An Overview of Software Architecture Education

Brauner R. N. Oliveira[1], **Lina Garcés**[2], **Kamila T. Lyra**[1], **Daniel S. Santos**[1], **Seiji Isotani**[1], **Elisa Y. Nakagawa**[1]

[1] University of São Paulo (USP)
São Carlos – SP – Brazil

[2]Federal University of Itajubá (UNIFEI)
Itajubá – MG – Brazil

{brauner,kalyra_03,danielss}@usp.br, lina@unifei.edu.br

{sisotani,elisa}@icmc.usp.br

Abstract. *Software architecture comprises the most relevant structure of a software system and is a factor to enable systems to achieve qualities that are crucial to the system's goals. In this sense, teaching software architecture to students and professionals remains a very important task. However, it is still a challenging matter for teachers and learners on different levels and contexts. We also lack an updated overview on how such a challenge has been addressed. This paper presents an overview of software architecture education experiences and initiatives. For this, we systematically examined the most relevant studies addressing software architecture education. After analyzing a total of 50 studies, we discuss how software architecture has been taught, the topic mostly taught, the learning objectives, and the learning methods. Based on our findings, we also present open issues that still remain to be further investigated, aiming to mature software architecture education.*

1. Introduction

The software architecture of a system comprises its main software elements, the relation among such elements, and their properties [Bass et al. 2021]. Different responsibilities within the system's context are assigned to such elements and can drastically affect qualities, such as performance, maintainability, and security, which the system must exhibit to achieve its goals. In this sense, designing a software architecture capable of meeting system requirements is a crucial activity during the early stages of the software development process. Just as software architecture is relevant to software development, so is teaching it to students of computer science-related courses and professionals working on software development. The topic of software architecture has been taught since its emergence in 1992.

Several initiatives and experiences of teaching software architecture have been published, pointing to what can be considered a consensus: Teaching software architecture is a challenging task [Mannisto et al. 2008, Lago and Vliet 2005, Rupakheti and Chenoweth 2015, Galster and Angelov 2016, Cervantes et al. 2016, Deursen et al. 2017, Angelov and Beer 2017, Lago et al. 2019]. Several are the reasons for that, such as students of computing-related degree programs often face a large gap between low-level courses (e.g., programming and data structures) and high-level courses, in which they have to design effective systems [Garlan et al. 1992]. Without appropriate directions for teaching software architecture, the already challenging task to define a suitable architecture becomes a barrier to the proper education of future software engineers [Garlan et al. 1992]. Moreover, students have difficulties with problem-solving when the problem can be solved in many ways (i.e., with no single best solution) [Lago and Vliet 2005, Galster and Angelov 2016], which is the case for software architecture design. In turn, the architectural design process is difficult to be learned and performed with confidence since many aspects have to be considered when making architectural decisions, and there are many of them to be made during the software development [Cervantes et al. 2016]. In general, the desired skills of architects are mostly developed or gained through experience over many years of developing and designing software architectures [Rupakheti and Chenoweth 2015, Lago et al. 2019, Cervantes et al. 2016]. It means that instructors have to choose those topics that students will face when they go into the industry and teach them in a way that will ease the path throughout their career. More reasons make teaching software architecture too difficult, as discussed in depth in [Galster and Angelov 2016].

Despite several experiences and initiatives already published, to the best of our knowledge, there is only one study that analyzed how software architecture education has been addressed [Rodrigues and Werner 2009]. This study examined 37 studies that reported on software architecture teaching initiatives and, more specifically, it discussed the maturity of such initiatives, the employment of large and complex systems as part of the content addressed, and the educational approaches and resources used. However, this study was published more than ten years. Considering the relevance of investigating software architecture education and the continuing evolution of both the state of practice and state of the art, an updated review becomes necessary to address other research questions and objectives.

The main goal of this work is to present an overview of software architecture education based on the literature. For this, we examined 50 main studies published on the matter since 1992 and analyzed different aspects, including the types of experience within software architecture education, how the subject has been addressed, which topics have been considered relevant for the learning process, which categories of learning objectives have received more attention, and the educational approaches that have been employed. As a main result, we found many different teaching experiences, which have addressed several perspectives of the learning and teaching process, and various subjects at different cognitive levels; however, little attention has been given to teaching recent or trending topics of software architectures, like microservices and cloud-based architectures. Another important finding is that software architecture is still challenging and difficult to teach and learn; therefore, more attention must be paid to maturing and consolidating the field of software architecture education.

The remainder of this paper is organized as follows. Section 2 presents the method we followed to search, select, and analyze the studies. Section 3 presents the result and findings, whereas Section 4 discusses the results and presents the threats to the validity. Finally, we outline our conclusion in Section 5.

2. Research Method

To provide an overview of software architecture education, we followed the guidelines proposed in [Petersen et al. 2015] for planning, conducting, and reporting systematic mapping studies (SMS). In this section, we outline the most important sections of the SMS protocol, which is fully available online[1].

During planning phase, we established the following research questions:

- **RQ1.** How has software architecture been taught?
- **RQ2.** What topics have been taught in the software architecture area?
- **RQ3.** Which learning objectives have been considered in software architecture teaching?
- **RQ4.** Which learning methods have been adopted to teach software architecture?

Based on such questions, we developed the following search string, after rounds of calibration: *((“software architecture”) AND (“education” OR “educational” OR “course” OR “training” OR “teaching”)).* This string was adapted to each database engine and applied in 2021 to the following databases: ACM Digital Library[2], Engineering Village[3], IEEE Xplore Digital Library[4], and Scopus[5]. Our search was limited to studies published since 1992 and, when possible, we restricted the search to the title, abstract, and keywords of the studies. The number of studies returned by each search engine is detailed in the SMS protocol provided in the external material. Moreover, we considered one inclusion criterion (IC) and five exclusion criteria (EC) to support the selection process:

- IC1: The study reports an experience with software architecture education.

- EC1: The study was published before 1992.
- EC2: The study is not presented in English.
- EC3: The full text of the study is not accessible or available.
- EC4: The study is reported in a book or gray literature.
- EC5: The study is a duplicate of another study included.

The execution phase process is shown in Figure 1. We obtained 4,741 studies from the database search and, after removing the duplicated studies, we started the first selection phase with 2,871 primary studies to be analyzed. In that step, all 2,871 studies had their titles and abstracts read by the reviewers. Sometimes, the introduction section was also analyzed, enabling a better application of selection criteria. After finishing the first selection, 80 studies were thoroughly analyzed during the second selection. During that step, the 80 studies were fully read to confirm whether the selection criteria were correctly applied or not. As a result, we excluded 32 more studies that were not filtered through exclusion criteria during the first selection. For the remaining 48 studies, we extracted all the pieces of information that were registered in our extraction form, including

[1]github.com/brauneroliveira/SAE-SMS/blob/master/Protocol.pdf
[2]dl.acm.org
[3]engineeringvillage.com
[4]ieeexplore.ieee.org
[5]scopus.com

the references flagged as potential candidates for snowballing [Wohlin 2014]. After finishing this step, we performed the backward snowballing procedure, which allowed us to identify other two primary studies relevant to our research. In total, we included 50 primary studies as relevant for our SMS, as listed in Table 1.

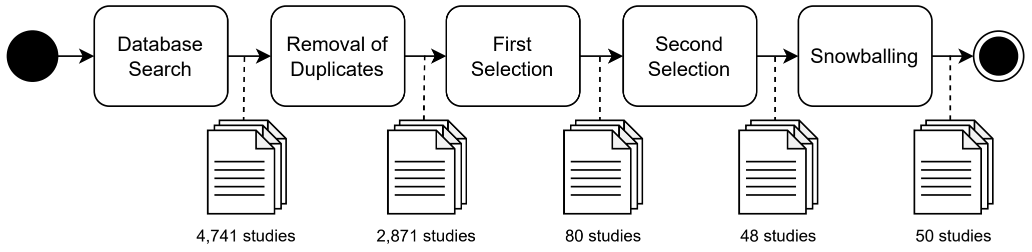

Figure 1. Review process overview

Table 1. Studies included for analysis phase.

ID	Reference
S1	Garlan, D., Shaw, M., Okasaki, C., Scott, C., and Swonger, R. (1992). **Experience with a course on architectures for software systems**. In 6th SEI Conference on Software Engineering Education, pages 23–43.
S2	Royce, W., Boehm, B. , and Druffel, C. (1994). **Employing UNAS technology for software architecture at the University of Southern California**. In 11th Annual Washington Ada Symposium and SIGAda Summer Meeting (WAdaS/SIGAda), page 113
S3	Butler, S. (1999). **A client/server case study for software engineering students**. In 12th Conference on Software Engineering Education and Training (CSEE&T), pages 156–165.
S4	Fairbanks, G. (2003). **Why can't they create architecture models like "developer x"? an experience report**. In 25th International Conference on Software Engineering (ICSE), pages 548–552.
S5	Karam, O., Qian, K., and Diaz-Herrera, J. (2004). **A model for SWE course "software architecture and design"**. In 34th Frontiers in Education (FIE), pages 4–8.
S6	Vickers, B. (2004). **Architecting a software architect**. IEEE Aerospace Conference, pages 4155–4161.
S7	Lago, P. and Vliet, H. (2005). **Teaching a course on software architecture**. In 18th Conference on Software Engineering Education and Training (CSEE&T), pages 35–42.
S8	Wang, A. and Stalhane, T. (2005). **Using post mortem analysis to evaluate software architecture student projects**. 18th Conference on Software Engineering Education and Training (CSEE&T), pages 43–50.
S9	Jarzabek, S. and Eng, P. (2005). **Teaching an advanced design, team-oriented software project course**. 18th Conference on Software Engineering Education and Training (CSEE&T), pages 223–232.
S10	Wang, A. and Sørensen, C. (2006). **Writing as a tool for learning software engineering**. Software Engineering Education Conference, Proceedings, 2006(7491):35–42.
S11	Golden, E. and Bass, L. (2007). **Creating Meaningful Assessments for Professional Development Education in Software Architecture**. In 20th Conference on Software Engineering Education & Training (CSEE&T), pages 283–290.
S12	McGregor, J. D., Bachman, F., Bass, L., Bianco, P., and Klein, M. (2007). **Using an architecture reasoning tool to teach software architecture**. 20th Conference on Software Engineering Education & Training (CSEE&T), pages 275–282.
S13	Svahnberg, M. and Martensson, F. (2007). **Six years of evaluating software architectures in student projects**. Journal of Systems and Software (JSS), 80(11):1893–1901.
S14	Wang, A. I., Arisholm, E., and Jaccheri, L. (2007). **Educational approach to an experiment in a software architecture course**. In 20th Conference on Software Engineering Education & Training (CSEE&T), pages 291–298.
S15	Andrade, R. M. and Arakaki, R. (2007). **Teaching Software Architecture Quality based on run-time metrics**. In International Conference on Engineering Education (ICEE), pages 1–6.
S16	Chenoweth, S., Ardis, M., and Dugas, C. (2007). **Adapting cooperative learning to teach software architecture in multiple role-teams**. ASEE Annual Conference and Exposition, Conference Proceedings.
S17	Gast, H. (2008). **Patterns and traceability in teaching software architecture**. In 6th PPPJ, pages 23–31.
S18	Mannisto, T., Savolainen, J., and Myllarniemi, V. (2008). **Teaching Software Architecture Design**. In 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), pages 117–124.
S19	Boer, R., Farenhorst, R., and Vliet, H. (2009). **A community of learners approach to soft ware architecture education**. In 22nd Conference on Software Engineering Education and Training (CSEE&T), pages 190–197.
S20	Wang, A. I. (2009). **Post-mortem analysis of student game projects in a software architecture course: Successes and challenges in student software architecture game projects**. 1st International IEEE Consumer Electronic Society's Games Innovation Conference, ICE-GiC 09, pages 78–91.
S21	Wu, B., Wang, A. I., Strom, J.-E., and Kvamme, T. B. (2009). **XQUEST used in soft- ware architecture education**. In International Consumer Electronics Society's Games Innovations Conference (ICE-GIC), pages 70–77.
S22	Gu, Q., Lago, P., and van Vliet, H. (2010). **A Template for SOA Design Decision Mak- ing in an Educational Setting**. In 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications, pages 175–182. IEEE

Continued on next page

Table 1 – *Continued from previous page*

ID	Reference
S23	Giraldo, F. D., Ochoa, S. F., Herrera, M., Neyem, A., Arciniegas, J. L., Clunie, C., Zapata, S., and Lizano, F. (2011). **Applying a distributed CSCL activity for teaching software architecture**. In i-Society, pages 208–214.
S24	Wang, A. I. (2011). **Extensive Evaluation of Using a Game Project in a Software Architecture Course**. ACM Transactions on Computing Education, 11(1):1–28.
S25	Wang, A. I. and Wu, B. (2011). **Using game development to teach software architecture**. International Journal of Computer Games Technology, 2011.
S26	Christensen, H. B. and Corry, A. (2012). **Lectures abandoned: active learning by active seminars**. In 17th Conference on Innovation and Technology in Computer Science Education (ITiCSE), pages 16–21, Haifa, Israel.
S27	Wu, B. and Wang, A. I. (2012). **Comparison of learning software architecture by developing social applications versus games on the android platform**. International Journal of Computer Games Technology.
S28	Urrego, J. S. and Correal, D. (2013). **Archinotes: A tool for assisting software architecture courses**. In 26th CSEET, pages 80–88.
S29	Cleland-Huang, J., Babar, M. A., and Mirakhorli, M. (2014). **An inverted classroom experience: Engaging students in architectural thinking for agile projects**. In 36th International Conference on Software Engineering (ICSE).
S30	Kiwelekar, A. W. and Wankhede, H. S. (2015). **Learning objectives for a course on software architecture**. In ECSA, volume 9278, pages 169–180.
S31	Rupakheti, C. and Chenoweth, S. (2015). **Teaching software architecture to undergraduate students: An experience report**. In 37th International Conference on Software Engineering (ICSE), pages 445–454.
S32	Grbac, T. G., Car, Z., and Vukovic, M. (2015). **Requirements and Architecture Modeling in Software Engineering Courses**. In Proceedings of the 2015 European Conference on Software Architecture Workshops, volume 07-11-Sept, pages 1–8, New York, NY, USA. ACM.
S33	Cervantes, H., Haziyev, S., Hrytsay, O., and Kazman, R. (2016). **Smart Decisions: An Architectural Design Game**. In 38th International Conference on Software Engineering (ICSE), pages 327–335.
S34	Georgas, J. C., Palmer, J. D., and McCormick, M. J. (2016). **Supporting software architecture learning using runtime visualization**. In 29th CSEET, pages 101–110.
S35	Ciancarini, P., Russo, S., and Sabbatino, V. (2016). **A Course on Software Architecture for Defense Applications**. In 4th International Conference in Software Engineering for Defence Applications, pages 321—330.
S36	Angelov, S. and Beer, P. (2017). **Designing and applying an approach to software architecting in agile projects in education**. Journal of Systems and Software, 127:78–90.
S37	Montenegro, C. H. and Astudillo, H. (2014). **A role-playing game to teach ATAM (Ar- chitecture Trade-off Analysis Method) a simulation tool and case study**. In IEEE ANDESCON 2014.
S38	Deursen, A., Aniche, M., Aue, J., Slag, R., de Jong, M., Nederlof, A., and Bouwers, E. (2017). **A Collaborative Approach to Teaching Software Architecture**. In ACM Technical Symposium on Computer Science Education (SIGCSE), pages 69–80.
S39	Tsur, E. E. (2018). **Delivering the fundamentals of software architecture, design and abstraction by developing a ray tracer for 3-dimensional graphical scenes**. Computer Applications in Engineering Education, (November 2017):1–10.
S40	Greising, L., Bartel, A., and Hagel, G. (2018). **Introducing a deployment pipeline for continuous delivery in a software architecture course**. In 3rd European Conference of Software Engineering Education (ECSEE), pages 102–107, Seeon/Bavaria, Germany.
S41	Vidoni, M., Montagna, J. M., and Vecchietti, A. (2018). **Project and team-based strategies for teaching software architecture**. IJEE, 34(5):1701–1708.
S42	Wedemann, G. (2018). **Scrum as a method of teaching software architecture**. In 3rd European Conference of Software Engineering Education (ECSEE), pages 108–112, Seeon/Bavaria, Germany.
S43	Lieh, O. E. and Irawan, Y. (2018). **Teaching Adult Learners on Software Architecture Design Skills**. In FIE. IEEE.
S44	Lieh, O. E. and Irawan, Y. (2018). **Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course**. In IEEE Frontiers in Education Conference (FIE), pages 1–9.
S45	Lago, P., Cai, J., De Boer, R., Kruchten, P., and Verdecchia, R. (2019). **DecidArch: Playing cards as software architects**. In 52nd Hawaii International Conference on System Sciences (HICCS), pages 7815–7824.
S46	Ouh, E. L. and Irawan, Y. (2019). **Applying case-based learning for a postgraduate software architecture course**. In Conference on Innovation and Technology in Computer Science Education (ITiCSE), pages 457–463.
S47	Backert, M., Blum, T., Kreuter, R., Paulisch, F., and 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), pages 1–6. IEEE.
S48	Lieh Ouh, E., Kok Siew Gan, B., and Irawan, Y. (2020). **Did our Course Design on Software Architecture meet our Student's Learning Expectations?** In 2020 IEEE Frontiers in Education Conference (FIE), volume 2020-Octob, pages 1–9. IEEE.
S49	Gonçalves, A. C., Vicente Graciano Neto, V., Ferreira, D. J., and Ferreira Silva, U. (2020). **Flipped Classroom Applied to Software Architecture Teaching**. In 2020 IEEE Frontiers in Education Conference (FIE), volume 2020-Octob, pages 1–8. IEEE.
S50	Capilla, R., Zimmermann, O., Carrillo, C., and Astudillo, H. (2020). **Teaching students software architecture decision making**. In ECSA 2020, pages 231–246.

3. Results

The studies included in our SMS cover a period of 28 years, with three studies published before 2003 and most of them (47) after that year, as shown in Figure 2. The higher num-

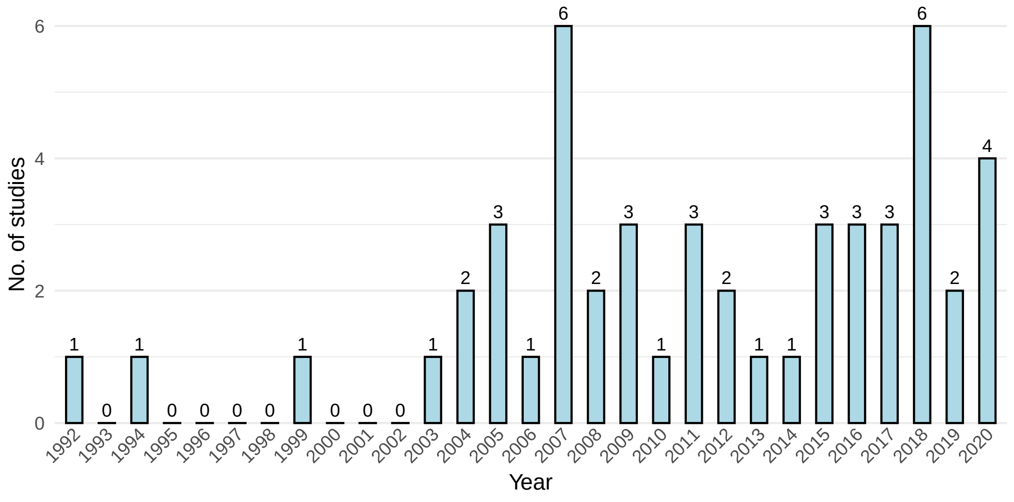

Figure 2. Number of studies by year from 1992 to 2020

ber of studies after 2003 may be related to the popularization phase of software architecture, as discussed in [Shaw and Clements 2006]. The most cited textbook on software architecture (entitled *Software Architecture in Practice* [Bass et al. 2003]) was published in that year as well, and its influence is present in the studies we investigated in this SMS. Regarding the publication venue, most experiences were published in event proceedings (i.e., 44 studies), whereas only six works were published in journals.

We also categorized the studies according to a broad classification of teaching experiences, i.e., course, activity, tool, game, and industry training, as shown in Figure 3. The most frequent category is **course**, which accounts for 36 studies. This category represents experiences in universities that have computing-related courses, such as computer science, computer engineering, and information systems. Studies in this category are still split into single or multiple experiences. **Single experience** encompasses studies that reports only one instance of a course whereas **multiple experience** reports on two or more (the highest number of course instances reported on was 28). The remaining categories account for 14 studies split in four additional categories: activities, tools, games, and industry training. **Activities** refer to single educational instructions that take place in one day for a limited amount of time, and have been split into single and multiple experiences as well (which refers to how many times an activity has been conducted). **Games** propose some sort of non-digital game (e.g., card games) to support or provide means for learning. With the same goal are **tools**, which represent software developed to specifically support activities within experiences. Finally, **industry training** accounts for training experiences that took place in industry or targeting industry professionals. This category is split into single and multiple experiences as well.

The following four subsections present the answer for each RQ previously defined.

3.1. RQ1. How has software architecture been taught?

Aiming to understand the curricular program of software architecture courses reported by the included studies, we investigated whether these courses were planned as a whole independent course, or were included as a sub-topic within a broader course. We found that 61.1% (22 studies) proposed an exclusive course to teach software architecture, while

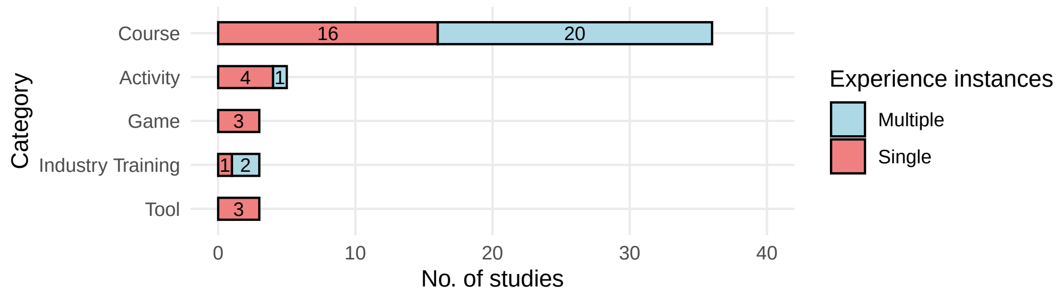

Figure 3. Classification of studies

22.2% (8 studies) included software architecture as a sub-topic of a software engineering or related courses. The remaining studies (16.7%) (6 studies) did not mention it explicitly. Regarding the course duration, we identified that 24% (12 studies) did not mention its duration, whereas 48% (24) had reported their duration ranging from five to 12 weeks. In particular, most courses (52.8% or 19 studies) had a duration of 12 weeks or more. Considering these numbers, we can say the duration of software architecture courses were not very different from other regular courses in computing programs (between three and six months).

Considering only those 36 studies that report an teaching experience during the courses, we also found that 44.4% of courses (16 studies) were proposed specifically as part of the curricula of graduate courses, and 30.6% (11) were defined at undergraduate studies level. Moreover, 11.1% (4 studies) were proposed for teaching software architecture in both undergraduate and graduate levels. Three courses (8.3%) involved industry professionals whereas two courses (5.6%) targeted graduate students and industry professionals. These percentages show some evidence that teaching software architecture for graduated students draw more attention than to others students. Moreover, most studies (2/3 or 24 studies) presented the number of students that enrolled in the courses. More specifically, 25% (9 studies) reported the course was lectured to class sizes smaller than 50 students, 27.8% (10) had between 51 and 100 students, 11.1% (4) had between 101 and 150 students, and 1 study had between 151 and 200 students. These numbers show that software architecture was taught to a good variety of class sizes regardless of the inherent challenges. The number of students attending a course, however, can be explained by external factors, such as those related to university characteristics or students' interests.

Considering the broad classification previously mentioned, for the category activity, five studies report diverse activities designed and applied to achieve particular learning objectives. An example is a 2-hour session in which students work as a team to extract architecturally significant requirements (S29, previously shown in Table 1). Other examples of activities are: S23, in which students of six universities distributed over different countries remotely participate in a computer-aided session to design the software architecture of a system. In S10 the *Writing to Learn* method is used for teaching the theory about software architecture tactics. Students individually wrote down every architectural tactic that they knew for achieving a list of quality attributes. In these studies, all activities had their outcomes assessed in different ways, such as student's feedback (gathered by means of oral evaluation (e.g., S10), questionnaires (e.g., S23) and controlled experiments (e.g., S29 and S49).

In the category tools, we found three software tools in three studies (S12, S28, and S34). These tools were developed specifically to assist software architecture courses. These tools are in fact GUI applications that provide functionality to manage and share architectural knowledge, or to visualize run-time metrics of a system under execution. Just one of the tools (S28) supports multi-user interaction, which is done by means of a mobile application installed in the student's phones to have access to a centralized repository.

Following the trend of adopting games in software engineering, three games were proposed for software architecture education (reported in S33, S37, and S45). Two are card games and one is an RPG (Role-Play Game), a classic and old game style in which players are assigned to a particular role for a given story. Both card games focus on the architectural decision process and can be fully played in sessions around one or two hours. The RPG focuses on ATAM (Architecture Tradeoff Analysis Method) for evaluating software architectures, relying on information from a case and from the ATAM specification for the assignment of roles to participants. This game was played in 1-hour sessions but it can take more time as in a real-world ATAM evaluation.

When it comes to industry experiences, we found three studies (S4, S6, and S47). Two of them (S6 and S47) report on long-term company programs for training professionals in the software architecture discipline. These programs have been improved over a considerable amount of years. The other study (S4) presents a particular training experience involving a development team of a large financial company facing problems with the integration of legacy systems.

We can observe that the RQ1 was able to provide us an overview of how software architecture has been taught along several years. We also observe that teaching has focused on different levels (undergraduate and graduate students, and professionals) with diverse course duration and formats (e.g., diverse activities) and using different means (e.g., games and tools). This panorama shows us that teaching software architecture (including its reporting through scientific studies) has been a concern and a diversity of initiatives have emerged dealing with different and complementary perspectives.

3.2. RQ2. What topics have been taught in the software architecture area?

We identified a broad variety of software architecture topics addressed by the studies included in our SMS. Figure 4 shows all topics and their frequency (one study sometimes considered more than one topic). Most of them address quality attributes, such as performance, portability, and fault tolerance. This finding was expected as the quality attributes refer to a basic requirement for addressing core architectural concepts, such as design and evaluation. The architectural evaluation method **ATAM** was the second most frequent topic in the list. ATAM is a well-known method for evaluating software architectures, and its conduction takes into account a number of core concepts associated with software architecture, such as quality attributes, trade-offs, scenarios, and decisions. So teaching ATAM can provide a learning opportunity for several topics at different cognitive levels.

Following ATAM, we found well-known topics, such as architectural design, styles, patterns, tactics, views, viewpoints, and trade-off analysis. Moreover, quality attribute scenarios and the method ADD (Attribute-Driven Design) were also considered in a relevant number of studies. Other important topics were also found but addressed in a few studies, as shown in Figure 4. Moreover other 30 different topics were addressed in

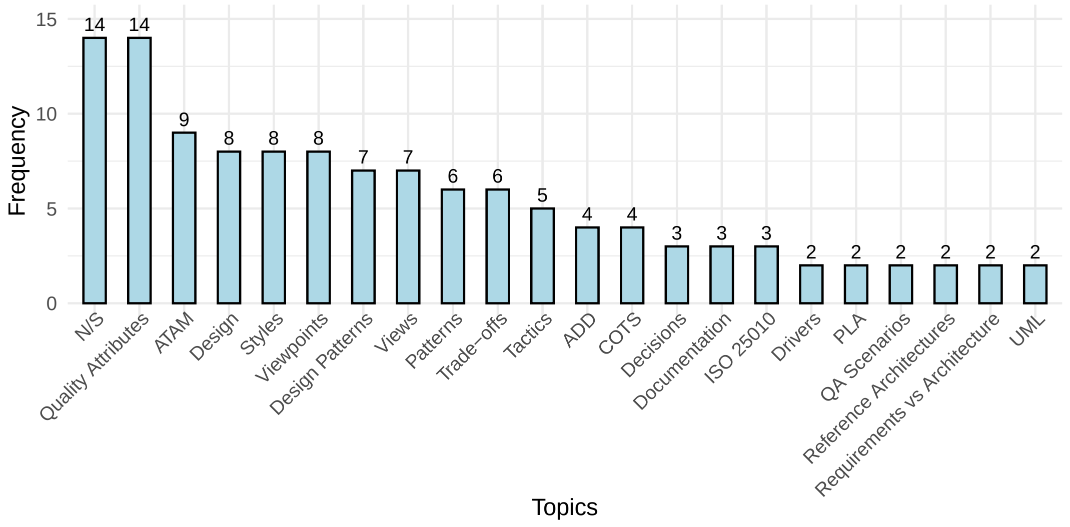

Figure 4. Topics addressed in two or more studies

single studies; they are listed in the external material.

The answer for this RQ shows us that diverse topics in the software architecture area have been taught. However, the two most taught ones are directly related to *quality attributes*, pointing out their relevance in the context of software architecture. We also observe that those topics were directly influenced by textbooks published on software architecture, including [Shaw and Garlan 1996, Bass et al. 2003, Van Vliet 2008, Clements et al. 2010, Rozanski and Woods 2011], as they have been used as reference in many experiences. In particular, an influential textbook is the second edition of "*Software Architecture in Practice*" [Bass et al. 2003].

3.3. RQ3. Which learning objectives have been considered in software architecture teaching?

In our SMS, we also investigated the studies regarding their learning objectives. An important finding is that most studies did not refer to any instrument, such as the original Bloom's Taxonomies [Bloom et al. 1956] or Revised Bloom's Taxonomies [Anderson et al. 2001], to support the definition of learning objectives (which refer to explicit formulations of what teachers intend students to learn [Anderson et al. 2001]). However, most studies (84% or 42 studies) stated explicitly one or more learning objectives, even when not employing the usual definition given by [Bloom et al. 1956] or [Anderson et al. 2001]. Only eight studies did not mention learning objectives (S2, S6, S8, S11, S17, S27, S28, and S50.

Aiming to better understand the studies included in our mapping regarding their learning objectives, we deeply analyzed and interpreted all 50 studies and classified their learning objectives within one (or more when pertinent) of the six cognitive levels (i.e., remember, understand, apply, analyze, evaluate, and create) [Anderson et al. 2001]. More specifically, **remember** refers to the capacity of retrieving relevant knowledge from long-term memory, **understand** refers to constructing meaning from instructional messages (e.g., oral, written, and graphic communication), **apply** refers to carrying out or using a procedure in a given situation, **analyze** refers to breaking material into smaller parts, determining how such parts relate to each other and to an overall structure, **evaluate** refers to making judgments based on criteria and standards, and **create** refers to forming a coherent or functional whole by putting elements together as well as to reorganizing elements to create a new pattern or structure [Anderson et al. 2001]. As result, we found that emphasis was given to four levels: understand, apply, evaluate, and create, while remember and analyze appear in just a few studies, as illustrated in Figure 5.

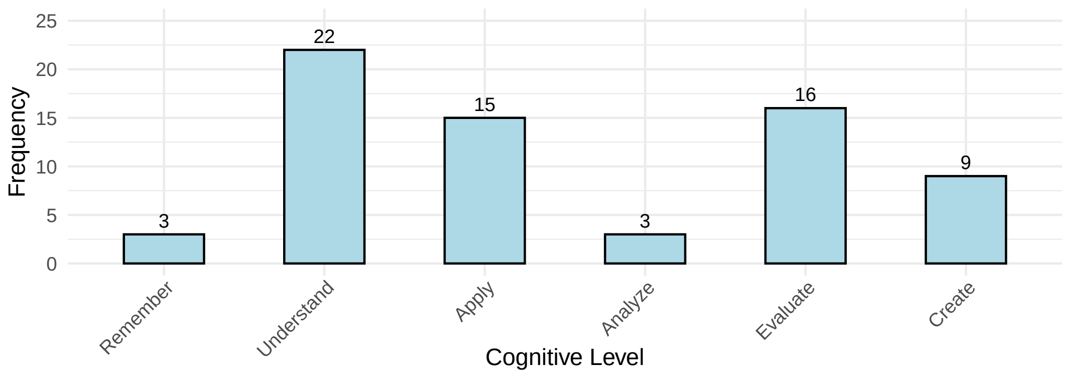

Figure 5. Bloom Cognitive Process Levels addressed in Learning Objectives

Understand appears in several learning objectives (22 studies) related to basic concepts, such as quality attributes, architectural styles, patterns, etc., whereas **apply** (15 studies) is related to the application of methods, such as ATAM, ADD, and trade-off analysis or even abstract concepts, such as styles and patterns. **Evaluate** (16 studies) is mostly present due to the high employment of ATAM for architecture evaluation. We understand, however, that applying ATAM does not imply the capability for evaluating software architectures as it possibly requires way more efforts to be developed for an individual. Lastly and perhaps the most expected level is **create**, which is surprisingly addressed in several (9) but not all studies, and is concerned with the definition (or creation) of an architecture for a specific or open-ended problem.

This RQ reveals that the cognitive process levels were considered in the learning objectives (reported by the studies) but more attention was paid in some of them. Remember is not less important than the other levels. In fact, it is required for solving more complex problems, applying methods or conducting procedures. However, remember as an ultimate goal for a software architecture course would not make much sense. On the other hand, analyze is a core cognitive process for many architectural activities but was not explicitly targeted in learning objectives of most studies.

3.4. RQ4. Which learning methods have been adopted to teach software architecture?

Different learning methods (which include resources and approaches) were adopted by one or more studies, as illustrated in Figure 6. In particular, learning methods refer to methodologies that put the students in a passive or active position towards the learning objective and uses physical, virtual or theoretical resources to support this process. However, it is important to highlight that most studies did not explicitly address learning methods/theories/practices or even the literature related to the education area, as expected in studies that deal with education.

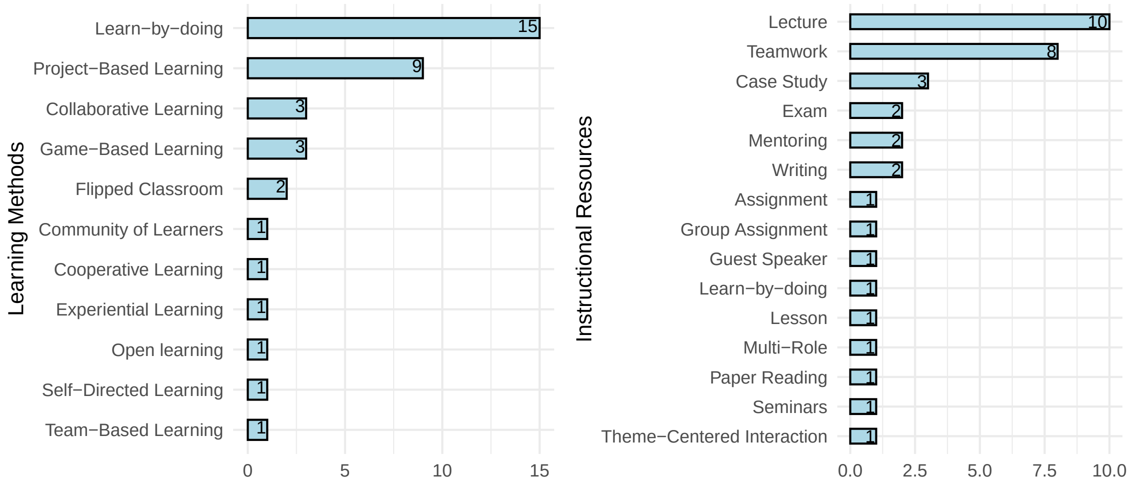

Figure 6. Learning Methods and Resources used by the Studies

We observe that traditional teaching methods that instructors have commonly used to present content to students (e.g., Learn-by-doing and Lecture) have been also adopted to introduce fundamental topics/concepts associated with software architecture. The most frequent approach is **Learn-by-doing** (found in 16 studies), in which learners perform one or a few architectural activities, such as design (reported for example in S7 and S18) or evaluation (e.g., S13 and S20), based on knowledge acquired before or during the actual experience. **Lectures** are also part of many experiences reported (10 studies).

Another interesting approach is **Project-Based Learning** that follows up as an way to mimic the software development in industry. We observe that projects with different sizes were used, from small projects (as those used in S3, S9, and S39) to large projects developed in partnership with software companies (as in S1, S2, and S4). Independently from the projects' size, studies reported advantages of this learning method, such as promoting soft skills by means of teamwork and developing good practices which can be later transferred to industry. In some cases, the projects are supposed to deliver value to real-world stakeholders. In this direction, **teamwork** is an approach that can involve different stakeholders and is frequent in our mapping (found in 8 studies).

There are also several other approaches but to a lesser extent in a few studies. For instance, **case studies** have been employed as a mean to promote learning based in real-world systems and architectures. In general, this approach was used in a later phase of the experiences to promote relevant discussions and show how software architecture concepts are present in real-world projects (as discussed in S3, S30, and S38). Another perspective is to ask learners to conduct case studies, gathering relevant information from real-world software projects, and documenting the findings (as discussed in S38).

Game-Based Learning was found in three studies (S33, S37, and S45) that describe the development and employment of games to support the learning process of specific topics, such as ATAM, Software Architecture Design Decision Making (SADDM) and Attribute-Driven Development (ADD). This approach has only been explored more recently and the games developed have usually followed a card style, requiring face-to-face interaction among players. **Collaborative Learning** also appears in a few studies

(S38, S50, and S23) and refers to practices where all students contribute somehow to achieve a particular goal such as the definition of an architecture or the writing of a book with knowledge acquired through the course. Many other interesting learning methods were explored to teach software architecture, and reported in one or two studies, as illustrated in Figure 6.

This RQ shows us that the studies employed a wide range of learning methods and resources but a great amount of methods were explored in just a few studies. The most recurring methods emphasize the practical nature of software architecture, as discussed in [Galster and Angelov 2016]. However, more experiences exploring other methods can help understand their benefits and drawbacks when they come to the software architecture education.

4. Discussion

This section presents the main findings of our work, some future research perspectives, and threats to validity.

The distribution of studies included in our SMS over the years made us believe that software architecture education&training is still an important topic to be investigated, even though the first study ever reported dates 1992. Considering the time span analyzed (1992-2020), 80% of studies were published after 2006 (i.e., the median year); this may indicate an increasing interest in the topic from the software architecture community, including researchers, events, and journals in the field. We also observe that even after many studies have been conducted and published, the difficulties and challenges of teaching and learning software architecture still remain.

Another finding worth highlighting is that the target audience of courses (or experiences teaching software architecture) was split into nearly two halves, undergraduate and graduate (Master/Ph.D.) students. In just one case, professionals were also part of the target audience. One may believe that teaching software architecture to graduate students would be more suitable as they are more likely to understand and learn the concepts due to prior experience with theory and practice. On the other hand, one may find it suitable to start teaching undergraduate students so they can understand software architecture concepts as early as possible, enabling them to develop capabilities to better deal with real-world software projects.

Regarding the software architecture topics addressed by the studies, an emphasis has been put on fundamental concepts associated with software architecture as well as well-known methods, like ATAM. However, new architectural styles and patterns, such as microservices and serverless, cloud-native architectures, IoT architectures, were not addressed, although they are trending in several domains and industry projects, so proper education regarding them can be very relevant. Other current relevant topics related to continuous architecting in the context of agile development have not been considered; only one study addresses it (S40). in particular, in this study, students conduct a project in which a continuous delivery pipeline is developed. Therefore, we can say that the teaching of software architecture should also encompass new trends, including those relevant to the current real-world project.

The most recurrent learning methods used to teach software architecture were those that we expected to find, following the nature of other engineering-related courses.

Less common and intuitive approaches were found as well, showing us a continuous interest in innovating the learning experiences. Teaching with games, for example, has only been addressed more recently, but its potential can improve the education of new generations of software architects/engineers. Collaborative approaches, which were explored in a few studies, can be attractive due to an increase of remote/hybrid job positions (most of them due to the pandemic). However, many supportive tools and platforms are still required to enable fast and suitable collaboration.

Our investigation was also interested in finding a correlation between the topics taught and the learning methods, but no relevant correlation was found in the studies. Therefore, it would be valuable for future investigation to find if such a correlation exists, aiming at success in the software architecture education experiences. We also intend to investigate whether the learning methods adopted are related to the instructional resources used and whether the learning objectives match the instructional activities employed.

4.1. Threats to Validity

The threats to the validity of our SMS are listed below along with the measures taken to mitigate their impact. We followed the classification proposed in [Wohlin et al. 2012], which is composed of four threat categories:

- **Internal validity**: During the data extraction from primary studies, in some cases, the authors did not clearly provide information with sufficient detail about one or more concepts, which may have led us to misconceptions. In this sense, some pieces of information had to be interpreted during our analysis. To minimize this threat, we assumed as less as possible and report the cases in which the information necessary to answer our research questions were not available or not clear to allow interpretation. In some cases the reviewers participated in consensus meetings to mitigate misinterpretation of some data extracted;
- **External validity**: In this category, another possible threat was the incorrect selection of primary studies according to the inclusion/exclusion criteria. To mitigate this threat, we had three reviewers participating during the selection process, ensuring all studies were reviewed by at least two of them. Consensus was achieved with the help of the third reviewer when necessary. Also, we created a detailed protocol, which was validated prior to the execution phase by members of our research group;
- **Construct validity**: Missing relevant studies was big a concern for this study. In some cases, a relevant study may not be found as a result of not having any of the search string terms in their title, abstract, or keywords. To mitigate this threat, we performed several interactions of a pilot test using the Scopus database and calibrated our search string analyzing carefully the results and confirming that the terms contained in our string were capable of finding a list of core studies we already knew prior to this study. Another construct validity threat might be our judgment concerning the selection of each study; hence, to minimize this threat, the decisions of inclusion or exclusion of studies were discussed by all researchers from this SMS; and
- **Conclusion validity**: Since some data had to be interpreted, there was a potential threat to the reliability of the results. To minimize this threat, we performed several brainstorming sessions to better define all elements of the SMS protocol (e.g., research questions, search string, databases, and selection strategy). Besides, the data extracted was

carefully reviewed by all authors of this work and any conflict was solved in a meeting using consensus.

5. Conclusion

The knowledge and experience in software architecture (which can be to some extent gathered via training courses) are sometimes essential for students who initiate in software companies. Hence, software architecture education is an important subject for curricula of computing-related courses. In this scenario, we provide an overview of the state of the art in the field of software architecture education through a deep analysis of 50 studies published from 1992. Different experiences were found, and we elicited important information, including the topics taught, the cognitive process categories (which are usually aimed by teachers and instructors when defining learning objectives), and the learning methods used. Such information can provide the current panorama of the field as well as the future directions to be followed to mature this field. Despite many years of research in the software architecture area, we can also conclude that education in this area still remains a challenging task.

Acknowledgment

This study was financed in part by CAPES - Brasil (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) (Finance Code 001 and Grant No.: 88882.328820/2019-01), National Council for Scientific and Technological (CNPq) (Grant No.: 313245/2021-5), and São Paulo Research Foundation (FAPESP) (Grant No.: 2015/24144-7).

References

Anderson, L., Krathwohl, D., and Bloom, B. (2001). *A taxonomy for learning, teaching, and assessing: a revision of Bloom's taxonomy of educational objectives*. Longman.

Angelov, S. and Beer, P. (2017). Designing and applying an approach to software architecting in agile projects in education. *Journal of Systems and Software*, 127:78–90.

Bass, L., Clements, P., and Kazman, R. (2003). *Software Architecture in Practice*. Addison-Wesley, 2nd edition.

Bass, L., Clements, P., and Kazman, R. (2021). *Software Architecture in Practice*. Addison-Wesley Professional, 4th edition.

Bloom, B., Engelhart, M., Furst, E., Hill, W., and Krathwohl, D. (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals: Handbook I Cognitive Domain*. David McKay Company, Inc.

Cervantes, H., Haziyev, S., Hrytsay, O., and Kazman, R. (2016). Smart Decisions: An Architectural Design Game. In *38th International Conference on Software Engineering (ICSE)*, pages 327–335.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., and Stafford, J. (2010). *Documenting Software Architectures: Views and Beyond*. Addison-Wesley Professional.

Deursen, A., Aniche, M., Aué, J., Slag, R., de Jong, M., Nederlof, A., and Bouwers, E. (2017). A Collaborative Approach to Teaching Software Architecture. In *ACM Technical Symposium on Computer Science Education (SIGCSE)*, pages 69–80.

Galster, M. and Angelov, S. (2016). What makes teaching software architecture difficult? In *38th International Conference on Software Engineering (ICSE)*, pages 356–359.

Garlan, D., Shaw, M., Okasaki, C., Scott, C., and Swonger, R. (1992). Experience with a course on architectures for software systems. In *6th SEI Conference on Software Engineering Education*, pages 23–43.

Lago, P., Cai, J., De Boer, R., Kruchten, P., and Verdecchia, R. (2019). DecidArch: Playing cards as software architects. In *52nd Hawaii International Conference on System Sciences (HICCS)*, pages 7815–7824.

Lago, P. and Vliet, H. (2005). Teaching a course on software architecture. In *18th Conference on Software Engineering Education and Training (CSEE&T)*, pages 35–42.

Mannisto, T., Savolainen, J., and Myllarniemi, V. (2008). Teaching Software Architecture Design. In *7th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, pages 117–124.

Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18.

Rodrigues, C. and Werner, C. (2009). Software architecture teaching: A systematic review. In *9th IFIP World Conference on Computers in Education (WCCE)*, pages 1–10.

Rozanski, N. and Woods, E. (2011). *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Pearson Education.

Rupakheti, C. and Chenoweth, S. (2015). Teaching software architecture to undergraduate students: An experience report. In *37th International Conference on Software Engineering (ICSE)*, pages 445–454.

Shaw, M. and Clements, P. (2006). The Golden Age of Software Architecture: A Comprehensive Survey. Technical Report CMU-ISRI-06-101, Institute for Software Research International.

Shaw, M. and Garlan, D. (1996). *Software Architecture - Perspectives on an Emerging Discipline*. Prentice Hall.

Van Vliet, H. (2008). *Software engineering: principles and practice*, volume 13. John Wiley & Sons Hoboken, NJ.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *18th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, pages 1–10.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Springer Berlin Heidelberg.