or industry experts** who participate by providing requirements and evaluating demonstrations. The college may request candidate clients from companies, with most candidates being graduates from the same educational institutions (Rupakheti & Chenoweth, 2015). In other instances, each group of students has a mentor who acts as a client (Abdool & Pooransingh, 2014). Furthermore, some courses use industry software experts to assess class projects through demonstrations. In addition to assisting with the assessment workload, industry experts provide students with practical feedback based on current industry best practices (Weerawarana et al., 2012).

**Conferences featuring guests from the software industry sector** are valuable in software architecture education. Experienced software architects who have worked on large-scale system designs share their experiences in software architecture issues (Wang, 2011; Zhang et al., 2020; Van Deursen et al., 2017; Li, 2019). Collaborative efforts between academia and industry are essential, with many colleges having agreements with software development companies (de Beer & Angelov, 2015). Some courses offer seminars during the semester with guests from the industry on innovative topics [29]. The conferences often involve student feedback to plan future events (Collins, 2020).

**Field visits to software development companies** also provide students with valuable insights into the world of work. Through these visits, students can interact with developers, architects, and businesspeople and see how companies organize their projects, work teams, and time. This approach enables students to have early contact with the industry (Collins, 2020).

**Instructors and guest contributors in software engineering from the industry sector** can also provide valuable practical software engineering skills. They are often more qualified to teach practical skills than pure scholars and can give their teaching material a greater sense of significance (Hu et al., 2018).

In some cases, each team of students is assigned a mentor with industry and project experience who plays the role of the company's project manager (Johns-Boast & Flint, 2013).

Students in software architecture education learn the software development process by **developing software under conditions faced in the industry**. The development process is essential for organizing all team members' roles and activities (Mohan et al., 2012).

**Global software development** is another approach, where students form teams of diverse nationalities to develop software projects. They use Scrum and are familiar with Jira, Git, and Moodle applications. Global Software Engineering is combined with open-source projects to provide students with an experience of the software industry (Hjelsvold & Mishra, 2019).

Through projects, students **can apply knowledge and skills demanded in the industry and develop the essential skills** required for successful careers. The skills acquired in the courses are compared with the soft skills of the industry reported in the literature (Sun et al., 2011). Some experiences involve role-playing software projects and building industry skills as analysts, architects, designers, testers, and integrators (Joy & Renumol, 2018).

Some experiences in designing study plans for software architecture education use **job advertisements to align curricula with industry needs**. Job adverts publish required professional skills, enabling curricula that align with the industry's needs (Ward et al., 2015).

Other experiences focus on **using industry-applied tools** such as bug tracking, continuous integration, Heroku, and Vagrant (Paez, 2017). The use of virtual machines simplifies the installation and configuration of these tools. For software architecture, students should become familiar with modeling and DevOps tools.

To stay up to date with current software architecture practices, students learn using papers and other materials from the **current state-of-the-art software architecture** (Zhang et al., 2020). There are many tasks that students can work on Zhang et al. (2020). First, the teacher assigns a student to read a paper in software architecture. Each student reads and answers a set of questions, including the problem, the solution, contributions, results, and lessons learned, then the student makes a report, using slides.

Another task requires students to read articles in pairs and cooperate to create reading notes to avoid individual biases. The students select articles from recognized journals and conferences. In a third assignment, students conduct research in both academia and industry. They search for research groups and industrial companies in software architecture, rank the top ten teams or organizations, and explain the reasons for the ranking in the report..

**Design thinking** is an approach to solving complex problems by using creative methods. It emphasizes the importance of empathy between designers and users and encourages shared exploration of issues and solutions. During the ideation phase, software architects assess the feasibility of the solution considered, eliminating fruitless prototyping of solutions according to the current state of technologies (Palacin-Silva et al., 2017).

Appendix B shows techniques in training software architects versus primary studies. The most used strategies are large project-based training (12 citations), open-source projects training (8 citations), and small project-based training (7 citations).

The following are the strategies related to teaching methodologies.

**Professors teach traditional classes in software architecture education** Professors explain and share their knowledge with students in person (Wang, 2011; Lieh Ouh et al., 2020). These classes focus on teaching students how to design software systems that solve real-world problems, meet the requirements of stakeholders, and provide a positive user experience (Péraire, 2019).

**The flipped classroom approach in software architecture education** involves educators offering lectures or showing videos to students before the face-to-face session. During the face-to-face meeting, students organize discussions and exercises, study the source code, and understand how to implement concepts such as concurrency, security, principles, modifiability, persistence, event management, and error handling (Palacin-Silva et al., 2017).

In some experiences, students have found the flipped classroom approach to be much more effective and engaging than the traditional approach because it focuses on problem-solving (Joy & Renumol, 2018). The flipped classroom favors immersive, collaborative, and active learning experiences (Paez, 2017). Some courses have a blended approach that combines flipped classroom classes with traditional instruction (Péraire, 2019).

**Problem-based learning** is a strategy used in software architecture education where teams of students address architectural problems, and instructors play a minimal role without interfering in the discussion. As students explore issues, instructors function as facilitators and guide questions to bring them back to the main learning goal. For example, one subgroup explains a particular design solution, while the other subgroup understands another group's design solution and prepares questions for the actual presentation. Each student listens to the experiences and thinking of various students applied in a real-life scenario (Lieh & Irawan, 2019).

Using projects and problem-based learning encourages collaborative work (Kiwelekar, 2021) that examines the effect of the problem/project-based approach on students' skill gains (Khakurel & Porras, 2020).

**Case-based learning** is another approach where case studies are “real-life scenarios that allow students to analyze, apply taught concepts, and mitigate potential trade-offs within a realistic context” (Lieh Ouh et al., 2020). It focuses on the design and analysis of actual software projects for companies.

Under this approach, students work in teams within real-life scenarios. The instructors play a minimal role and do not interfere in the discussion. When students explore doubts, instructors function as facilitators and use guiding questions to bring them back to the main learning goal (Lieh & Irawan, 2019).

Teachers conduct case-based teaching in a flipped classroom setting. The cases include a client–server style ATM system, a service-oriented online vending system, an emergency monitoring system, a web-based automated management system, and software architecture design in real-time. Each student analyzes the case and completes the design tasks required for the [35] assignment within one week (Li, 2019).

Students work in teams of 5 or 6 members with specific roles assigned to each member. Two senior architects make the architectural design decisions, identify design problems, and capture architectural design decisions. Two cognitive architects challenge senior architects' decisions and raise concerns when risks or incomplete



information are present. One or two junior architects oversee understanding and modeling the captured decisions and provide a solution architecture (Capilla et al., 2020).

**Online learning** in software architecture education is delivered through two fundamental modalities: MOOCs (Massive Open Online Courses) and SPOCs (Small Private Online Courses).

Wendt et al. (2016) offer a Small Private Online Course (SPOC) on software design, and after completing the course, they apply a survey where participants value the course content. The industry will receive SPOC courses for staff training.

Schmidt and McCormick (2013) describe the experiences of producing and delivering a MOOC entitled “Pattern-Oriented Software Architecture for Concurrent and Networked Software.” They also discuss “the broader implications of MOOCs on lifelong learning and their role in improving the quality and productivity of software professionals in academia and industry” (Schmidt & McCormick, 2013).

Another approach to online learning in software architecture education is the **Intelligent Tutorial System**, which provides personalized learning pathways by tracking each student’s progress. In Thevathayan and Hamilton (2017), students interact with an Intelligent Tutorial System (ITS) that meets varied needs by enabling different learning modes. An ITS works with multiple-choice questionnaires that pose UML and design problems. Crafting questions to show specific concepts or patterns is a challenging task. One advantage of cloud-based ITS is that the developed courseware can be shared with other institutions across the globe.

Appendix C shows teaching methodologies versus primary studies. The most used strategies are case-based learning (6 citations) and flipped classrooms (4 citations).

The literature review presents courses to train software architects with the necessary skills to enter the job market. Each experience explores one or more teaching strategies from a pedagogical and technological perspective. However, the review also highlights ineffective teaching strategies in training software architects. Table 4 provides a summary of these ineffective strategies.

## 5.2 RQ2: What software architecture content is typically included in college courses?

This question involves two perspectives: identifying the topics covered in the literature and aligning them with the learning goals of the courses. Table 5 displays the learning goals identified through systematic mapping. Although all these courses focus on software architecture, they have different approaches. Courses can be either basic or advanced and may be based on existing projects or project formulations.

Table 6 presents the topics covered in the literature. As with the learning goals, the content varies from one course to another. However, the literature does not provide specific information, such as topics and subtopics, which makes it challenging to conduct a more comprehensive examination of the content. Nevertheless, the reported experiences share common content, including software architecture fundamentals, architecture patterns, quality attributes, design principles, design patterns, analysis, design, and architecture evaluation.



**Table 4** Strategies that are not effective for training software architects

Category	Strategy	Ref
Scope	The literature review shows that a single course covering software architecture and design is time-consuming.	Rupakheti and Chenoweth (2015)
	Dealing with all the software engineering in one or two courses takes time and effort.	
Scope	Combining the reading of scientific articles with the development of a course project is not convenient since it does not allow students to focus on the project’s development.	Hjelsvold and Mishra (2019)
Project	Working with projects that do not motivate students is an inadequate strategy.	Van Deursen et al. (2017)
	Instead, they could choose the project that encourages them.	
Pedagogy	Using classroom time to teach the abstract theory of software architecture design makes students uninterested.	Li (2019)
	The teacher can promote active learning through workshops, case studies, and debates.	
Resources	The case studies of textbooks are limited in size and inappropriate for teaching software architectures.	Li (2019)
Industrial context	Traditional lectures and “toy” projects are unsuitable in industrial or business training because students already have practical experience in large software projects. They need more time to present, compare, and practice various architectural theories. The examples and exercises should focus on existing software systems developed by the companies.	Ciancarini et al. (2016)

In Appendix D, we can see that the most cited topics include architecture patterns (6 citations), quality attributes (3 citations), design principles (3 citations), analysis, design, and evaluation of software architecture (3 citations), and design patterns (3 citations).

**5.3 RQ3: What are the key skills that a student in software architecture should develop during their training?**

Tables 7 and 8 provide a summary of the technical and soft skills that software architects need to develop, as reported in the literature. In this section, we will discuss the soft skills of software architects.

One of the key skills that software architects need is **effective communication**. This includes the ability to speak, write, and present ideas in a clear and concise manner, especially when dealing with complex problems that require a straightforward design solution. Various sources in the literature emphasize the importance of communication

**Table 5** Learning Goals of the courses reported by the literature review

No	Learning goal	References
1	Defining and explaining central concepts of software architecture and using and describing design/architecture patterns, methods for designing software architectures, methods/techniques for achieving software qualities, and methods for documenting and evaluating software architecture.	Wang (2011)
2	Teaching a set of distributed architectural patterns.	Li (2020); Backert et al. (2020)
3	Fixing architectural problems in existing systems.	Rupakheti and Chenoweth (2015)
4	Analyzing and designing software architecture and gaining relevant experience in large-scale systems.	Zhang et al. (2020)
5	Reviewing software system architectures and relating software engineering methods to design complex software-intensive systems.	Ciancarini et al. (2016)
6	Teaching students modern software architecture methods and leading students to grow into good system architects within 5 or 8 years after graduation.	Ji and Song (2015)
7	Explaining how quality attributes affect the system architecture design.	Ji and Song (2015)
8	Showing the ability to be successful in a real-world software design experience.	Mohan et al. (2012)

**Table 6** Content of the courses reported by the literature review

No	Topic	References
1	Architecture patterns	Lieh Ouh et al. (2020); Van Deursen et al. (2017); Ji and Song (2015); Capilla et al. (2020); Li (2019)
2	Quality attributes	Lieh Ouh et al. (2020); Lieh and Irawan (2018); Wendt et al. (2016)
3	Case studies on the web, mobile, and cloud applications.	Lieh Ouh et al. (2020)
4	Fundamentals of software architectures: principles and concepts	Lieh and Irawan (2018); Ji and Song (2015)
5	Architectural views	Van Deursen et al. (2017)
6	Principles of design	Lieh and Irawan (2019); Van Deursen et al. (2017); Ji and Song (2015)
7	Product lines	Van Deursen et al. (2017)
8	Technical debt	Van Deursen et al. (2017)
9	Analysis, design and evaluation of software architecture	Chauhan et al. (2019); Capilla et al. (2020)
10	Design patterns	Capilla et al. (2020); Ji and Song (2015); Wendt et al. (2016)
11	Software modeling and use of UML diagrams	Wendt et al. (2016)
12	Refactoring	Wendt et al. (2016)

**Table 7** Soft skills of the software architect

No	Soft skills	References
1	Effective communication skills	Rupakheti and Chenoweth (2015); Silva et al. (2019) Mohan et al. (2012); Hjelmsvold and Mishra (2019); Backert et al. (2020); Johns-Boast and Flint (2013) Khakurel and Porras (2020); Sun et al. (2011); Péraire (2019); Pratheesh and Devi (2013) Desai et al. (2012); Weerawarana et al. (2012); Van Deursen et al. (2017); Hu et al. (2018) Lee et al. (2015); Jarzabek (2013); Wei et al. (2020)
2	Teamwork	Li (2020); May et al. (2018); Lieh and Irawan (2019); Mohan et al. (2012) Hjelmsvold and Mishra (2019); Zhang et al. (2020); Johns-Boast and Flint (2013); Palacin-Silva et al. (2017) Khakurel and Porras (2020); Sun et al. (2011); Péraire (2019); Desai et al. (2012); Joy and Renumol (2018) Angelov and de Beer (2017); Li (2019); Giraldo et al. (2011); Lee et al. (2015) Jarzabek (2013)
3	Leadership and negotiation	Abad et al. (2019); Lieh and Irawan (2019); Rupakheti and Chenoweth (2015)
4	Artistic skills	Rupakheti and Chenoweth (2015); Wei et al. (2020)

**Table 8** Technical skills of the software architect

No	Technical skills	References
1	Being familiar with multiple technologies	Rupakheti and Chenoweth (2015); Palacin-Silva et al. (2017); Wei et al. (2020)
2	Understand the domain in which a system will operate	Rupakheti and Chenoweth (2015); Wei et al. (2020)
3	Analytical skills	Rupakheti and Chenoweth (2015); Lieh Ouh et al. (2020); Lieh and Irawan (2018); Joy and Renumol (2018) Angelov and de Beer (2017); Li (2020); Wei et al. (2020)
4	Design, model, analyze, and evaluate software architectures	Lopes et al. (2019)  Silva et al. (2018); Mohan et al. (2012); Zhang et al. (2020); Lieh and Irawan (2018); Desai et al. (2012) Van Deursen et al. (2017); Schmidt and McCormick (2013); Ciancarini et al. (2016); Jarzabek (2013)
5	Research abilities	Li (2020); Zhang et al. (2020)
6	Minimum programming skills	Lieh and Irawan (2018); Li (2020)
7	Architectural Decision-Making	Lieh Ouh et al. (2020); Angelov and de Beer (2017)
8	Systemic thinking	Wei et al. (2020)

skills for architects (Rupakheti & Chenoweth, 2015; Silva et al., 2019; Hjelsvold & Mishra, 2019; Johns-Boast & Flint, 2013; Sun et al., 2011; Péraire, 2019; Pratheesh & Devi, 2013; Desai et al., 2012; Weerawarana et al., 2012; Lee et al., 2015; Jarzabek, 2013; Hu et al., 2018; Mohan et al., 2012; Backert et al., 2020).

Moreover, communication skills are crucial for students pursuing a career in software architecture (Hu et al., 2018). It is vital to ensure that students develop fundamental communication skills, such as reading, writing, listening, speaking, and comprehension (Mohan et al., 2012).

Software architects engage and communicate with experts in various disciplines to effectively conduct their role. Understanding the terminology, common problems, needs, and diverse cultural approaches (including ways of working and thinking) within these disciplines is important for them (Backert et al., 2020).

Effective communication is crucial for architects, both externally (with clients) and internally (with team members) (Khakurel & Porras, 2020). Architects need to have skills in communicating with stakeholders and achieving consensus on decisions (Van Deursen et al., 2017). Leadership skills are essential for architects to lead, present, negotiate, and justify their designs and decisions (Lieh & Irawan, 2019). They need to listen to and solve problems with design implementers (Wei et al., 2020).

**Teamwork** is another critical skill for software architects. They collaborate closely with other members of the development team, such as programmers (May et al., 2018; Mohan et al., 2012; Hjelsvold & Mishra, 2019; Zhang et al., 2020; Johns-Boast & Flint, 2013; Palacin-Silva et al., 2017; Khakurel & Porras, 2020; Sun et al., 2011; Péraire, 2019; Desai et al., 2012; Joy & Renumol, 2018; Li, 2019; Giraldo et al., 2011; Lee et al., 2015; Jarzabek, 2013). Learning about architecture involves intensive teamwork (Li, 2020), and the team should analyze and make architectural decisions together (Angelov & de Beer, 2017).

**Leadership and negotiation skills** are crucial for software architects to lead, present, negotiate, and justify their architectural designs and decisions (Abad et al., 2019; Lieh & Irawan, 2019; Rupakheti & Chenoweth, 2015). These skills include making independent decisions, taking the initiative, demonstrating independent judgment, being influential, and commanding respect (Bass et al., 2012).

Artistic skills are also necessary for software architects to create simple designs that are easy to understand and solve complex problems (Rupakheti & Chenoweth, 2015; Wei et al., 2020). Artistic skills involve knowledge of unique design techniques and tools, how to design complex multi-product systems, object-oriented analysis and design, UML diagrams, and UML analysis modeling (Bass et al., 2012).

In addition to soft skills, software architects possess a range of technical skills. These are explained below.

**Being familiar with multiple technologies** allows them to choose the appropriate technology for the project. While software architects do not have to be technology experts, keeping up to date with the latest technology trends is important (Rupakheti & Chenoweth, 2015; Palacin-Silva et al., 2017; Wei et al., 2020).

As a software architect, it is crucial to **understand the domain in which a system will operate**. The domain means comprehending the business environment in which the software will function to design and develop an appropriate solution. It involves understanding the specific business processes, rules, and operations and the needs

and requirements of users and stakeholders. This knowledge is essential in designing an architecture that aligns with the business goals and objectives (Rupakhetti & Chenoweth, 2015; Wei et al., 2020).

**Analytical skills** are crucial for software architects to understand the problem, identify potential root causes, and make informed decisions for the project (Rupakhetti & Chenoweth, 2015; Lieh Ouh et al., 2020; Joy & Renumol, 2018; Angelov & de Beer, 2017; Li, 2020). Architects need to identify the root causes of high-level problems in existing designs, such as system performance issues or security vulnerabilities (Wei et al., 2020).

Another critical skill for software architects is the **ability to design, model, analyze, and evaluate software architectures** (Van Deursen et al., 2017; Lopes et al., 2019; Silva et al., 2018; Zhang et al., 2020; Lieh & Irawan, 2018; Desai et al., 2012; Jarzabek, 2013). Architects need to be adept at applying patterns and frameworks to build high-quality applications (Schmidt & McCormick, 2013).

Designing the architecture of complex, large-scale software systems with numerous requirements and millions of lines of code requires exceptional modeling and abstraction skills (Ciancarini et al., 2016).

In addition to technical skills, software architects **need solid research abilities to manage and coordinate** all elements of a successful application project. Research skills enable architects to understand complex situations and solve problems (Li, 2020). Before designing any software solution, architects conduct research to understand the business requirements and needs. This includes conducting interviews, discussions, and analyses with users and stakeholders to understand what is expected from the system. Research enables architects to capture requirements accurately and define an architecture that meets business needs (Zhang et al., 2020).

Although software architects do not need to be programming experts, they **should possess minimum programming skills**. Architectural roles typically require software design and programming experience. Architects need to know enough about programming to communicate effectively with developers (Lieh & Irawan, 2018; Li, 2020).

**Effective architectural decision-making requires** a software architect to gather and analyze relevant information to make informed decisions. This involves understanding system requirements, technical constraints, business needs, and other relevant factors. By analyzing this information, architects can evaluate different options and make decisions based on sound data. In situations with uncertainty, time pressure, or limited exploration of alternatives, the ability to make informed design decisions is critical to success (Lieh Ouh et al., 2020).

**Systemic thinking** is a way of understanding and analyzing interactions between variables within a system or multiple subsystems, expressed in terms of feedback. This approach seeks to analyze these interactions in an orderly and comprehensive manner, considering all interrelated elements. Software architects need to possess systemic and holistic thinking abilities to approach problems from multiple perspectives (Wei et al., 2020).

Appendix E, Appendix E illustrates that common soft skills for software architects are communication (18 citations) and teamwork (15 citations). These skills are

essential for effective collaboration and communication with team members and stakeholders.

In Appendix F, the common technical skills for software architects are building and assessing architectures (9 citations), analytical skills (5 citations), and knowledge of multiple technologies (4 citations).

#### 5.4 RQ4: What are the challenges that students encounter during software architecture training?

This question aims to identify the challenges that arise during the software architecture training process, as reported in the literature. Table 9 presents the findings of this research. Below, we outline each difficulty.

1. *Lack of real-world experience among instructors.* To be effective, instructors should possess a diverse range of real-world experience. However, many instructors have academic backgrounds and may have limited experience designing large-scale projects (Rupakheti & Chenoweth, 2015).
2. *The abstract and ambiguous nature of software architecture poses* a significant challenge for students. Concepts such as design principles, offsets, architectural patterns, and product lines can be difficult to grasp (Zhang et al., 2020; Li, 2020; Sun et al., 2011; Van Deursen et al., 2017). Unlike programming, there are no clear-cut right or wrong solutions in software architecture. One of the main challenges of teaching software architecture is making these abstract concepts more accessible to students. For beginners, this requires a shift in mindset from concrete software programming to abstract architectural design. Architects used to write code and achieving deterministic results adapt to structuring components in a diagram and justifying their decisions based on trade-offs since perfect solutions do not exist. Furthermore, environmental aspects and principles may vary based on students' experiences and backgrounds, making software architecture solutions abstract and challenging to understand (Lieh & Irawan, 2019).
3. *Difficulty involving clients and industry experts is* a common challenge in academic environments (de Beer & Angelov, 2015). These individuals often have busy schedules, making it challenging to involve them in software architecture training. (de Beer & Angelov, 2015).
4. *Teaching software architecture requires going beyond traditional teaching methods* to account for the complexity of social interactions and collaborative software development in a real-world environment (Pinto et al., 2019). Unlike traditional college education, which focuses on imparting knowledge, software architecture requires the ability to apply knowledge effectively to deliver results (Pillutla & Alladi, 2013; Pratheesh & Devi, 2013; Van Deursen et al., 2017). Traditional lectures are not motivating for software architecture students (Paez, 2017).
5. *Small and simple projects are insufficient for providing students with practical experience.* To teach software architecture effectively, complex problems with multiple quality attributes and an appropriate context are necessary (Galster & Angelov, 2016). Class projects are often “toy projects” that lack the scope and maturity necessary for actual software development (Pinto et al., 2019). Academics

**Table 9** Challenges to training software architects aligned with the software industry

No	Challenge	References
1	Instructors' real project experience	Rupakheti and Chenoweth (2015)
2	Abstract and fuzzy nature	Li (2020); Zhang et al. (2020); Lieh and Irawan (2018)
3	Difficulty in involving clients and industry experts	Sun et al. (2011); Van Deursen et al. (2017); Lieh and Irawan (2019); Li (2019); Kiwelekar (2021)
4	Going beyond the traditional way of teaching	de Beer and Angelov (2015)
5	Small and simple projects do not provide a real practical experience	Pinto et al. (2019)
6	Lack of involvement in open-source projects	Chatley and Field (2017); Silva et al. (2019); Galster and Angelov (2016)
7	Each student required personal monitoring	Silva et al. (2019); Hjelsvold and Mishra (2019)
8	Modern practices developed in the industry take time to reach academia	Thevathayan and Hamilton (2017)
9	Teaching software architectures is still a difficult task	Thevathayan and Hamilton (2017)
10	Courses must adapt to changes	Palacin-Silva et al. (2017); Abad et al. (2019)
11	Finalized projects intimidate students	Angelov and de Beer (2017); Wei et al. (2020)
12	Gaps between the classroom and the industry	Matthies et al. (2018)
13	Clarity with agile methodologies	Weerawarana et al. (2012)
14	Prerequisites required	Joy and Renumol (2018)
		Angelov and de Beer (2017)
		Li (2019)

and researchers do not typically work on projects in industrial settings, making it challenging to use industry-developed projects in the classroom or interact with companies in the educational context (Silva et al., 2019).

6. **Lack of involvement in open-source projects.** Open-source repositories offer benefits for learning software architectures. However, instructors need to gain experience in this type of project (Silva et al., 2019). Working on open-source projects is challenging for teachers and their students, as they require in-depth knowledge and experience in technology. A learning stage is necessary before the implementation of tasks (Hjelsvold & Mishra, 2019).
7. **Personalized monitoring** is necessary for each student. Like a piano student, software architecture students require personalized attention from their teachers. However, this can be challenging for large teams (Thevathayan & Hamilton, 2017).

8. *Modern methods developed in the industry may take time to reach academia*, resulting in classroom teachings that may soon become outdated (Thevathayan & Hamilton, 2017).
9. *Teaching software architectures is a challenging task* that requires educators to balance quality, scope, depth, applicability, soft and technical skills, and individual and collaborative learning (Palacin-Silva et al., 2017). Learning requires a realistic context, teamwork, adequate complexity, and practical training (Angelov & de Beer, 2017). Educators need to find the right balance between “realism” and classroom control for software architecture students (Abad et al., 2019). Most architecture courses focus on theoretical concepts or cannot immerse students in complex industrial-level projects unless instructors come from the industry sector (Wei et al., 2020).
10. *The need for courses to adapt to changes* is crucial in providing students with the best possible education. Universities should strive to improve the course learning experience and update course contents to meet changing requirements (Matthies et al., 2018). Software architecture classes should be adjustable and regularly adjust to fluctuations in the business, technological breakthroughs, and student needs so that students can get the most precise and current instruction conceivable.
11. *Completed project scenarios are helpful in training software architecture, but they can also intimidate students* who may feel compelled to make changes. Instead, students prefer new projects that are more accessible and enable them to understand the requirements and have complete control over the software’s architecture, design, and structure (Weerawarana et al., 2012).
12. There is a significant *gap between students’ skills and the expectations of industry managers or hiring personnel*. This gap prevents students from getting job offers or meeting the expectation level of the software industry. There is a knowledge gap between the theory learned in the classroom and the actual requirements demanded by the industry (Joy & Renumol, 2018).
13. *There is a need to clarify how to work with software architecture in agile methodologies*. In agile methods, it is not always clear when to create the architecture, the architect’s role, how the architecture documentation should be, the methods and activities involved, and the value and costs associated with them (Angelov & de Beer, 2017).
14. *Activities in the software architecture stages require students to apply knowledge* from previous courses, such as networking, databases, operating systems, distributed systems, and software engineering. Careful consideration and application of this knowledge are necessary (Li, 2019).

Appendix G reveals that the most reported difficulties in the literature are the abstract and ambiguous nature of software architecture (8 citations), lack of real practical experience in small and simple projects (4 citations), the need to go beyond traditional teaching methods when teaching software architecture (5 citations), and the challenging nature of teaching software architectures (4 citations). The abstract and ambiguous nature of software architecture is the most discussed obstacle, making the teaching of this subject a considerable undertaking.



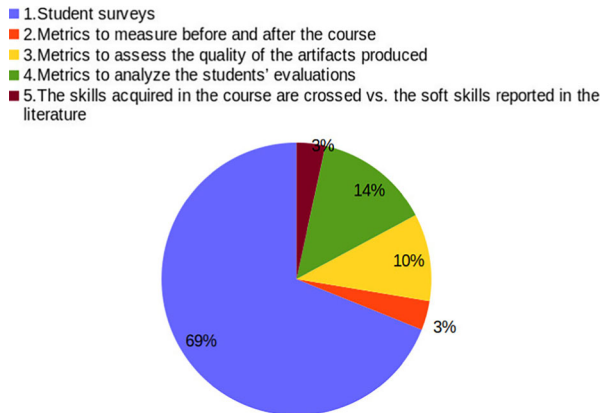
### 5.5 RQ5: What methods were used to validate the training experiences and achieve the proposed objectives?

Each experience presented in the literature details a specific approach to training students in software architectures through the application of teaching strategies. Each approach requires a course design and subsequent implementation. After these experiences have been conducted, it is essential to understand the effectiveness of the teaching strategies employed.

Various methods have been proposed to determine the effectiveness of the teaching-learning processes. Table 10 summarizes the validation methods used for courses. Surveys are often administered to students at the end of the course to measure satisfaction using the Technology Acceptance Model (TAM). Another technique involves comparing satisfaction with the proposed course to that of a traditional course (with studies using the Kruskal-Wallis test for data verification). A third technique measures

**Table 10** Validation methods of the proposed courses and their usage percentages

No	Validation method	References
1	Student surveys.	Rupakheti and Chenoweth (2015); Li (2020); Pinto et al. (2019) Chatley and Field (2017) Lopes et al. (2019); Lieh and Irawan (2019); Meneely and Lucidi (2013); Lieh Ouh et al. (2020) Lieh and Irawan (2018); Palacin-Silva et al. (2017) Péraire (2019); Wendt et al. (2016); Matthies et al. (2018) Joy and Renumol (2018); Angelov and de Beer (2017) Van Deursen et al. (2017); Li (2019); Ciancarini et al. (2016) Paez (2017); Giraldo et al. (2011)
2	Metrics to measure before and after the course.	May et al. (2018)
3	Metrics to measure the quality of the artifacts produced.	Silva et al. (2018) Portela et al. (2017); Joy and Renumol (2018)
4	Metrics to analyze the students' evaluations.	Johns-Boast and Flint (2013) Li (2019); Ciancarini et al. (2016); Ji and Song (2015)
5	The skills acquired in the course are crossed vs. the soft skills reported in the literature.	Khakurel and Porras (2020)



**Fig. 5** Validation methods of the courses reported in the literature

the quality of the artifacts produced, such as requirements specifications, analysis models, design models, and tests. Fourth, evaluations of the students during the course are analyzed to understand the learning process and objectives. Finally, a fifth technique involves analyzing and comparing the skills achieved in the course with those reported in the literature.

Appendix H illustrates the validation techniques used for software architecture courses compared to primary studies. The most used techniques were student surveys (20 citations), metrics to analyze student evaluations (4 citations), and metrics to assess the quality of artifacts produced (3 citations). Most experiences utilized the survey technique (69%, as seen in Fig. 5) because it allows for the collection of students' perceptions and analysis of their comments. However, we recognize that this technique may reduce objectivity in validation, as students may not be able to assess the quality of the course.

We have analyzed the validation techniques in conjunction with teaching strategies from RQ1, and the results are shown in Fig. 6, which presents a bubble diagram with three dimensions. The Y-axis represents teaching strategies, the X-axis represents course validation techniques, and the width of the bubble corresponds to the number of experiences that use the intersected strategy and validation. The diagram reveals that the most used technique is crossing short projects with validation through surveys (six experiences). Additionally, certain strategies have no associated validation techniques, such as current state-of-the-art software architecture.

## 6 Discussion

This systematic mapping analyzed the strategies, contents, skills, challenges, and course validation methods for training students in software architecture from selected studies. The papers were reviewed across six databases, and documents on teaching software architecture from January 2011 to March 2021 were examined. From this analysis, 56 primary studies were selected for further data extraction and

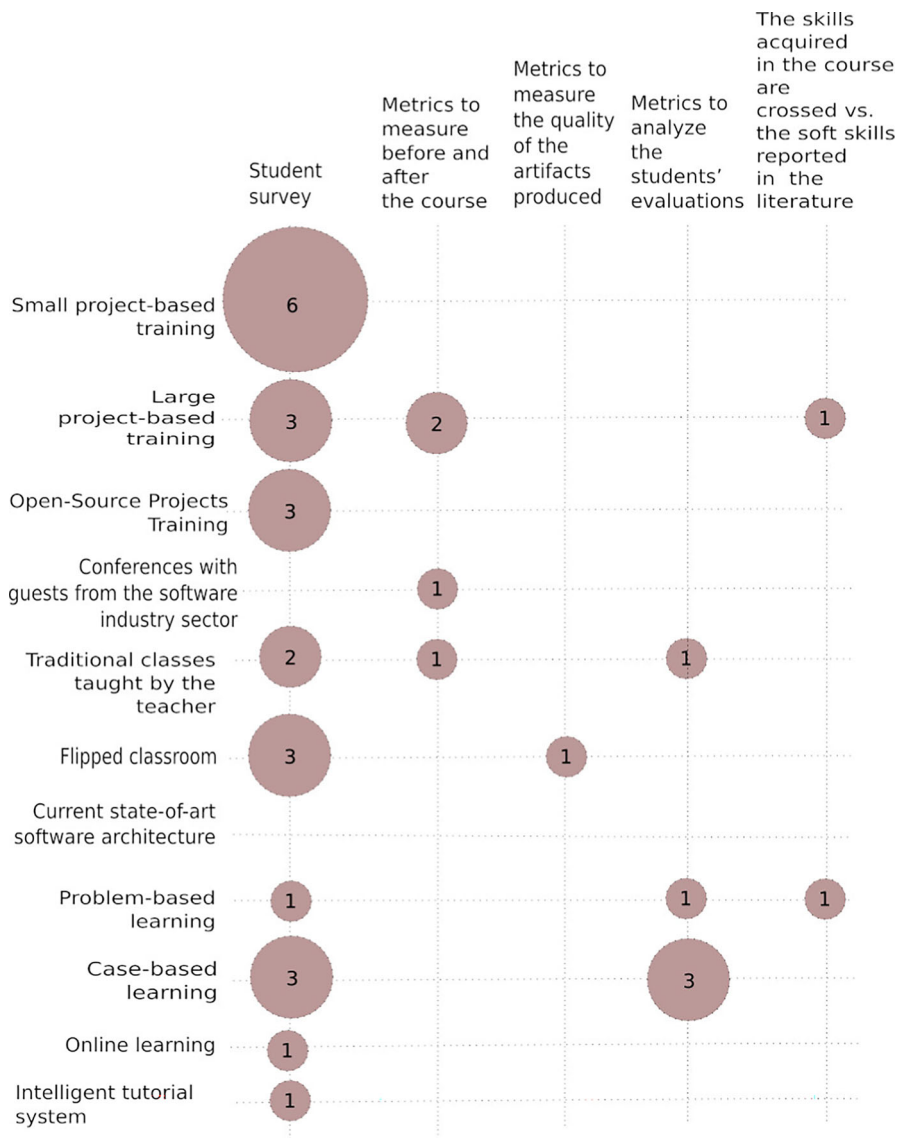


Fig. 6 Teaching Strategies Vs. validations techniques

analysis, providing a comprehensive understanding of how the training process for software architecture is being implemented in colleges, considering the needs of the industry. The main highlights of this systematic mapping are summarized as follows:

1. Teaching software architecture in computer science and software engineering programs is a pertinent topic in the scientific community, with a consistent number of publications over the last decade. The most publications were recorded between 2018 and 2020, with an average of eight publications per year. The countries most

interested in this topic are the USA, India, China, and Brazil, which accounted for 44% of total publications over the last ten years.

2. Teaching software architectures presents many challenges for educators. The most reported difficulties in the literature include the abstract and ambiguous nature of software architecture (8 citations), the lack of real practical experience in small and simple projects (4 citations), the need to go beyond traditional teaching methods when teaching software architecture (5 citations), and the challenging nature of training software architectures (4 citations).
3. Despite the fact that the most significant challenge reported is the abstract and ambiguous nature of software architectures, no training strategies have been developed to address this challenge.
4. The literature review presents various teaching strategies to align software architecture training courses with the needs of the software industry. These techniques have been classified into two categories: training software architects and teaching methodologies. The most used strategies for training software architects are large project-based training (12 citations), Open-Source Projects training (8 citations), and small project-based training (7 citations). Case-based learning (6 citations) and flipped classrooms (4 citations) are the most used strategies related to teaching methodologies. Despite these strategies, no complete solution addresses the challenges of training software architects. Educators still struggle to train software architects and rely on their experience or trial-and-error testing. New educators require guidelines or patterns for designing software architecture curricula.
5. While the teaching experiences use diverse training strategies, there is a gap in integrating multiple strategies to enhance the teaching process.
6. Table 3 shows two experiences related to online learning, whereas only one reference pertains to intelligent tutorial systems. Exploring these two distinct lines of inquiry could present an intriguing research context for software architecture training.
7. Software architecture students need to develop their abilities gradually over time, and it is necessary to agree on the relevant skill sets. While the range of technical and soft abilities is wide, the two most often mentioned soft skills are communication (cited 18 times) and collaboration (cited 15 times).
8. Tables 7 and 8 present the soft and technical skills of the software architect. However, it is essential to investigate and study the teaching strategies that facilitate the development of these skills.
9. The reported experiences cover fundamental topics such as software architectures, architecture patterns, design patterns, quality attributes, design principles, analysis, design, and evaluation of architectures.
10. The most used validation techniques for software architecture courses are student surveys (20 citations), metrics to analyze student evaluations (4 citations), and metrics to measure the quality of artifacts produced (3 citations).
11. Figure 6 shows that surveys are the most utilized validation techniques, but when using surveys, it is necessary to consider students may not offer an objective analysis of the quality of the course. Validations should reflect industry-relevant skills from real-world needs.

12. Teaching software architectures is a current research topic that aligns with the needs of the industry. While researchers have made noteworthy contributions, the complexity of architecture, the industry's expectations, the transformation of the architecture concept, the emergence of recent technologies, and the quality criteria mean that many facets of this topic still need to be explored. Some studies provide insight into topics such as the architect's role, lessons learned, and details on what and how to teach a software architecture course.

The results of this systematic mapping can help researchers and educators understand the challenges of teaching software architecture, propose new courses that build skills in architecture aligned with the needs of the industry, and improve existing courses by incorporating and combining strategies reported in the literature that have been effective under certain conditions.

## 7 Conclusions

In this study, we conducted a systematic mapping to investigate teaching experiences in software architectures. We selected 56 primary studies published from January 2011 to March 2021 and classified the experiences into three groups: (i) experiences focused on teaching software architectures, (ii) experiences focused on teaching software engineering, and (iii) other experiences with interesting strategies that support the teaching of software patterns and architectures.

We discovered that teaching strategies focus on aligning and providing feedback on software architecture training courses to meet the needs of the software industry, the software architecture course contents offered by colleges, and the skills associated with the software architect role.

Despite two decades of teaching software architecture, challenges remain, making this area a challenging research topic. Software architecture education requires more maturity due to the complexity and knowledge required. The findings of this study will be beneficial for researchers and educators who aim to overcome the challenges of teaching software architectures.

## Appendix

### A. Primary studies

See Table 11 Primary studies.

**Table 11** Primary studies

No	Authors	Title	Name of the conference or journal	References	Category
S1	Chandar R. Rupakheti, Stephen V. Chenoweth	Teaching software architecture to undergraduate students: an experience report	International Conference on Software Engineering 2015	Rupakheti and Chenoweth (2015)	Teaching of software architecture
S2	Alf Inge Wang	Extensive Evaluation of Using a Game Project in a Software Architecture Course	ACM Trans. Comput. Educ	Wang (2011)	Teaching of software architecture
S3	Zheng Li	Using public and free platform as a service (PaaS) based lightweight projects for software architecture education	Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training	Li (2020)	Teaching of software architecture
S4	Li Zhang, Yanxu Li, Ning Ge	Exploration on Theoretical and Practical Projects of Software Architecture Course	15th International Conference on Computer Science and Education, ICCSE 2020	Zhang et al. (2020)	Teaching of software architecture
S5	Eng Lieh Ouh, Benjamin Kok Siew Gan, Yungahns Irawan	Did our Course Design on Software Architecture meet our Student's Learning Expectations?	2020 IEEE Frontiers in Education Conference (FIE)	Lieh Ouh et al. (2020)	Teaching of software architecture
S6	Matthias Backert, Thomas Blum, Rüdiger Kreuter, Frances Paulisch, Peter Zimmerer	Software Curriculum @ Siemens — The Architecture of a Training Program for Architects	2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE T)	Backert et al. (2020)	Teaching of software architecture

**Table 11** continued

No	Authors	Title	Name of the conference or journal	References	Category
S7	Ouh Eng Lieh, Yunghans Irawan	Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course	2018 IEEE Frontiers in Education Conference (FIE)	Lieh and Irawan (2018)	Teaching of software architecture
S8	Shahani Markus Weerawarana, Amal Shehan Perera, Vishaka Nanayakkara	Promoting creativity, innovation and engineering excellence	Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALF) 2012	Weerawarana et al. (2012)	Teaching of software architecture
S9	Samuil Angelov, P de Beer	Designing and Applying an Approach to Software Architecting in Agile Projects in Education	Journal of Systems and Software	Angelov and de Beer (2017)	Teaching of software architecture
S10	Arie Van Deursen, Maurício Aniche, Joop Aué, et al	A Collaborative approach to teaching software architecture	Proceedings of the Conference on Integrating Technology into Computer Science Education, ITICSE	Van Deursen et al. (2017)	Teaching of software architecture
S11	Ouh Eng autLieh, Yunghansors Irawan	Teaching adult learners on software architecture design skills	Proceedings - Frontiers in Education Conference, FIE	Lieh and Irawan (2019)	Teaching of software architecture
S12	Douglas C. Schmidt, Zach McCormick	Producing and delivering a MOOC on pattern-oriented software architecture for concurrent and networked software	SPLASH 2013 - Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, and Applications: Software for Humanity	Schmidt and McCormick (2013)	Teaching of software architecture

Table 11 continued

No	Authors	Title	Name of the conference or journal	References	Category
S13	Weigang Li	Teaching Reform and Practice of Software Architecture Design Course under the Background of Engineering Education	Proceedings of the 2019 International Conference on Advanced Education, Management and Humanities (AEMH 2019)	Li (2019)	Teaching of software architecture
S14	Paolo Ciancarini, Stefano Russo, Vincenzo Sabbatino	A Course on Software Architecture for Defense Applications	Proceedings of 4th International Conference in Software Engineering for Defence Applications	Ciancarini et al. (2016)	Teaching of software architecture
S15	Arvind W. Kiwelekar	A Software Architecture Teacher's Dilemmas	<a href="#">arXiv:2101.09434</a>	Kiwelekar (2021)	Teaching of software architecture
S16	Muhammad Aufeef Chauhan, Christian W Probst, Muhammad Ali Babar	Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods	Book: Agile and Lean Concepts for Teaching and Learning: Bringing Methodologies from Industry to the Classroom	Chauhan et al. (2019)	Teaching of software architecture
S17	Fáber D Giraldo, Sergio F Ochoa, Myriam Herrera et al	Applying a distributed CSCL activity for teaching software architecture	International Conference on Information Society (i-Society 2011)	Giraldo et al. (2011)	Teaching of software architecture
S18	Zhenyan Ji, Jing Song	Improved Teaching Model for Software Architecture Course	Proceedings of the 2015 International Conference on Education, Management, Information and Medicine	Ji and Song (2015)	Teaching of software architecture
S19	Bingyang Wei, Yihao Li, Lin Deng, Nicholas Visalli	Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application	Book: Studies in Computational Intelligence	Wei et al. (2020)	Teaching of software architecture



**Table 11** continued

No	Authors	Title	Name of the conference or journal	References	Category
S20	Rafael Capilla, Olaf Zimmermann, Carlos Carrillo, Hernán Astudillo	Teaching Students Software Architecture Decision Making	Software Architecture 2020	Capilla et al. (2020)	Teaching of software architecture
S21	Patrick de Beer, Samuel Angelov	Fontys ICT, Partners in Education Program: Intensifying Collaborations Between Higher Education and Software Industry	Proceedings of the 2015 European Conference on Software Architecture Workshops	de Beer and Angelov (2015)	Teaching of software engineering
S22	Gustavo Pinto, Clarice Ferreira, Cleice Souza, et al	Training software engineers using open-source software: The students' perspective	Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET 2019	Pinto et al. (2019)	Teaching of software engineering
S23	Kun May, Bo Yang, Jin Zhou, et al	Outcome-based school-enterprise cooperative software engineering training	ACM International Conference Proceeding Series	May et al. (2018)	Teaching of software engineering
S24	Robert Chatley, Tony Field	Lean learning - Applying lean techniques to improve software engineering education	Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET 2017	Chatley and Field (2017)	Teaching of software engineering

Table 11 continued

No	Authors	Title	Name of the conference or journal	References	Category
S25	Daniela Castelluccia, Bari Aldo, Giuseppe Visaggio, et al	Teaching evidence-based software engineering: learning by a collaborative mapping study of open source software	ACM SIGSOFT Software Engineering Notes	Castelluccia et al. (2013)	Teaching of software engineering
S26	Peter J Clarke, Jairo Pava, Yali Wu, Tariq M King	Collaborative Web-Based Learning of Testing Tools in SE Courses	Proceedings of the 42nd ACM Technical Symposium on Computer Science Educatio	Clarke et al. (2011)	Teaching of software engineering
S27	Fernanda Gomes Silva, Paolo Dos Santos, et al	FLOSS in Software Engineering Education: Supporting the Instructor in the Quest for Providing Real Experience for Students	Proceedings of the XXXIII Brazilian Symposium on Software Engineering	Silva et al. (2019)	Teaching of software engineering
S28	Charles Thevathayan, Margaret Hamilton	Imparting Software Engineering Design Skills	Proceedings - Frontiers in Education Proceedings of the Nineteenth Australasian Computing Education Conference, FIE	Thevathayan and Hamilton (2017)	Teaching of software engineering
S29	Charles Thevathayan, Margaret Hamilton	Imparting Software Engineering Design Skills	Proceedings of the Nineteenth Australasian Computing Education Conference	Meneely and Lucidi (2013)	Teaching of software engineering
S30	Matti Luukkainen, Arto Vihavainen, Thomas Vikberg	Three Years of Design-Based Research to Reform a Software Engineering Curriculum	Proceedings of the 13th Annual Conference on Information Technology Education	Luukkainen et al. (2012)	Teaching of software engineering

**Table 11** continued

No	Authors	Title	Name of the conference or journal	References	Category
S31	Sriram Mohan, Stephen Chenoweth, Shawn Bohner	Towards a Better Capstone Experience	Proceedings of the 43rd ACM Technical Symposium on Computer Science Education	Mohan et al. (2012)	Teaching of software engineering
S32	Rune Hjelmsvold, Deepti Mishra	Exploring and Expanding GSE Education with Open Source Software Development	ACM Transactions on Computing Education	Hjelmsvold and Mishra (2019)	Teaching of software engineering
S33	Lynette Johns-Boast, Shayne Flint	Simulating industry: An innovative software engineering capstone design course	2013 IEEE Frontiers in Education Conference (FIE)	Johns-Boast and Flint (2013)	Teaching of software engineering
S34	Maria Palacin-Silva, Jayden Khakurel, Ari Happonen, et al	Infusing Design Thinking into a Software Engineering Capstone Course	2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)	Palacin-Silva et al. (2017)	Teaching of software engineering
S35	Ravi Shankar Pillutla, Anuradha Alladi	Methodology to bridge the gaps between engineering education and the industry requirements	Eurocon 2013	Pillutla and Alladi (2013)	Teaching of software engineering
S36	Azim Abdool, Akash Pooransingh	An industry-mentored undergraduate software engineering project	2014 IEEE Frontiers in Education Conference (FIE) Proceedings	Abdool and Pooransingh (2014)	Teaching of software engineering
S37	Jayden Khakurel, Jari Porras	The Effect of Real-World Capstone Project in an Acquisition of Soft Skills among Software Engineering Students	2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE T)	Khakurel and Porras (2020)	Teaching of software engineering

Table 11 continued

No	Authors	Title	Name of the conference or journal	References	Category
S38	L Sun, Ting-qin Lv, Lichun Pan	Reinforcing practice teaching of Software Engineering for fostering the creative talent	2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)	Sun et al. (2011)	Teaching of software engineering
S39	Cécile Péraire	Dual-Track Agile in Software Engineering Education	2019 IEEE/ACM 41st International Conference on Software Engineering: Education and Training (ICSE-SEET)	Péraire (2019)	Teaching of software engineering
S40	Kevin D Wendt, Ken Reilly, Mats P E Heimdahl	First Steps towards Exporting Education: Software Engineering Education Delivered Online to Professionals	2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)	Wendt et al. (2016)	Teaching of software engineering
S41	Zahra Shakeri Hossein Abad, Muneera Bano, Didar Zowghi	How Much Authenticity Can Be Achieved in Software Engineering Project Based Courses?	Proceedings of the 41st International Conference on Software Engineering: Education and Training	Abad et al. (2019)	Teaching of software engineering
S42	Christoph Matthies, Ralf Teusner, Guenter Hesse	Beyond Surveys: Analyzing Software Development Artifacts to Assess Teaching Efforts	2018 IEEE Frontiers in Education Conference (FIE)	Matthies et al. (2018)	Teaching of software engineering
S43	N Pratheesh, T Devi	Assessment of student's learning style and engagement in traditional based software engineering education	2013 International Conference on Intelligent Interactive Systems and Assistive Technologies	Pratheesh and Devi (2013)	Teaching of software engineering

Table 11 continued

No	Authors	Title	Name of the conference or journal	References	Category
S44	Padmashree Desai, G H oshi, M ijyalaskhmi	A novel approach to carrying out mini project in Computer Science Engineering	2012 IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA)	Desai et al. (2012)	Teaching of software engineering
S45	Jeevamol Joy, V. G. Renumol	Activity oriented teaching strategy for software engineering course: An experience report	Journal of Information Technology Education: Innovations in Practice	Joy and Renumol (2018)	Teaching of software engineering
S46	Carlos Portela, Alexandre Vasconcelos, Sandro Oliveira, Maurício Souza	The Use of Industry Training Strategies in a Software Engineering Course: An Experience Report	2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)	Portela et al. (2017)	Teaching of software engineering
S47	Nicolás Martín Paez	A Flipped Classroom Experience Teaching Software Engineering	2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)	Paez (2017)	Teaching of software engineering
S48	Camelia Șerban, Virginia Niculescu, Andreea Vescan	Attaining competences in software quality oriented design based on cyclic learning	2020 IEEE Frontiers in Education Conference (FIE)	Șerban et al. (2020)	Teaching of software engineering
S49	G J Collins	Integrating Industry Seminars within a Software Engineering Module to Enhance Student Motivation	2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)	Collins (2020)	Teaching of software engineering

**Table 11** continued

No	Authors	Title	Name of the conference or journal	References	Category
S50	Jaejoon Lee, Gerald Kotonya, Jon Whittle, Christopher Bull	Software Design Studio: A Practical Example	2015 IEEE/ACM 37th IEEE International Conference on Software Engineering	Lee et al. (2015)	Teaching of software engineering
S51	Stan Jarzabek	Teaching advanced software design in team-based project course	2013 26th International Conference on Software Engineering Education and Training (CSEET)	Jarzabek (2013)	Teaching of software engineering
S52	Adriana Lopes, Igor Steinmacher, Tayana Conte	UML Acceptance: Analyzing the Students' Perception of UML Diagrams	Proceedings of the XXXIII Brazilian Symposium on Software Engineering	Lopes et al. (2019)	Complementary experiences
S53	Williamson Silva, Bruno Gadelha, Igor Steinmacher, Tayana Conte	What Are the Differences between Group and Individual Modeling When Learning UML?	Proceedings of the XXXII Brazilian Symposium on Software Engineering	Silva et al. (2018)	Complementary experiences
S54	Anthony Ward, Adeyosola Gbadebo, Bidyut Baruah	Using job advertisements to inform curricula design for the key global technical challenges	2015 International Conference on Information Technology Based Higher Education and Training (ITHET)	Ward et al. (2015)	Complementary experiences
S55	Matthias Galster, Samu Angelov	What makes teaching software architecture difficult?	Proceedings - International Conference on Software Engineering	Galster and Angelov (2016)	Complementary experiences
S56	Zhewei Hu, Yang Song, Edward F Gehringer	Open-Source Software in Class: Students' Common Mistakes	Proceedings of the 40th International Conference on Software Engineering Education and Training	Hu et al. (2018)	Complementary experiences

## **B Techniques in training software architects vs. primary studies**

See Table [12](#) **Techniques in training software architects vs. primary studies.**

**Table 12** Techniques in training software architects vs. primary studies

No	Primary studies	Strategies or techniques				Open-Source Projects training	Real clients or industry experts	Conferences featuring guests from the software industry sector	Field visits	Instructors and guest contributors in software engineering from the industry sector
		Small project-based training	Large project-based training							
1	Rupakheti and Chenoweth (2015)	X					X			
2	Li (2020)	X								
3	Li (2020)	X								
4	Chatley and Field (2017)	X								
5	Li (2019)	X						X		
6	Mohan et al. (2012)	X								
7	Palacin-Silva et al. (2017)	X								
8	Wang (2011)		X					X		
9	May et al. (2018)		X							
10	Luukkainen et al. (2012)		X							
11	Hjelsvold and Mishra (2019)		X		X					
12	Zhang et al. (2020)		X		X			X		
13	Abdool and Pooransingh (2014)		X		X		X			
14	Sun et al. (2011)		X		X					
15	Pénaire (2019)		X		X					
16	Matthies et al. (2018)		X		X					
17	Angelov and de Beer (2017)		X		X					



Table 12 continued

No	Primary studies	Strategies or techniques					Real clients or industry experts	Conferences featuring guests from the software industry sector	Field visits	Instructors and guest contributors in software engineering from the industry sector
		Small project-based training	Large project-based training	Open-Source Projects training						
18	Chauhan et al. (2019)		X							
19	Giraldo et al. (2011)		X							
20	Khakurel and Porras (2020)		X							
21	Pinto et al. (2019)			X						
22	Silva et al. (2019)			X						
23	Van Deursen et al. (2017)			X			X			
24	Hu et al. (2018)			X						
25	Wei et al. (2020)			X						
26	Weerawarana et al. (2012)			X		X				
27	Meneely and Lucidi (2013)							X		
28	de Beer and Angelov (2015)							X		
29	Ji and Song (2015)							X		
30	Collins (2020)							X		
31	Johns-Boast and Flint (2013)								X	
32	Ward et al. (2015)							X		
33	Joy and Renumol (2018)									
34	Paez (2017)									
7	Total		13	8			3	7	1	1

Table 12 continued

No	Primary studies	Strategies or techniques					Industry-applied tools	Current state-of-the-art software architecture	Design thinking
		developing software under conditions faced in the industry	Global software development	knowledge and skills demanded in the industry and develop the essential skills	Job advertisements to align curricula with industry needs				
1	Rupakheti and Chenoweth (2015)								
2	Li (2020)								
3	Li (2020)								
4	Chatley and Field (2017)								
5	Li (2019)								
6	Mohan et al. (2012)		X						
7	Palacin-Silva et al. (2017)								X
8	Wang (2011)								
9	May et al. (2018)								
10	Luukkainen et al. (2012)								
11	Hjelsvold and Mishra (2019)								
12	Zhang et al. (2020)		X					X	
13	Abdool and Pooransingh (2014)								
14	Sun et al. (2011)								
15	Péaire (2019)								
16	Mathies et al. (2018)								
17	Angelov and de Beer (2017)								

Table 12 continued

No	Primary studies	Strategies or techniques					
		developing software under conditions faced in the industry	Global software development	knowledge and skills demanded in the industry and develop the essential skills	Job advertisements to align curricula with industry needs	Industry-applied tools	Current state-of-the-art software architecture
18	Chauhan et al. (2019)						
19	Giraldo et al. (2011)						
20	Khakurel and Porras (2020)						
21	Pinto et al. (2019)						
22	Silva et al. (2019)						
23	Van Deursen et al. (2017)						
24	Hu et al. (2018)						
25	Wei et al. (2020)						
26	Weerawarana et al. (2012)						
27	Meneely and Lucidi (2013)						
28	de Beer and Angelov (2015)						
29	Ji and Song (2015)						X
30	Collins (2020)						
31	Johns-Boast and Flint (2013)						
32	Ward et al. (2015)		X		X		
33	Joy and Renumol (2018)		X				
34	Paez (2017)					X	
Total		1	1	2	1	1	2
							1

### **C Teaching methodologies in training software architects vs. primary studies**

See Table 13 Teaching methodologies in training software architects vs. primary studies.

**Table 13** Teaching methodologies in training software architects vs. primary studies

No	Primary studies	Strategies or techniques Professors teach traditional classes in software architecture education					Flipped classroom	Problem-based learning	Case-based learning	Online learning	Intelligent tutorial system
1	Wang (2011)	X									
2	Lieh Ouh et al. (2020)	X							X		
3	Péaire (2019)	X					X				
4	Palacin-Silva et al. (2017)						X				
5	Joy and Renumol (2018)						X				
6	Kiwelekar (2021)							X			
7	Paez (2017)						X				
8	Lieh and Irawan (2019)							X	X		
9	Khakurel and Porras (2020)							X			
10	Li (2019)								X		
11	Ji and Song (2015)								X		
12	Capilla et al. (2020)								X		
13	May et al. (2018)								X		
14	Wendt et al. (2016)									X	
15	Schmidt and McCormick (2013)									X	
16	Thevathayan and Hamilton (2017)										X
Total		3		4	3	6	2	1			

## **D Content of software architecture courses vs. primary studies**

See Table 14 **Content of software architecture courses vs. primary studies.**

**Table 14** Content of software architecture courses vs. primary studies

Primary studies	Content of the software architecture courses				
	Architecture patterns	Quality attributes	Case studies on the web, mobile, and cloud applications	Fundamentals of software archi-tectures: principles and concepts	Architectural views
Lieh Ouh et al. (2020)	X	X	X		
Van Deursen et al. (2017)	X				X
Ji and Song (2015)	X			X	X
Capilla et al. (2020)	X				
Li (2019)	X				
Li (2019)	X				
Lieh and Irawan (2018)		X		X	
Wendt et al. (2016)		X			
Lieh and Irawan (2019)					X
Chauhan et al. (2019)					
Capilla et al. (2020)					
Total	6	3	1	2	1
					3

Table 14 continued

Primary studies	Content of the software architecture courses				
	Product lines	Technical debt	Analysis, design and evaluation of software architecture	Design patterns	Software modeling
Lieh Ouh et al. (2020)					
Van Deursen et al. (2017)	X	X			
Ji and Song (2015)			X		
Capilla et al. (2020)		X	X		
Li (2019)					
Li (2019)					
Lieh and Irawan (2018)					
Wendt et al. (2016)				X	X
Lieh and Irawan (2019)					
Chauhan et al. (2019)		X			
Capilla et al. (2020)		X			
Total	1	1	3	1	1



E Soft skills vs. Primary studies

See Table 15 Soft skills vs. Primary studies.

Table 15 Soft skills vs. Primary studies

Primary studies	Soft skills			
	Effective communication	Teamwork	Leadership and negotiation	Artistic skills
Rupakheti and Chenoweth (2015)	X		X	X
Silva et al. (2019)	X			
Lieh and Irawan (2019)	X		X	
Mohan et al. (2012)	X	X		
Hjelsvold and Mishra (2019)	X	X		
Li (2020)		X		
Backert et al. (2020)	X			
Johns-Boast and Flint (2013)	X	X		
Khakurel and Porras (2020)	X	X		
Sun et al. (2011)	X	X		
Péaire (2019)	X	X		
Pratheesh and Devi (2013)	X			
Desai et al. (2012)	X	X		
Weerawarana et al. (2012)	X			
Van Deursen et al. (2017)	X			
Hu et al. (2018)	X			
Lee et al. (2015)	X	X		
Jarzabek (2013)	X	X		
Wei et al. (2020)	X			X
May et al. (2018)		X		
Zhang et al. (2020)		X		
Palacin-Silva et al. (2017)		X		
Zhang et al. (2020)		X		
Palacin-Silva et al. (2017)		X		
Abad et al. (2019)			X	
Total	18	15	3	2

## **F Technical skills vs. Primary studies**

See Table [16](#) **Technical skills vs. Primary studies.**

Table 16 Technical skills vs. Primary studies

Primary studies	Technical skills Being familiar with multiple technologies	Understand the domain in which a system will operate	Analytical skills	Design, model, analyze, and evaluate software architectures	Research skills	Minimum program- ming skills	Architectural decision- making	Systemic thinking
Rupakheti and Chenoweth (2015)	X	X	X					
Palacin-Silva et al. (2017)	X							
Wei et al. (2020)	X	X	X				X	
Lieh Ouh et al. (2020)								
Lieh and Irawan (2018)	X					X	X	
Joy and Renumol (2018)			X					
Angelov and de Beer (2017)			X				X	
Li (2020)			X		X	X		
Lopes et al. (2019)				X				
Silva et al. (2018)				X				
Mohan et al. (2012)				X				
Zhang et al. (2020)				X	X			
Desai et al. (2012)				X				
Van Deursen et al. (2017)				X				
Schmidt and McCormick (2013)				X				
Ciancarini et al. (2016)				X				
Jarzabek (2013)				X				
Total	4	2	5	9	2	2	2	1

## **G Challenges to training vs primary studies**

See Table [17](#) **Challenges to training vs primary studies.**

Table 17 Challenges to training vs primary studies

Primary	Challenges to training about software architecture to student					
	Lack of real-world experience among instructors	The abstract and ambiguous nature of software architecture	Difficulty in involving clients and industry experts	Teaching software architecture requires going beyond traditional teaching methods	Small and simple projects are insufficient for providing students with practical experience	Lack of involvement in open-source projects
Rupakheti and Chenoweth (2015)						
Li (2020)		X				Personalized monitoring
Zhang et al. (2020)		X				
Lieh and Irawan (2018)		X				
Sun et al. (2011)		X				
Van Deursen et al. (2017)		X		X		
Lieh and Irawan (2019)		X				
Li (2019)		X				
Kiwelekar (2021)		X				
de Beer and Angelov (2015)			X			
Pinto et al. (2019)				X	X	
Pillutla and Alladi (2013)				X		
Pratheesh and Devi (2013)				X		
Paez (2017)				X		

Table 17 continued

Primary	Challenges to training about software architecture to student					
	Lack of real-world experience among instructors	The abstract and ambiguous nature of software architecture	Difficulty in involving clients and industry experts	Teaching software architecture requires going beyond traditional teaching methods	Small and simple projects are insufficient for providing students with practical experience	Lack of involvement in open-source projects
Chatley and Field (2017)					X	
Silva et al. (2019)					X	X
Galster and Angelov (2016)					X	
Hjelsvold and Mishra (2019)						X
Thevathayan and Hamilton (2017)						X
Palacin-Silva et al. (2017)						
Abad et al. (2019)						
Angelov and de Beer (2017)						
Wei et al. (2020)						
Matthies et al. (2018)						
Weerawarana et al. (2012)						
Joy and Renumol (2018)						
Total	1	8	1	5	4	2
						1



**Table 17** continued

Primary	Challenges to training about software architecture to student					
	Modern methods developed in the industry may take time to reach academia	Teaching software architectures is a challenging task	The need for courses to adapt to changes	Completed project scenarios intimidate students	Gap between students' skills and the expectations of industry	Clarify how to work with software architecture in agile methodologies
Chatley and Field (2017)						
Silva et al. (2019)						
Galster and Angelov (2016)						
Hjelsvold and Mishra (2019)						
Thevathayan and Hamilton (2017)	X					
Palacin-Silva et al. (2017)	X					
Abad et al. (2019)	X					
Angelov and de Beer (2017)	X				X	
Wei et al. (2020)	X					
Matthies et al. (2018)		X				
Weerawarana et al. (2012)			X			
Joy and Renumol (2018)				X		
Total	1	4	1	1	1	1



## **H Validation-techniques of software architecture courses vs. primary studies**

See Table 18 **Validation-techniques of software architecture courses vs. primary studies.**

**Table 18** Validation-techniques of software architecture courses vs. primary studies

Primary studies	Validation techniques		Metrics to measure the quality of the artifacts produced	Metrics to analyze the students' evaluations	The skills acquired in the course are crossed vs. the soft skills reported in the literature
	Student surveys	Metrics to measure before and after the course			
Rupakheti and Chenoweth (2015)	X				
Li (2020)	X				
Pinto et al. (2019)	X				
Chatley and Field (2017)	X				
Lopes et al. (2019)	X				
Lieh and Irawan (2019)	X				
Meneely and Lucidi (2013)	X				
Lieh Ouh et al. (2020)	X				
Lieh and Irawan (2018)	X				
Palacin-Silva et al. (2017)	X				
Pénaire (2019)	X				
Wendt et al. (2016)	X				
Matthies et al. (2018)	X				
Joy and Renumol (2018)	X		X		
Angelov and de Beer (2017)	X				
Van Deursen et al. (2017)	X				
Li (2019)	X			X	
Ciancarini et al. (2016)	X			X	
Paez (2017)	X				
Giraldo et al. (2011)	X				
May et al. (2018)		X			
Silva et al. (2018)			X		
Portela et al. (2017)			X		
Johns-Boast and Flint (2013)				X	
Ji and Song (2015)				X	
Khakurel and Porras (2020)					X
Total	20	1	3	4	1

**Author Contributions** WLPY co-designed the study, collected and analyzed the data, and wrote the paper. JAHA designed the study, edited the paper, and provided guidelines through the process. AWK reviewed and edited the paper. All authors read and approved the final manuscript. AB reviewed and edited the paper. All authors read and approved the final manuscript.

**Funding** Open Access funding provided by Colombia Consortium

**Data Availability** The datasets generated during and/or analysed during the current study are available in the repository: <https://github.com/wlibar/TrainingSoftwareArchitecture>.

## Declarations

**Conflict of interest** The authors have no competing interests to declare relevant to this article's content: "Training Software Architects Suiting Software Industry Needs: A Literature Review."

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

- Abad, Z.S.H., Bano, M., & Zowghi, D. (2019). How Much Authenticity Can Be Achieved in Software Engineering Project Based Courses?. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '19)*, 208–219. IEEE Press, Montreal Quebec Canada. <https://doi.org/10.1109/ICSE-SEET.2019.00030>
- Abdool, A., & Pooransingh, A. (2014). An industry-mentored undergraduate software engineering project. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, 1–4. IEEE, Madrid, Spain. <https://doi.org/10.1109/FIE.2014.7044180>
- Angelov, S., & de Beer, P. (2017). Designing and Applying an Approach to Software Architecting in Agile Projects in Education. *Journal of Systems and Software*, 127(C), 78–90. 0164-1212, <https://doi.org/10.1016/j.jss.2017.01.029>
- Backert, M., Blum, T., Kreuter, R., Paulisch, F. & Zimmerer, P. (2020). Software Curriculum @ Siemens - The Architecture of a Training Program for Architects. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEET)*, 1–6. IEEE, Munich, Germany. <https://doi.org/10.1109/CSEET49119.2020.9206182>
- Bass, L., Clements, P., & Kazman, R.(2012). *Software architecture in practice, third Edition*. Pearson Education, Massachusetts, USA. 978-0321815736, <https://www.amazon.com/Software-Architecture-Practice-3rd-Engineering/dp/0321815734>
- Brito, M.S., Silva, F.G., von Flach, C., Chavez, G., Nascimento, D.C., & Bittencourt, R.A. (2018). FLOSS in Software Engineering Education: An Update of a Systematic Mapping Study. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering (SBES '18)*, 250–259. Association for Computing Machinery, New York, NY, USA. 9781450365031. <https://doi.org/10.1145/3266237.3266249>
- Capilla, R., Zimmermann, O., Carrillo, C. & Astudillo, H. (2020). Teaching Students Software Architecture Decision Making. In A. Jansen, I. Malavolta H. Muccini, I. Ozkaya, & O. Zimmermann (Eds.), *Software Architecture*, 231–246. Springer International Publishing, Cham. 978-3-030-58923-3
- Castelluccia, D., Aldo, B., Visaggio, G., Aldo, B., Castelluccia, D., & Visaggio, G. (2013). Teaching evidence-based software engineering: learning by a collaborative mapping study of open source software. In *ACM SIGSOFT Software Engineering Notes*, vol. 38, 1–4. ACM, Madrid, Spain. 0163-5948, <https://doi.org/10.1145/2532780.2532803>

- Chatley, R., & Field, T. (2017). Lean learning - Applying lean techniques to improve software engineering education. In *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET 2017*, 117–126. IEEE Press, Buenos Aires. 9781538626719 <https://doi.org/10.1109/ICSE-SEET.2017.5>
- Chauhan, M.A., Probst, C.W., & Babar, M.A. (2019). *Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods*. Springer Singapore, Singapore, 325–351. 978-981-13-2751-3. [https://doi.org/10.1007/978-981-13-2751-3\\_16](https://doi.org/10.1007/978-981-13-2751-3_16)
- Ciancarini, P., Russo, S., & Sabbatino, V. (2016). A Course on Software Architecture for Defense Applications. In P. Ciancarini, A. Sillitti, G. Succi, & A. Messina (Eds.), *Proceedings of 4th International Conference in Software Engineering for Defence Applications*, 321–330. Springer International Publishing, Cham. 978-3-319-27896-4
- Clarke, P.J., Pava, J., Wu, Y., & King, T.M. (2011). Collaborative Web-Based Learning of Testing Tools in SE Courses. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education. (SIGCSE '11)*, 147–152. Association for Computing Machinery, New York, NY, USA. 9781450305006. <https://doi.org/10.1145/1953163.1953208>
- Collins, G.J. (2020). Integrating Industry Seminars within a Software Engineering Module to Enhance Student Motivation. In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, 1497–1502. IEEE, Opatija, Croatia. <https://doi.org/10.23919/MIPRO48935.2020.9245232>
- da Cunha, J.A.O.G., Marques, G.A., Lemos, W.L., Câmara, U.D., & Vasconcellos, F.J.S. (2018). Software Engineering Education in Brazil: A Mapping Study. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering (SBES '18)*, 348–356. Association for Computing Machinery, New York, NY, USA. 9781450365031 <https://doi.org/10.1145/3266237.3266259>
- de Beer, P., & Angelov, S. (2015). Fontys ICT, Partners in Education Program: Intensifying Collaborations Between Higher Education and Software Industry. In *Proceedings of the 2015 European Conference on Software Architecture Workshops (ECSAW '15)*, 4. Association for Computing Machinery, New York, NY, USA. 9781450333931. <https://doi.org/10.1145/2797433.2797468>
- De Boer, R.C., & Van Vliet, H. (2009). On the similarity between requirements and architecture. *Journal of Systems and Software*, 82(3), 544–550. 0164-1212. <https://doi.org/10.1016/j.jss.2008.11.185>
- Desai, P., Joshi, G. H., & Vijayalaskhmi, M. (2012). A novel approach to carrying out mini project in Computer Science Engineering. In *2012 IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA)*, 1–4. IEEE, Kottayam, Kerala, India. <https://doi.org/10.1109/AICERA.2012.6306699>
- Dobrica, L., & Niemela, E. (2002). A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7), 638–653. 2326-3881. <https://doi.org/10.1109/TSE.2002.1019479>
- Fowler, M. (2003). Who Needs an Architect? *IEEE Software*, 20(5), 11–13. 0740-7459. <https://doi.org/10.1109/MS.2003.1231144>
- Galster, M., & Angelov, S. (2016). What makes teaching software architecture difficult?. In *Proceedings - International Conference on Software Engineering*, 356–359. Association for Computing Machinery New YorkNYUnited States, Austin Texas. 9781450341615. 02705257. <https://doi.org/10.1145/2889160.2889187>
- Garousi, V., Giray, G., Tüzün, E., Catal, C., & Felderer, M. (2020). Closing the gap between software engineering education and industrial needs. *IEEE Software*, 37, 68–77. <https://doi.org/10.1109/MS.2018.2880823>
- Giraldo, F.D., Ochoa, S.F., Herrera, M., Neyem, A., Arciniegas, J.L., Clunie, C., Zapata, S. & Lizano, F. (2011). Applying a distributed CSCL activity for teaching software architecture. In *International Conference on Information Society (i-Society 2011)*, 208–214. IEEE, London, United Kingdom. <https://doi.org/10.1109/i-Society18435.2011.5978540>
- Groeneveld, W., Vennekens, J., & Aerts, K. (2019). Software engineering education beyond the technical: A systematic literature review. [arXiv:1910.09865](https://arxiv.org/abs/1910.09865)
- Harrison, N. B., & Avgeriou, P. (2010). How do architecture patterns and tactics interact? A model and annotation. *Journal of Systems and Software*, 83(10), 1735–1758.
- Hjelsvold, R., & Mishra, D. (2019). Exploring and Expanding GSE Education with Open Source Software Development. *ACM Transactions on Computing Education*, 19(2), 23. <https://doi.org/10.1145/3230012>
- Hu, Z., Song, Y., & Gehringer, E.F. (2018). Open-Source Software in Class: Students' Common Mistakes. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering*

- Education and Training (ICSE-SEET '18)*, 40–48. Association for Computing Machinery, New York, NY, USA. 9781450356602. <https://doi.org/10.1145/3183377.3183394>
- Jansen, A., & Bosch, J. (2005). Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, 109–120. IEEE, Pittsburgh, PA, USA. <https://doi.org/10.1109/WICSA.2005.61>
- Jarzabek, S. (2013). Teaching advanced software design in team-based project course. In *2013 26th International Conference on Software Engineering Education and Training (CSEET)*, 31–40. IEEE, San Francisco, CA, USA. <https://doi.org/10.1109/CSEET.2013.6595234>
- Ji, Z., & Song, J. (2015). Improved Teaching Model for Software Architecture Course. In *Proceedings of the 2015 International Conference on Education, Management, Information and Medicine*, 333–338. Atlantis Press, No City. 978-94-62520-68-4. 2352-5428. <https://doi.org/10.2991/emim-15.2015.65>
- Johns-Boast, L., & Flint, S. (2013). Simulating industry: An innovative software engineering capstone design course. In *2013 IEEE Frontiers in Education Conference (FIE)*, 1782–1788. IEEE, Oklahoma City, OK, USA. <https://doi.org/10.1109/FIE.2013.6685145>
- Joy, J., & Renumol, V. G. (2018). Activity oriented teaching strategy for software engineering course: An experience report. *Journal of Information Technology Education: Innovations in Practice*, 17 181–200. 2165316X. <https://doi.org/10.28945/4116>
- Khakurel, J., & Porras, J. (2020). The Effect of Real-World Capstone Project in an Acquisition of Soft Skills among Software Engineering Students. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEET)*, 1–9. IEEE, Munich, Germany. <https://doi.org/10.1109/CSEET49119.2020.9206201>
- Kiwelkar, A. W. (2021). A Software Architecture Teacher's Dilemmas. 10 pages. [arXiv:2101.09434](https://arxiv.org/abs/2101.09434)
- Lee, J., Kotonya, G., Whittle, J., & Bull, C. (2015). Software Design Studio: A Practical Example. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, 389–397. IEEE, Florence, Italy. <https://doi.org/10.1109/ICSE.2015.171>
- Li, W. (2019). Teaching Reform and Practice of Software Architecture Design Course under the Background of Engineering Education. In *Proceedings of the 2019 International Conference on Advanced Education, Management and Humanities (AEMH 2019)*, vol. 352, 17–21. Atlantis Press, Wuhan, China. <https://doi.org/10.2991/aemh-19.2019.4>
- Li, Z. (2020). Using Public and Free Platform-as-a-Service (PaaS) based Lightweight Projects for Software Architecture Education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*, 1–11. Association for Computing Machinery, Seoul South Korea. <https://doi.org/10.1145/3377814.3381704>
- Lieh, O. E., & Irawan, Y. (2018). Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course. In *2018 IEEE Frontiers in Education Conference (FIE)*, 1–9. IEEE, San Jose, CA, USA. <https://doi.org/10.1109/FIE.2018.8659200>
- Lieh, O. E., & Irawan, Y. 2019. Teaching adult learners on software architecture design skills. In *Proceedings - Frontiers in Education Conference, FIE*, Vol. 2018-October, 1–9. IEEE, Uppsala, Sweden. 9781538611739. 15394565. <https://doi.org/10.1109/FIE.2018.8658714>
- Lieh Ouh, E., Gan, Kok Siew, B., & Irawan, Y. (2020). Did our Course Design on Software Architecture meet our Student's Learning Expectations?. In. (2020). IEEE Frontiers in Education Conference (FIE), 1–9. IEEE, Uppsala, Sweden. <https://doi.org/10.1109/FIE44824.2020.9274014>
- Lopes, A., Steinmacher, I., & Conte, T. (2019). UML Acceptance: Analyzing the Students' Perception of UML Diagrams. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES 2019)*, 264–272. Association for Computing Machinery, New York, NY, USA. 9781450376518. <https://doi.org/10.1145/3350768.3352575>
- Luukkainen, M., Vihavainen, A., & Vikberg, T. (2012). Three Years of Design-Based Research to Reform a Software Engineering Curriculum. In *Proceedings of the 13th Annual Conference on Information Technology Education (SIGITE '12)*, 209–214. Association for Computing Machinery, New York, NY, USA. 9781450314640. <https://doi.org/10.1145/2380552.2380613>
- Matthies, C., Teusner, R., & Hesse, G. (2018). Beyond Surveys: Analyzing Software Development Artifacts to Assess Teaching Efforts. In *2018 IEEE Frontiers in Education Conference (FIE)*, 1–9. IEEE, San Jose, CA, USA. <https://doi.org/10.1109/FIE.2018.8659205>
- May, K., Yang, B., Zhou, J., Lin, Y Zhang, K., & Yu, Z. (2018). Outcome-based school-enterprise cooperative software engineering training. In *ACM International Conference Proceeding Series*, 15–20. Association for Computing Machinery New YorkNYUnited States, Shanghai China. 9781450364157. <https://doi.org/10.1145/3210713.3210722>

- Meneely, A., & Lucidi, S. (2013). Vulnerability of the Day: Concrete Demonstrations for Software Engineering Undergraduates. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*, 1154–1157. IEEE Press, San Francisco CA USA. 9781467330763
- Mohan, S., Chenoweth, S., & Bohner, S. (2012). Towards a Better Capstone Experience. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE '12)*, 111–116. Association for Computing Machinery, New York, NY, USA. 9781450310987. <https://doi.org/10.1145/2157136.2157173>
- Oliveira, B.R.N., Garcés, L., Lyra, K.T., Santos, D.S., Isotani, S. & Nakagawa, E.Y. (2022). An Overview of Software Architecture Education. In *Anais do XXV Congresso Ibero-Americano em Engenharia de Software*, 76–90. SBC
- Paez, N.M. (2017). A Flipped Classroom Experience Teaching Software Engineering. In *2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)*, 16–20. IEEE, Buenos Aires, Argentina. <https://doi.org/10.1109/SECM.2017.6>
- Palacin-Silva, M., Khakurel, J., Happonen, A., Hynninen, T., & Porras, J. (2017). Infusing Design Thinking into a Software Engineering Capstone Course. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEET)*, 212–221. IEEE, Savannah, Georgia, USA. <https://doi.org/10.1109/CSEET.2017.41>
- Péraire, C. (2019). Dual-Track Agile in Software Engineering Education. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 38–49. IEEE, Montreal, QC, Canada. <https://doi.org/10.1109/ICSE-SEET.2019.00013>
- Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08)*, 68–77. BCS Learning & Development Ltd., Swindon, GBR. <https://doi.org/10.14236/ewic/EASE2008.8>
- Pillutla, R.S., & Alladi, A. (2013). Methodology to bridge the gaps between engineering education and the industry requirements. In *Eurocon 2013*, 926–932. IEEE, Zagreb, Croatia. <https://doi.org/10.1109/EUROCON.2013.6625093>
- Pinto, G., Ferreira, C., Souza, C., Steinmacher, I., & Meirelles, P. (2019). Training software engineers using open-source software: The students' perspective. In *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET 2019*, 147–157. IEEE, Montreal Quebec Canada. 9781728110004. <https://doi.org/10.1109/ICSE-SEET.2019.00024>
- Portela, C., Vasconcelos, A., Oliveira, S., & Souza, M. (2017). The Use of Industry Training Strategies in a Software Engineering Course: An Experience Report. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEET)*, 29–36. IEEE, Savannah, Georgia. <https://doi.org/10.1109/CSEET.2017.16>
- Pratheesh, N., & Devi, T. (2013). Assessment of student's learning style and engagement in traditional based software engineering education. *2013 International Conference on Intelligent Interactive Systems and Assistive Technologies* (pp. 25–31). Coimbatore, India: IEEE.
- Qadir, M.M., & Usman, M. (2011). Software Engineering Curriculum: A systematic mapping study. In *2011 Malaysian Conference in Software Engineering*, 269–274. <https://doi.org/10.1109/MySEC.2011.6140682>
- Richards, M., & Ford, N. (2020). *Fundamentals of Software Architecture: An Engineering Approach 1st Edition*. O'Reilly Media, Inc., Canada. 383 pages. 978-1492043454 <https://www.amazon.com/Fundamentals-Software-Architecture-Comprehensive-Characteristics/dp/1492043451>
- Rodrigues, C., & Werner, C. (2009). Software architecture teaching: A systematic review. In *9th IFIP World Conference on Computers in Education (WCCE)*, 1–10. Bento Gonçalves.
- Ronchieri, E., & Canaparo, M. (2019). Metrics for software reliability: A systematic mapping study. *Journal of Integrated Design and Process Science*, 22(2), 5–25. 10920617. <https://doi.org/10.3233/JID180008>
- Rupakheti, C.R., & Chenoweth, S.V. (2015). Teaching Software Architecture to Undergraduate Students: An Experience Report. In *Proceedings - International Conference on Software Engineering*, vol. 2, 445–454. IEEE Press, Florence Italy. 9781479919345. 02705257. <https://doi.org/10.1109/ICSE.2015.177>
- Sabry, A. E. (2015). Decision model for software architectural tactics selection based on quality attributes requirements. *Procedia Computer Science*, 65, 422–431.
- Schmidt, D.C., & McCormick, Z. (2013). Producing and delivering a MOOC on pattern-oriented software architecture for concurrent and networked software. In *SPLASH 2013 - Proceedings of the*

- 2013 Companion Publication for Conference on Systems, Programming, and Applications: Software for Humanity, 167–176. ACM, Indianapolis Indiana USA. 9781450319959. <https://doi.org/10.1145/2508075.2508465>
- Silva, F.G., dos Santos, P.E.D., & Chavez, C.v.F.G. (2019). FLOSS in Software Engineering Education: Supporting the Instructor in the Quest for Providing Real Experience for Students. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, 234–243. ACM, Salvador Brazil. 9781450376518. <https://doi.org/10.1145/3422392.3422493>
- Silva, W., Gadelha, B., Steinmacher, I., & Conte, T. (2018). What Are the Differences between Group and Individual Modeling When Learning UML?. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering (SBES '18)*, 308–317. Association for Computing Machinery, New York, NY, USA. 9781450365031 <https://doi.org/10.1145/3266237.3266255>
- Souza, F.C., Santos, A., Andrade, S., Durelli, R., Durelli, V., & Oliveira, R. (2018). Automating Search Strings for Secondary Studies. In S. Latifi (Ed.), *Information Technology - New Generations*, 839–848. Springer International Publishing, Cham. 978-3-319-54978-1
- Sun, L., Lv, T.-Q., & Pan, L. (2011). Reinforcing practice teaching of Software Engineering for fostering the creative talent. *2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)* (pp. 1548–1551). Xianning, China: IEEE.
- Thevathayan, C., & Hamilton, M. (2017). Imparting Software Engineering Design Skills. In *Proceedings of the Nineteenth Australasian Computing Education Conference (ACE '17)*, 95–102. Association for Computing Machinery, New York, NY, USA. 9781450348232 <https://doi.org/10.1145/3013499.3013511>
- Van Deursen, A., Aniche, M., Aué, J., Slag, R., De Jong, M., Nederlof, A., & Bouwers, E. (2017). A Collaborative approach to teaching software architecture. In *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, 591–596. ACM, Seattle Washington USA. 9781450346986 <https://doi.org/10.1145/3017680.3017737>
- Wang, A. I. (2011). Extensive Evaluation of Using a Game Project in a Software Architecture Course. *ACM Transactions on Computing Education (TOCE)*, 11(1), 28. <https://doi.org/10.1145/1921607.1921612>
- Ward, A., Gbadebo, A., & Baruah, B. (2015). Using job advertisements to inform curricula design for the key global technical challenges. In *2015 International Conference on Information Technology Based Higher Education and Training (ITHET)*, 1–6. IEEE, Antalya, Turkey. <https://doi.org/10.1109/ITHET.2015.7218042>
- Weerawarana, S.M., Perera, A.S., & Nanayakkara, V. 2012. Promoting creativity, innovation and engineering excellence. In *Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE) 2012*, T1C–12–T1C–17. IEEE, Wuhan, China. <https://doi.org/10.1109/TALE.2012.6360374>
- Wei, B., Li, Y., Deng, L., & Visalli, N. (2020). *Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application*, vol. 845. Springer International Publishing, Cham, 43–54. 978-3-030-24344-9 18609503 [https://doi.org/10.1007/978-3-030-24344-9\\_3](https://doi.org/10.1007/978-3-030-24344-9_3)
- Wendt, K.D., Reily, K., & Heimdahl, M.P.E. (2016). First Steps towards Exporting Education: Software Engineering Education Delivered Online to Professionals. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, 241–245. IEEE, Dallas, TX, USA. <https://doi.org/10.1109/CSEET.2016.32>
- Zhang, L., Li, Y., & Ge, N. (2020). Exploration on theoretical and practical projects of software architecture course. In *15th International Conference on Computer Science and Education, ICCSE 2020*, 391–395. IEEE, Delft, Netherlands. 9781728172675 <https://doi.org/10.1109/ICCSE49874.2020.9201748>
- Şerban, C., Niculescu, V., & Vescan, A. (2020). Attaining competences in software quality oriented design based on cyclic learning. In *2020 IEEE Frontiers in Education Conference (FIE)*, 1–9. IEEE, Uppsala, Sweden. <https://doi.org/10.1109/FIE44824.2020.9274227>

## Authors and Affiliations

**Wilson Libardo Pantoja Yépez<sup>1</sup>**  · **Julio Ariel Hurtado Alegría<sup>1</sup>** · **Ajay Bandi<sup>2</sup>** · **Arvind W. Kiwelekar<sup>3</sup>**

Julio Ariel Hurtado Alegría  
ahurtado@unicauca.edu.co

Ajay Bandi  
ajay@nwmissouri.edu

Arvind W. Kiwelekar  
awk@dbatu.ac.in

- <sup>1</sup> Departamento de Sistemas, Universidad del Cauca, Popayán, Colombia, Edificio de Ingenierías, Sector Tulcán, Popayán 190002, Colombia
- <sup>2</sup> Northwest Missouri State University, Maryville, Missouri, USA
- <sup>3</sup> Department of Computer Engineering, Dr. Babasaheb Ambedkar Technological University, Raigad, Maharashtra 402 103, India