International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

Aligning Software Architecture Training with Software Industry Requirements

W. Libardo Pantoya Y.
System Departament, Universidad del Cauca, Calle 5 No 4-70
Popayán, Cauca,Colombia
wpantoja@unicauca.edu.co
http://www.unicauca.edu.co

Julio Ariel Hurtado A.
System Departament, Universidad del Cauca, Calle 5 No 4-70
Popayán, Cauca, Colombia
ahurtado@unicauca.edu.co

Arvind Kiweleker
Department of Computer Engineering, Technological University Lonere-402 103, Raigad
Raigad, India
awk@dbatu.ac.in

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

The activities of software design, documenting, and evaluating the structure of software systems, referred to as Software Architecture, have been increasingly getting significant attention in industries. This situation is because of the explicit and prominent role assigned to quality attributes while developing software systems. Considering the high relevance of Software Architecture to industry, many academic institutes have introduced a course on Software Architecture as a part of undergraduate programs in Software Engineering. However, teachers offering this course face numerous challenges. Some of these challenges stem from how software architecture is practised in industries and others from teaching them in an academic setting. This paper describes an experience of designing a software architecture course that aligns the competencies expected from professional software architects with teaching practices imparting those competencies. Such an alignment is necessary to improve the employability of graduate students, make their progression from academic institutes to industries an effortless one, and for better learning outcomes. In the absence of such alignment, fresh graduates need to re-train, leading to training costs and delayed recruitment of fresh graduates by their prospective employers. The experience reports recurring challenges observed by earlier researchers, strategies to address them and our experience in implementing those strategies. The teaching strategies suggested are potentially helpful and practical, especially to less-experienced instructors teaching a course on Software Architecture.

Keywords: Software Engineering Education;Software Architecture Training;Industry Relevant Skills;Software Architecture Modeling and Competencies;Software Architecture

Curriculum;Instructional methods and strategies.

1. Introduction

Taking inspiration from the Civil Engineering and Building Construction disciplines, Software Engineering has adopted the ideas of architecture and quality as a set of externally visible properties of software systems [42]. Software Architecture has been recently emerged as a sub-discipline within Software Engineering to play a critical role in assuring the quality of software products [39], [14]. The significance of Software architecture to attain business goals has also been continuously emphasized by practitioners [26]. In case of a software failure, individuals and businesses experience minor inconvenience to catastrophic consequences. Therefore, the software shall meet expected quality parameters and support them as long as the software is operational and is orchestrating business processes [9].

Traditionally, undergraduate programs in engineering such as Computer Science and Engineering, Information Technology, Software Engineering, and Systems Engineering include many industry-relevant courses. These courses impart adequate skills and technical knowledge related to programming languages and development platforms [32]. However, students of these programs have little knowledge about software architecture and design issues [41] despite its increasing significance to software industries. Especially fresh graduates lack sufficient skills to make good design decisions, apply design practices, patterns, and principles [41] when they are faced with the task of providing technical solutions to a business problem.

The existing literature on Software Architecture education and practice points to different reasons for this lack of skills related to software architecture and design [18]. First and foremost is a considerable difference between the academic perception of software architecture and its practice in the industrial sector [38]. Sometimes, the problems industries consider the most critical and challenging are not given due importance for research or training in universities [4]. Second, there exists a gap between the skills expected from graduates in areas of Software Engineering and skills taught in the curricula [1]. Third many institutes have no clear vision about the topics on which software architects shall be trained [41].

This article aims to bridge the gap between these industry requirements and educational practices in the context of Software Architecture training. We identified the recurrent challenges mentioned in the literature to provide software architecture training. We classify these challenges into two categories—the first kind of challenges originate from industrial practices, and the second deal with the conduct of Software architecture courses and training in university settings. In addition to this, we specify some of the strategies to deal with these challenges. We also describe our experience of implementing these strategies at our University.

In this way, the experience reported in the paper contributes in the following way:

(1) It identifies the challenges faced by educators to train students on software

architecture skills as expected in software industries.
(2) The challenges are classified in two categories as challenges originated from the practice of software architecture by industry professionals and as educators teach them.
(3) The paper further specifies strategies to address both kinds of challenges.
(4) An evaluation of implementing those strategies at our University, is presented.

The strategies presented in the paper have the potential to be helpful to less-experienced teachers. They may reduce the cost and time to induct the freshly graduated students by prospective employers. Thus they can make the progress of fresh graduates from university campuses to the business community a smoother one.

The rest of the paper is organized as follows: Section 2 motivates undertaking the experience in the context of existing literature. The detailed design of the experience is presented in Section 3. Section 4 introduces the software architecture concepts that recur regularly in the literature. Section 5 and 6 describe the challenges to impart software architecture knowledge and skills in a university setting and strategies to overcome them. Our experiences teaching a course on software architecture and its evaluation are presented in sections 7 and 8.

2. Related work

Researchers have been exploring educational and training aspects of Software Engineering Education in general and notably software architecture. A substantial amount of literature presents several course initiatives, both at the undergraduate and postgraduate levels. This section describes some of the current and the most relevant attempts developed by researchers dealing with teaching a Software Architecture course.

The work of Zheng Li et al. [35] proposes an educational strategy of adopting minor projects to teach software architecture concepts. This work aims to investigate the benefits of using little projects with free public resources. Such minor projects help students (i) accumulate architectural experience and (ii) strengthen the fundamental architectural knowledge.

Li et al., [34] present content and pedagogical aspects of a software architecture course for undergraduate students, including typical case studies and various class discussions. The learning objectives of the course proposed by Li et al. [34] include: (i) to understand stakeholders and their concerns, (ii) to analyze quality attributes and how to ensure them, and (iii) to apply architectural styles and patterns for architecture design. Further, Li et al., [34] observe that architectures are high-level abstractions describing a solution; however, students need to deal with many developments and operational problems using programming languages. Hence, students must work on more business cases, particularly requiring the reuse of existing components and systems.

Deursen et al. [52] propose a collaborative software architecture course in which participants work together to study and document a sizeable open-source software system of their own choice. The course objectives include fostering an open learning environment and adopting publishing results as an online book to be accessed by a broader open source community. Their experience suggests that (i) students gain real-life experience when they work on open-source systems (ii) students strengthen their architectural knowledge by making code contributions to open-source projects, (iii) by working together on a joint book helps students look beyond their work and to study architectural descriptions produced by the other groups.

Lago [31] presents a work that reports experiences with courses at the master's level in software architecture with a focus on communication aspects. It notes the significance of well-destined course content to teach the course effectively. Mannisto et al. [40] have designed a course taking into account industrial needs. The form of the course is similar to an industrial architecture studio assigned to a team of architects. Fraga y Llorens [16] propose a methodology based on ontological structures and reinforced learning.

A cursory look at the existing literature on software architecture education conveys that these course initiatives address various challenges and content issues and propose strategies to deal with them. However, such dispersed knowledge that describes some of the best practices hampers their adoption in an academic setting, especially by inexperienced teachers teaching software architecture. To unify this dispersed knowledge, we executed an experience of designing a course on Software Architecture that addresses industry requirements. This paper describes the result of the course and our experiences in addressing the challenges observed by earlier researchers.

3. Experience Design

We present the design of a course on Software Architecture. This experience investigates the phenomenon of designing an undergraduate course on an emerging topic for which no standard body of knowledge exists either at the global or national levels [7, 28]. We adopted the case-study-based approach for designing a course because it supports exploratory and investigative studies[45]. The experience presented in this paper is descriptive and exploratory as it aims to understand various aspects related to the training of software architects.

This experience is descriptive as it describes the challenges instructors face in software architecture design. It is an exploratory study because we survey various learning strategies from the literature and align our experiences of improving the course delivery over the last seven years. While executing the course, our objective is to maintain the integrity and significant characteristics of the events carried out to develop the course content, goals, and strategies for implementing the said course.

	Science Direct	IEEE Xplore	ACM Digital	Google Scholar
No of papers accessed	10	40	141	179
No of papers referenced after exclusion criteria	0	31	18	21

Table 1. Number of Papers accessed and used for the study from different sources

3.1. *Context*

This experience describes the design of a course on Software Architecture offered as a part of the undergraduate program at our University. The undergraduate program at the university has been accredited by a national-level accreditation board. The stringent quality assurance policy laid out by the National Accreditation Committee forces the program to be in a state of continuous improvement. So, the program is continuously evolving through a traditional methodology of design, implementation, and evaluation of learning outcomes.

Starting from the need to have an undergraduate course that considers software architecture topics, we restructured the sixth-semester courses called Software Engineering II and their laboratory activities. These courses are focused on software design aspects. The restructuring began in 2019. The 2021 curricular renewal process is quite well established by identifying teaching strategies, well-defined learning outcomes, and course content.

3.2. *Method*

We perceive the course design as an exploratory activity that relies on the experiences of designing similar courses from earlier researchers. To get the insights from the equivalent course design activity undertaken earlier, we adopted the method used to perform a systematic literature review[27]. However, our goal is not to conduct a comprehensive review of the literature by examining software architecture courses and mapping them to the research questions that drive the literature review. The literature survey aims to extract the relevant information applicable to the design and delivery of a course on software architecture. Hence, we executed the following steps.

(1) *Decide upon Digital libraries:* We used ScienceDirect, IEEEXplore, ACM Digital, and Google Scholar as digital sources following a non-systematic snowballing procedure outlined in [55] to extract relevant papers describing software architecture courses. Table 1 shows the number of papers collected from different sources.

INCLUSION CRITERIA
Studies presented in English focused on the training of software architects aligned with the needs of the industry.
Full papers published in peer-reviewed journals, conferences, or congresses.
The paper is based on empirical data and not only on the opinion of the authors.
A reader from a trusted source can access the full text.
EXCLUSION CRITERIA
The primary study does not present a solution that contributes to the training of software architects aligned with the needs of the industry.
Studies not accessible in full text.
Papers presenting non-peer-reviewed material.
Secondary or tertiary studies that report and analyze the results of other investigations. It is a duplicate publication by the same author.

Table 2. Criteria for inclusion and exclusion of the research.

(2) *Formulate search string to extract relevant papers:* Our initial search string was ("training" OR "teaching") AND ("software architecture" OR "software design") AND ("software industry" AND ("course"). We search for papers in the last 11 years, i.e., from 2009 to 2020.

(3) *Decide upon inclusion and exclusion criteria:* We used the inclusion and exclusion criteria (see Table 2) to determine which are the most relevant studies for the research and thus answer the questions posed. Therefore, we considered a study appropriate if it met all the inclusion criteria. On the other hand, it was not considered if a study met at least one exclusion criterion. Table 3 shows the papers we referred to for our final studies.

Sr. No	Title of the paper	Ref.
1	Role of Software Architect: A Pakistani Software Industry Perspective	[48]
2	What makes teaching software architecture difficult?	[17]
3	Project-Based Learning	[19]
4	Revealing the Gap Between Skills of Students and the Evolving Skills Required by the Industry of Information and Communication Technology	[1]
5	What do software architects think they (should) do? Research in progress	[49]
6	Teaching a Course on Software Architecture	[31]
7	A Collaborative approach to teaching software architecture	[53]
8	Teaching Software Architecture to Undergraduate Students: An Experience Report	[46]
9	The Challenge of Training New Architects: an Ontological and Reinforcement-Learning Methodology	[16]
10	A Community of Learners Approach to Software Architecture Education	[13]
11	The reflective practitioner perspective in software engineering education	[21]
12	Using Public and Free Platform-as-a-Service (PaaS) based Lightweight Projects for Software Architecture Education	[35]
13	Teaching Software Architecture Design	[40]
14	Industry trends in software engineering education: A systematic mapping study	[11]
15	Closing the gap between software engineering education and industrial needs	[18]
16	Exploration on theoretical and practical projects of software architecture course	[56]
17	Teaching adult learners on software architecture design skills	[37]
18	Teaching Reform and Practice of Software Architecture Design Course under the Background of Engineering Education	[33]
19	A Course on Software Architecture for Defense Applications	[10]
20	Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application	[54]
21	Did our Course Design on Software Architecture meet our Student's Learning Expectations?	[38]
22	Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course	[36]
23	Teaching Students Software Architecture Decision Making	[8]
24	Improved Teaching Model for Software Architecture Course	[24]
25	Designing and Applying an Approach to Software Architecting in Agile Projects in Education	[2]

Table 3. Papers used for final study

3.3. *Objective and Research Questions*

The introduction of a course on Software Architecture was driven by the motive of improving the level of acceptance by the Popayán software industry for the students who graduate with the System Engineering program. To achieve designing an industry-relevant course on Software Architecture, the experience tries to answer the following program-specific research questions.

(1) What are the recurring software architecture knowledge and skills that are frequently expected from freshly graduated students?
(2) What challenges does an instructor teaching a course on Software Architecture face while imparting these pieces of knowledge and skills?
(3) What teaching strategies does an instructor usually adopt to overcome those teaching challenges?

The following sections provide the insights developed by us while attempting to answer these questions. Seeking answers to these questions can help to improve the teaching practices and effective delivery of a course on software architecture aligned to industry requirements.

4. Recurring concepts in software architecture from industrial practice

Software architecture is defined as a structure or structures of a system, which organize software elements and their relationships [12]. The emphasis is on the externally visible properties of the pieces of software and how they are related. Software Architecture provides a mechanism for the establishment, documentation, and communication of a system's main design decisions. Therefore, it is significant in multiple ways.

Some of the frequently mentioned values of software architecture are [5]:

- It is a primary abstraction of the system that: people use to think, design, code, and communicate in terms of large conceptual blocks.
- It promotes high-level reuse and component reuse.
- It influences development productivity by reusing large frameworks to support the construction of product line.
- It assures quality throughout the software life cycle by explicitly treating quality attributes such as modifiability, portability, scalability, and security. As the size of the system grows, developers can trace the solutions to these problems by analyzing architecture.

This section elaborates on some of the recurring architectural concepts in industrial practices and courses on software architecture: (i) Quality Attributes, (ii) Architectural Tactics, (iii) Architecture Decisions, (iv) Architecture Decisions. These concepts recur in literature regularly, so we defined these concepts. Identifying these concepts is crucial to designing software architecture training programs.

4.1. *Quality Attributes*

Quality attributes are the characteristics that a software product shall satisfy. Each quality attribute is associated with specific metrics that define the levels of quality for a software product [47]. Few examples of quality attributes include security, reliability, performance, interoperability, and others. These attributes arise from highly complex business rules and quality concerns [15]. Improper handling of these quality attributes poses a business risk. The architect must consider the conflicts between the quality attributes known as architectural trade-offs. A software product has to meet several relevant qualities, and the decisions made to satisfy one quality attribute may affect another quality attribute. For example, designing software systems for high availability can affect security. Because to provide higher availability, redundancy of elements needs to be increased, adding more points of vulnerability to the solution, thus affecting security.

4.2. *Architecture Patterns*

Architecture patterns are common solution structures to a similar design problem that are well understood and documented [20]. Each pattern describes a general software system's structure or high-level behavior and is intended to satisfy a software product's functionality, qualities, and constraints. Architecture patterns are chosen in response to early design decisions, including decisions about satisfying functional requirements, non-functional requirements or quality attributes, and physical constraints such as the physical distance between user and service provider. Applying a pattern is generally documented as consequences, in which the positive and negative aspects of the solution are indicated.

Many of the benefits and responsibilities relate to achieving quality attributes. For example, an advantage of the layer pattern indicates that standardized interfaces between layers generally limit code changes on the layer being changed, supporting the ease of modifying the system.

4.3. *Architectural Tactics*

Architectural tactics are high-level abstractions that capture decisions made to achieve quality goals. Tactics can be design-time, such as *hiding information* to improve modifiability, or they can be run-time, such as *managing concurrency* to improve performance [20]. Architectural tactics impact software design in several ways. In some cases, a developer can quickly implement a tactic through a particular architecture pattern. For example, the *layering* pattern is a way to implement the modifiability tactic, which *restricts communication paths.* On the other hand, a tactic may require significant changes in the pattern's structure and behavior or need entirely new structures and behavior. In such cases, implementing tactics and future maintenance of a system will be more complex and tedious.

4.4. *Architecture Decisions*

Architecture decisions define the rules for how a system should be constructed. Architecture decisions usually involve the choices faced by an architect while designing a software system. Further it provides a selection made by the designer in a specific context along with its justification or rationale behind selecting the choice. The choices may be about the structure of the application or system, selection of a technology to implement the design, or a tradeoff between quality attributes. Whatever the context, an exemplary architecture decision helps development teams make the right technical choices [43]. Hence an architecture decisions needs to be explained in terms of following features:

- Available design alternatives.
- Justifying the decision.
- Documenting the decision.
- Effectively communicating that decision to the right stakeholders.

5. Software Architecture Teaching Challenges

This section reports the challenges as discussed in the existing literature on Software Architecture Education. The challenges are classified into two categories, i.e., the practice-related challenges and teaching-related challenges. The practice-related challenges arise from how software architecture design activities are performed in the software industry. Teaching-related challenges arise from the problems faced while delivering a course on Software Architecture.

5.1. *Practice Related Challenges*

As the disciplines of Law and Medicine, Software Architecture is a practice-oriented discipline. Some teaching challenges are rooted in how software architecture and design are practiced in the software industry. This section reviews some of the frequently reported challenges in the literature. Each challenge is labeled as Industrial Practice Challenge (IP-Cn.) These are:

[IP-C1] The role of software architects in the software development process is vaguely defined: According to the IEEE standard, 1471-2000 [23], the role of the architect in any project is crucial. According to this standard, an architect may be a person, a team, or an entire organization responsible for designing the fundamental structure of the system referred to as architecture. To establish the system's architecture, a software architect must understand the development process, know the business domain, and have analysis, development, and software development and operation skills. Furthermore, an architect must be a good communicator who knows and negotiate different aspects of the software project. For example, integrating organizational policies in decision-making during the project [48]. Although software architects are not project managers, an architect is considered a software project's technical leader.

A software architect needs to perform two kinds activities on regular basis. First kind of activities are related to project in hand and it includes activities such as listening to customers, driving the development teams, spending time to make the right design choices, validating them, and documenting them. The second kind of activities are related to profession in general and it includes activities such as monitoring emerging technologies and developing a long-term vision. [30].

The books and blogs [44, 50, 41, 3, 6] describe the role and responsibilities of software architects depending on the development processes, but there is no global consensus on this. These works emphasize the significance of a distinct role for software architects, but they diverge on their responsibilities. Further, these responsibilities vary from one organization to another.

[IP-C2] The software architect's knowledge and skills are unclear: The skills and knowledge level that a software architect should possess vary from organization to organization. This situation has serious consequences for planning a career map for graduate students. It would be convenient to have a model that indicates maturity levels for software architecture as a profession. Such maturity levels hierarchically organize knowledge and skills from entry-level graduate to expert level professional. Such maturity levels guide students to prepare a roadmap for pursuing a career in architecture design [17, 41].

[IP-C3] In practice, most architectural decisions happen in a team: Since most of the architectural choices occur in groups, learning architecture decision-making requires learning collaboration and communication with team members [8]. In team-based teaching activities, an instructor must differentiate the project process and the learning process [19]. Therefore, educators must continuously exchange their roles between instructor, coach, and stakeholders, among others.

[IP-C4] Architecture design in agile context: In Industry, architecture design is carried out in two different contexts. The first is in the conventional rigorous software engineering process, and the second is part of agile methodology.

More and more companies seek to incorporate the role of software architects in their development processes, but this role is still diffuse in terms of their responsibilities and competencies. Conventionally, it can be defined which activities an architect should engage in in the context of various development processes. But in the recent past, many companies have started adopting agile methodologies. In this context, architects' roles need to be redefined for organizations adopting agile methodologies [1] [2].

Architecture design in agile projects introduces more options, i.e., when to design, how much, and how to document architecture. In agile contexts, doing all that is necessary, but as little as necessary. The architectural freedom offered in agile methods adds to the vagueness perceived by students [17]. For a student, making a judgment about just enough architecture design is difficult because of a lack of experience.

[IP-C5] Architecture decision making needs experience: Software architects are in charge of proposing the quality attributes satisfying business goals for the software applications to be developed. According to Sherman and Unkelos-Shpigel [49], the architect's responsibility is to make design, engineering, and technological decisions during the software development process. Finding the right balance between software quality attributes such as security, performance, usability, availability, maintainability, interoperability is a complex task because when developers attempt to achieve one quality attribute, another deteriorates. Since these quality attributes conflict with each other, software architects shall know tactics, patterns, and principles that help them make decisions appropriate to a specific context. Thus, a pragmatic architect defines software systems by applying abstract knowledge, proven methods, recent technologies to create a maintainable solution. Besides making the right architectural decisions, architects shall communicate those decisions to different stakeholders.

Architecture decision-making requires extensive knowledge and expertise to make decisions, which goes on assimilating over time. It may take a lot of background studies to understand a specific technology or a domain. As a result, students face difficulties grasping the significance of architectural decisions.

[IP-C6] Architecture design happens in Software Engineering context: In software engineering practice, there is often a distinction between high-level architecture and low-level implementation activities. Overcoming this requires creating a mindset in students not to treat architecture design in isolation. To intertwine architecture with other architecture-related activities, educators must allow students to work on requirements, architecture, and implementation simultaneously [17].

Moreover, the realization of architectural decisions goes hand in hand with the advancements in technologies. A software architect must have a broad knowledge of technologies to decide which platforms, frameworks, and languages favor particular non-functional requirements [41]. The problem lies in keeping pace with the technological changes as these technologies rapidly evolve. Along with the rapid development of technologies, the required job skills also change quickly. As a result, it becomes difficult for students to assess the gap between their abilities and those evolving abilities. Although universities conduct regular curriculum assessments, the time required to implement those revisions take time. It is challenging for universities to cope with the complex and rapid changes occurring in the industry [1, 41, 31].

5.2. *Teaching Challenges*

The challenges belonging to this category are rooted in factors affecting teaching a course on Software Architecture either in the conventional classroom or online mode. These factors include teaching methods, availability of learning resources, and achieving better learning outcomes. This section reviews some of the frequently reported teaching challenges in the literature. Each challenge is labeled as Teaching Challenge (TC-n.) These are:

[TC-1] Multiple teaching methodologies: One of the main requirements for effective teaching is clear and adequate teaching methodologies. In this sense, the literature review shows several proposals for courses and teaching techniques [53] [31] [46] [29] [37] [33] [10] [54] [36] [24]. Still, no strategy is replicable in different contexts, or their effectiveness is limited to a specific context.

[TC-2] Teaching a practice-oriented course is challenging: The nature of software architecture is practical rather than theoretical. Therefore, to provide a real practical experience, the teaching of architecture activities requires an appropriate context and problems of sufficient complexity, that is, large-scale, with various quality attributes and complex decisions. Hence it isn't easy to provide a learning environment through textbooks, lectures, articles, or individual self-study assignments. An actual architecture includes code and other artifacts that are difficult to be captured in classroom settings [17].

[TC-3] Short course projects are ineffective: Students are typically asked to implement short course projects to explain the consequences of architectural decisions. But such projects do not reveal the wrong decision results, such as choosing the bad pattern or technology, because these decisions have a long-term impact [17].

[TC-4] Lack of learning resources: Open educational resources available to teach a complete course on Software Architecture are scarce. This situation may be because of confidentiality in software organizations, or the size and complexity of software projects are often large, making it challenging to publish. Many projects show only partial descriptions of their architectures, some views, or only partial textual descriptions without a context [17]. Therefore, students do not have real-life examples as a source of learning.

[TC-5] Teaching architecture design in an agile context is even more challenging: Teaching architecture in agile contexts requires exemplifying a context that motivates the need for its design activities even more than in traditional development projects. Although there is literature on approaching architecture design in agile projects, there are only a few guidelines available on how to conduct architecture design in an educational context [1].

[TC-6] Lack of supplementary learning material: Ideally, the architecture design theory should be complemented with comprehensive examples based on practice, including examples of problem situations and solution approaches. In addition, educators must make the value of architectural activities explicit. Fraga and Llorens propose learning architecture based on a set of courses [16]. De Boer et al. suggest student communities build architectural knowledge [13].

[TC-7] Adopting effective teaching methodologies is challenging: Educators need more effort to teach software architecture and participate in collaborative and experiential group activities. Instead of classroom teaching or textbook reading, the learning outcome is better if teachers take on the role of coaches and provide detailed feedback on architectural issues [21].

6. Strategies for conducting training of software architectures

The previous section describes the challenges reported in the literature while teaching courses on Software Architectures consistent with the industry's needs. Researchers have also identified strategies and recommendations to address these challenges. These strategies are:

Adopt Project-based Learning: Working with software projects, especially open-source projects, allows students to confront real complex systems, as well as practice the activities like architectural recovery, designing various architectural views, among others [31, 35, 56].

Define a course hierarchy based on complex problem solving: Researchers have recommended organizing courses in order of levels of complexity involved in problem-solving. These courses may be at introductory, intermediate, and advanced levels. The level of practical problems from low to high levels gradually grows in complexity [16].

Encourage team-based and collaborative learning: Teamwork is essential because it provides scenarios that simulate real environments, exchange ideas, and collaboration. For example, assigning a task to develop an initial architectural design to a team leads to discovering the main architectural concerns and solutions related to a planned product [40].

Teaching software architecture is challenging because it is abstract and best understood when students experience the process of architecture. Additionally, students need to practice technical and social skills to become good software architects. To overcome these teaching challenges, researchers suggest conducting software architecture courses collaboratively. In these types of classes, participants work together to study and document an extensive open-source software system [53].

Provide adequate supplementary material: The courses must have guidebooks to guide students. For example, the books such as *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives* [44] or the book *Software Architecture in Practice third edition* [5] were found quite helpful to learn the practice-oriented aspects of software architecture.

Provide entrepreneurship experience: Recent publications suggest that software engineering students should be exposed to entrepreneurship activities or interdisciplinary teams at technology startups. This approach could motivate students to deliver more realistic products and transform them into budding entrepreneurs. Further, it makes them aware of the agile manifesto for face-to-face communication and external pressure [11].

7. Course Delivery Experiences at our University

The curriculum at our University covers software architecture-related topics under Software Engineering II and Software Engineering Laboratory II courses, which are seventh-semester courses. These courses include fundamental topics such as architectural design, detailed design, software design principles, architectural patterns, design patterns, architecture tactics, architecture evaluation, architecture documentation, among others. The learning objectives of these courses are:

(1) To understand design principles, methods, patterns, and strategies for creating software systems and supporting documents.
(2) To apply criteria for designing a system or a family of software systems, including selecting patterns and analyzing inherent trade-offs.
(3) To document architectural-level design and detailed design through structural and behavioral UML diagrams.
(4) To apply appropriate design patterns and UML notation for the design of a product's components.
(5) To evaluate the quality of software design.
(6) To relate the design properly to software implementation and testing at a practical level.

The skills to be developed for software architects are broad and wide. Hence, we prioritized certain skills and included them while implementing the course on Software Engineering Laboratory II. The course is deployed on a Moodle platform. The course imparts skills such as creation, evaluation, documentation of architecture, decision-making, and communication. These skills are included considered the time available to complete the course and its industry relevance.

In the previous section, we presented various challenges reported in the literature related to teaching a course on Software Architecture. The experiences concerning these challenges of the faculty members at our University who offer software architecture-related courses, i.e., Software Engineering II, are marked in Table 4. We experience most of all the challenges as reported in the literature. In the following section, strategies developed to deal with these challenges are described.

At our University, we developed and implemented strategies to deal with the challenges in Table 4. The second column indicates whether the challenge is presented in our University. The third column describes the strategies to deal with the challenges. These strategies that are either fully or partially implemented are described in this paper.

7.1. *Strategies to deal with Industrial Challenge*

As described in the earlier section, a few challenges originate from software architecture in industries. To deal with these challenges following strategies have been implemented at our University.

(1) *Explicitly document the roles and responsibilities in the software development process. [Strategy for IP-C1]:* This strategy addresses the challenge that the role of software architects in the software development process is not uniformly defined. We have documented the software development process in Business Processing Modelling Notation (BPMN) as shown in Figure 1 that identifies

Id	Challenge description	Our program	Strategies
INDUSTRY PRACTICE RELATED CHALLENGES			
[IP-C1]	The role of software architects in the software development process is vaguely defined	Yes	Explicitly document the roles and responsibilities in the software development process.
[IP-C2]	The software architect's knowledge and skills are unclear.	Yes	Identify software architect's knowledge and skills specific to one organization or a platform (e.g., Amazon's AWC Cloud Architect).
[IP-C3]	In practice, most architectural decisions happen in a team.	Yes	Prescribe multiple roles to teachers.
[IP-C4]	Architecture design in agile context.	Yes	None
[IP-C5]	Architecture decision making needs experience.	Yes	None
[IP-C6]	Architecture design happens in Software Engineering context.	Yes	Reiterate the significance of Software Engineering processes.
INDUSTRY TEACHING CHALLENGES			
[TC-1]	Multiple teaching methodologies.	Yes	Prepare teaching manuals.
[TC-2]	Teaching a practice-oriented course is challenging.	Yes	Adopt project-based learning.
[TC-3]	Short course projects are ineffective.	Yes	None
[TC-4]	Lack of learning resources.	Yes	Develop a repository of projects and learning materials.
[TC-5]	Teaching architecture design in an agile context is more complicated.	Yes	None
[TC-6]	Lack of supplementary learning material.	Yes	Develop a repository of projects and learning materials.
[TC-7]	Adopting effective teaching methodologies is challenging.	Yes	Prepare teaching manuals.

Table 4. Problems or challenges in the teaching of software architectures in our University

various roles, responsibilities, and documents to be exchanged during the development process. The software development process has been illustrated with an example of developing web applications following agile methodology in the context of micro, small and medium-sized enterprises. The curriculum followed at our University has been revised to implement this strategy.

(2) *Gather information from all sources [Strategy for IP-C2]:* This strategy deals with the challenge, which implies that the skills and knowledge required for software architects are not clearly defined. This strategy recommends collecting relevant information from all sources, including existing literature, reports, and documents that formulate curricula in Software Engineering for undergraduate programs such as [22] [25] [51] and find out relevant sets of skills and knowledge. Also, establish contacts with project managers from the local industry to have a general idea of the graduate profile required by companies for software architects.

(3) *Prescribe multiple roles to teachers [Strategy for IP-C3]:* The strategy is meant for handling the challenge that architecture decisions happen in teams. Teaching software architecture requires actively participating in the design process as one of the stakeholders of a project. As a result, a teacher must assume different roles, sometimes as an instructor, user, or manager. To implement this strategy, a special teaching manual needs to be designed that outlines how to "simulate" a multi-role for teachers in developing software engineering activities.

(4) *Reiterate the significance of Software Engineering processes [Strategy for IP-C6]:* Architecture design happens in the Software Engineering context. The strategy to meet this challenge prescribes to develop greater insight among students about various software engineering activities. Students shall not think about architectural decisions in isolation but should work on software projects that simultaneously interweave requirements, architecture, and implementation activities. The course begins with SOLID principles, then proceeds to architectural design, quality attributes, architectural patterns, and tactics.

While designing and delivering a course on Software Architecture at our institute, we de-emphasized the challenges IP-C4 and IP-C5. This decision to de-emphasize IP-C4 is taken because the pre-requisite course on Software Engineering covers conventional software engineering life-cycle methodologies such as waterfall, spiral, and rapid prototyping. In this course, students are exposed to preliminary elements of agile methods. This topic is not covered in-depth as required to explain architecture in the context of agile methodologies. The challenge IP-C5 refers to the ability of decision making accrued over a long period, and a semester long-course typically of 15 weeks falls short of time for this purpose.

7.2. *Strategies to deal with Teaching Challenges*

At our University, we have been addressing the teaching-related challenges in various ways. Some of the initiatives undertaken by the university include training teachers

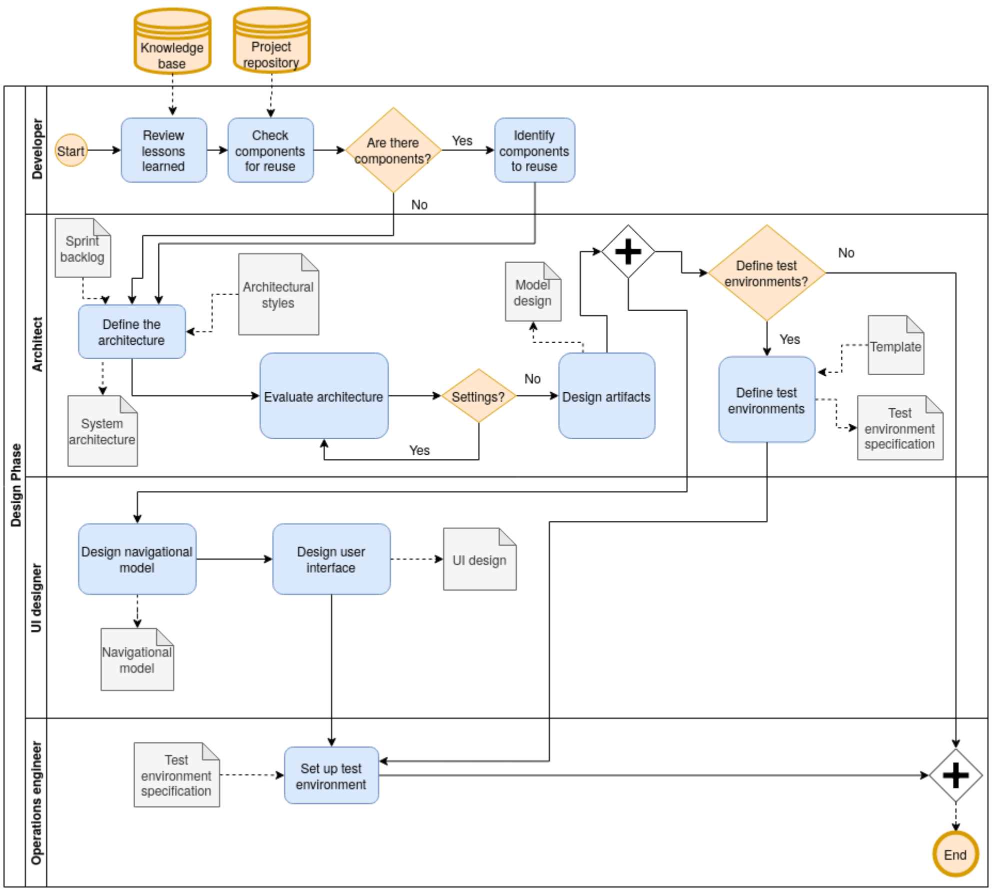
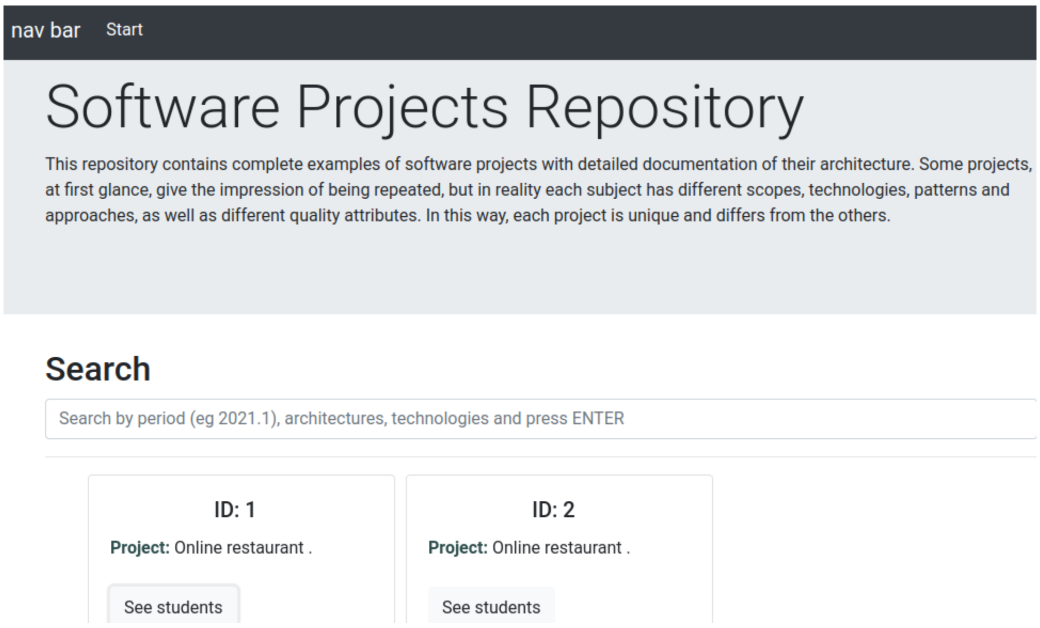


Fig. 1. Development process and the role of the architect

on emerging pedagogies (e.g., Flipped Classroom, Blended Learning, Project-based learning), redesigning curriculum to include emerging topics, and increasing participation of professionals from the industry. Following are some of the strategies we have adopted in recent years.

(1) *Prepare teaching manuals [Strategy for TC-1 and TC7]:* This strategy recommends preparing a detailed teaching manual consisting of well-defined learning objectives, outcomes, and teaching methodologies for every topic included in the curriculum. It aims to overcome the challenge of multiple teaching methodologies. For example, an instructor can teach documenting a software system effectively through flipped classroom technique. In this case, a video explaining views and viewpoints is supplemented with a classroom activity documenting a small-scale software system. Similarly, architectural patterns described in a video followed by a classroom design activity are more effective than simply


nav bar Start

Software Projects Repository

This repository contains complete examples of software projects with detailed documentation of their architecture. Some projects, at first glance, give the impression of being repeated, but in reality each subject has different scopes, technologies, patterns and approaches, as well as different quality attributes. In this way, each project is unique and differs from the others.

Search

Search by period (eg 2021.1), architectures, technologies and press ENTER

ID: 1
Project: Online restaurant .
See students

ID: 2
Project: Online restaurant .
See students

Fig. 2. Repository of projects - link

teaching a topic on the architectural pattern.

(2) *Adopt project-based learning [Strategy for TC-2]*: This strategy recommends project-based learning to overcome the challenge that software architecture is a practice-oriented discipline. Software architecture requires implementing an application of sufficient complexity in an industrial context. Students learn these skills when they are asked to realize a medium-complexity project in a team. The rationale behind this strategy is to provide an environment as close to an organization as possible.

(3) *Develop a repository of projects and learning materials [Strategy for TC-4 and TC6]*: The strategy recommends building a comprehensive online repository of projects, design books, industrial reports, and case studies to overcome the lack of learning resources and supplementary materials. Such a repository guides students for self-preparation and developing their skills. Also, such repositories can have a recommender system that recommends the resources based on student's profiles and learning progress. The initial design of such a repository is shown in Figure 2.

The strategies for teaching challenges TC-3 and TC-5 are not formulated. The challenge TC-3 refers to the *ineffective role* of short course projects, and the challenge TC-5 is not addressed because of the lack of pre-requisite sufficient coverage to the topic of agile methodologies in earlier courses.

Also, we developed a prototype for a platform to teach software architectures

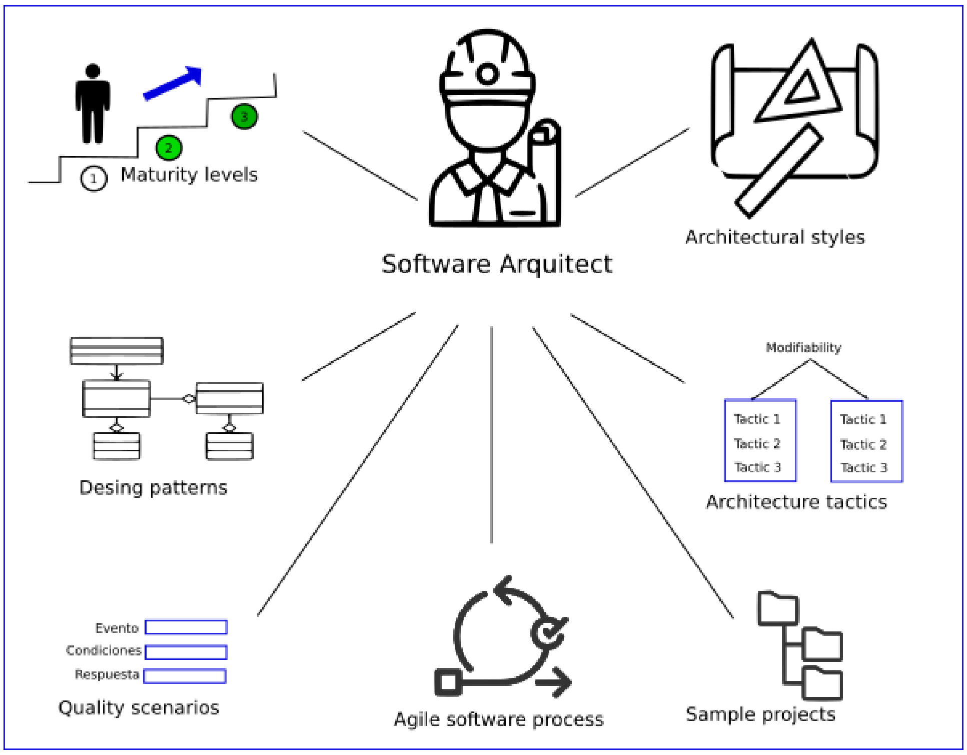


Fig. 3. Software architecture teaching platform.

(see Figure 3). The platform includes the following components: (i) Maturity model of knowledge and skills of software architects, (ii) Catalog of architectural patterns, (iii) Catalog of design patterns, (iv) Examples of architecture tactics, (v) Examples of quality scenarios, (vi) Example projects, (vii) Agile software development process with the role of architect involved.

8. Course Evaluation

We evaluated the course by organizing an architecture katas workshop at the end of the course (https://nealford.com/katas/). We invited four software architects from the industry to evaluate student solutions. In addition, a focus group was set up with the architects to discuss topics related to the skills and knowledge that the industry expects at the level of software architects. Moreover, a survey was performed using the questionnaire shown in Table 5 to evaluate the students' perceived abilities in the kata workshop. We use the Likert scale with the values:

(1) Strongly disagree, (2) Little agree, (3) Agree, (4) Very agree, (5) Strongly agree. The questions with their results are detailed below. Figure 4 shows the analysis of questions with ID from 1 to 8. The Questions with ID from 9 to 11 are

open-ended; they were discussed in the focused group meetings. Some of the comments and recommendations received in the focused group meetings include:

QID	Feedback Question
1	Do you consider the Architectural Katas Workshop a good exercise that satisfactorily contributes to training software architects?
2	Do you consider that the level reached by the students in the architectural katas, in terms of architecture and software design, is by the needs demanded by the software industry?
3	Do you consider that the knowledge and skills around the SOLID design principles, perceived during the students' sustainments, were satisfactory?
4	Do you consider that the knowledge and skills around architectural patterns, perceived during the students' sustainments, were satisfactory?
5	Do you consider that the knowledge and skills around the tactics of architecture, perceived during the sustenance of the students, were satisfactory?
6	Do you consider that the knowledge and skills to recognize and describe the quality attributes of a software project, perceived in the students' solutions, were satisfactory?
7	Do you consider that the knowledge and skills to decide on the appropriate technology to tackle a software project, perceived in the students' solutions, were satisfactory?
8	Do you consider that the knowledge and skills to choose the type of application (traditional web desktop, RIA web, native mobile, hybrid mobile) perceived during the students' solutions were satisfactory?
9	In general terms, how valid for the industry were the solutions proposed by the students?
10	What could methodological and technical elements be added to a meeting of Katas to achieve the objectives of training in architecture?
11	What recommendations can you give us to train and motivate students in architecture and software design issues?

Table 5. The questionnaire used to evaluate the course

(1) Architectural Kata is a novel and exciting exercise. It exposes students to face real-world practices without dealing with the economic risks involved in a business. Architectural katas workshop allows students to put design's theoretical

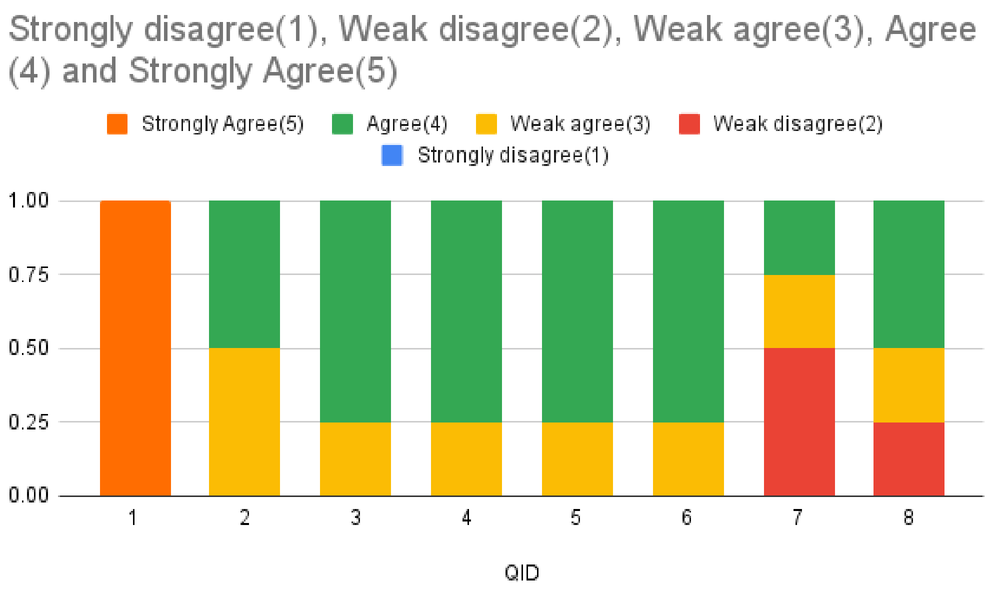


Fig. 4. Course evaluation - Answers to questions 1 to 8

knowledge into practice through real, practical, and exciting challenges.

(2) The students presented industry-acceptable proposals by applying architectural patterns, quality attributes, and design patterns. However, some of the proposals missed frequently needed issues such as load balancing, the configuration of web servers, authorization servers, among other essential elements for a solution based on microservices.

(3) The Kata Workshop should allow students to think better about its response and prepare a presentation without time pressure while solving the exercises.

(4) The Katas Workshop should consider constraints, such as project funding, resources, and the technologies mastered by the development team. Further, the workshop should involve topics relevant to the software industry, such as cloud application deployment and DevOps.

(5) The topics related to technology are very dynamic and change at a rapid rate. Often, teachers attempt to train students with diverse technologies. But students should be trained on core skills and fundamentals to learn new technologies and adopt them smoothly.

8.1. *Course Evaluation By The Students*

We applied a survey to measure the time students need to get their first job in the software industry to two groups of students: before and after using the software architecture training strategies. Twenty-six persons answered the survey after applying the training strategies and thirty-six before. The survey questions were:

(1) How many semesters elapsed after you took the Software Engineering II course and got your first job in the software industry? Figure 5 shows the results of responses.
(2) Did the skills acquired in the software engineering II course regarding software architecture and design issues allow me to enter the world of software development satisfactorily? We use the Likert scale with the values: (1) Strongly disagree, (2) Little agree, (3) Agree, (4) Very agree, (5) Strongly agree. Figure 6 shows the results of responses.

Figure 6 shows that most students who received the training strategies in software architecture still do not get their first job in the industry; this is understandable because they have not yet finished their degree. However, the strategies began to have an effect since two students got a job two semesters after seeing the course, and one got a job three semesters later. We must consider that this experience was carried out during the Covid-19 pandemic, which could affect employability in the country and the world.

Figure 6 shows the high satisfaction of the students who received the strategies: 38% very agree (ten persons), and 53.8% (14 persons) strongly agree with the skills acquired in the course.

9. Conclusions and Future Work

Based on the analysis of the literature dealing with educational aspects of software architecture and our experience of teaching software architecture course at our University, we observe that:
(i) A course on Software Architecture is becoming an essential part of the curriculum for graduates programs in Computer Science and System Engineering courses. This course emphasizes high-level design aspects to assure the quality of software products and services expected in industries.
(ii) However, training graduates on this emerging and highly industry-relevant skill is challenging. Some of these originate from how software architecture is practised in industries and how that are taught in an academic institute.
(iii) Adoption of proper teaching interventions and strategies can overcome most of those challenges making a smoother transition of graduates from universities to industries.
(iv) Software architecture mainly impacts projects having a lifespan with medium and long-term duration, making it difficult to measure the impact of architecture design for a semester-long project. Therefore, it is essential to work with medium or high complexity systems, partially developed projects, with various quality attributes, collaborative work, and particular support from the teacher with different roles.

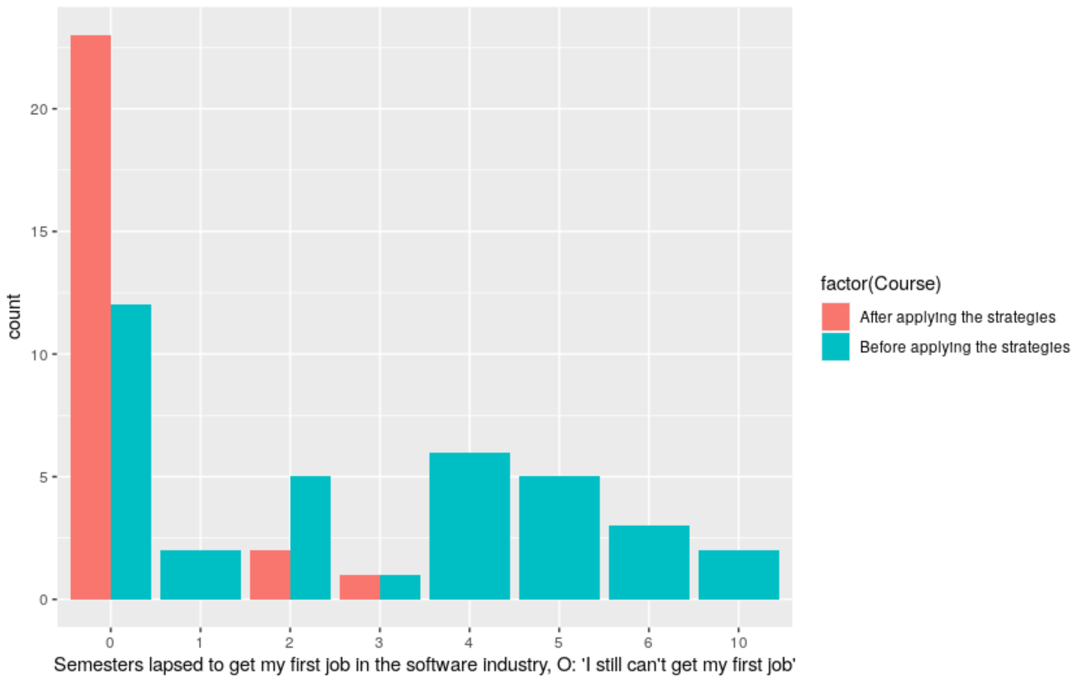


Fig. 5. Course evaluation - the time students need to get their first job in the software industry

(v) Students can be exposed to some of the recurring concepts such as architecture patterns, styles, and quality attributes by preparing teaching manuals, project-based learning, and developing repositories of projects. These fundamental concepts are hidden within a large number of secondary ideas. Therefore, we must consider strategies that organize architectural knowledge so that development activities can be structured to facilitate incremental learning. However, the ability of architecture decisions takes time to develop, and it accrues through multiple project experiences.
(vi) Architecture practices require a context that motivates the need for their use, which involves the different software processes models. In people-centric software engineering processes such as agile methodologies, architecture modeling is challenging because it de-emphasizes the role of analysis, design, and other processes over people's roles.
(vii) As a result of this software architecture teaching and evaluation exercise, there are aspects to improve. First,it will be interesting to explore further adoption of other strategies reported in the related works such as project-based learning. Second, we can evaluate a strategy that gradually exposes students to increased complexity of software system in an incremental manner. The goal is for students to be able to adapt and evolve architecture, simulating what happens in real projects. Third, extending the course-content with anti-patterns to train architects. Since it

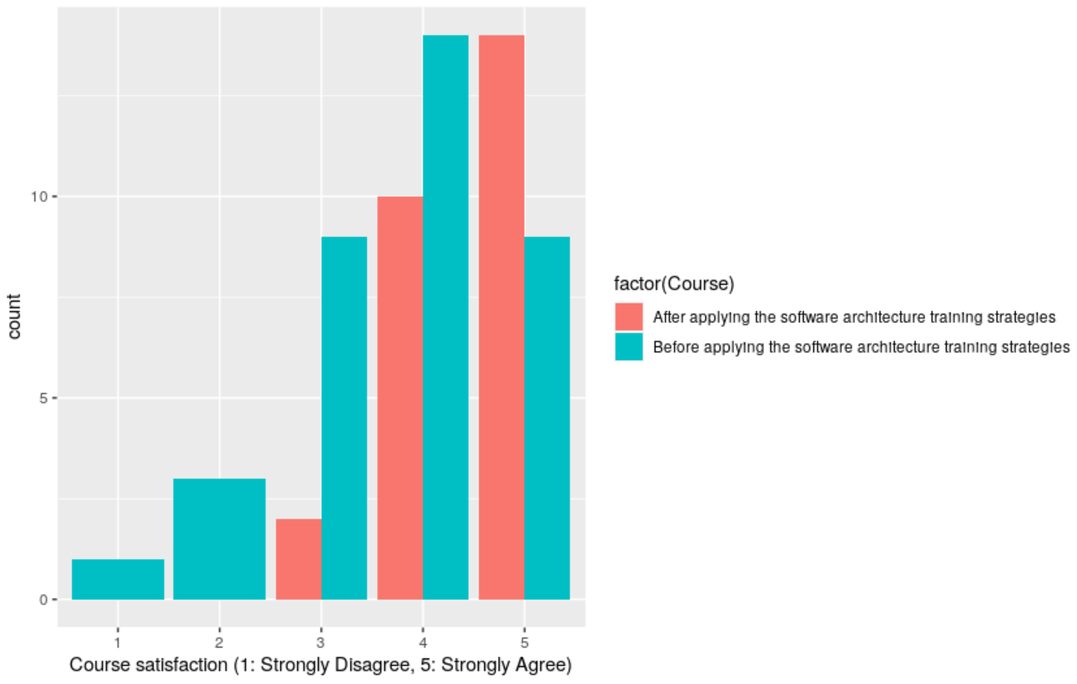


Fig. 6. Course evaluation - Student satisfaction with the skills acquired in the course

is equally important to know what works and what does not work. (viii) Further, we found that the Architecture Katas Workshop effectively increases student engagement during learning and assessment. The work presented in this paper can be extended further by developing a comprehensive learning platform around the prototype suggested in Figure 3.

References

[1] Tubagus Akhriza, Yinghua Ma, and Jianhua Li. Revealing the Gap Between Skills of Students and the Evolving Skills Required by the Industry of Information and Communication Technology. *International Journal of Software Engineering and Knowledge Engineering*, 27:675–698, 2017.

[2] S Angelov and P de Beer. Designing and Applying an Approach to Software Architecting in Agile Projects in Education. *Journal of Systems and Software*, 127(C):78–90, 2017.

[3] Apiumhub. The role of a software architect. `https://apiumhub.com/tech-blog-barcelona/role-of-a-software-architect/`, 2018. Accecced: 01-08-202.

[4] Lugo Manuel Barbosa Guerrero. Arquitectura De Software Como Eje Temático De Investigación. *Avances Investigación en ingeniería*, 1(04):78–85, sep 2006.

[5] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice, third Edition*. Pearson Education, Massachusetts, USA, 2012.

[6] Betterteam. Software Architect. `https://www.betterteam.com/software-architect-job-description`, 2020. Accecced: 01-08-202.

[7] Pierre Bourque and Richard Fairley. Guide to the Software Engineering Body of Knowledge - SWEBOK Guide. *IEEE Computer society*, page 200, 2004.

[8] Rafael Capilla, Olaf Zimmermann, Carlos Carrillo, and Hernán Astudillo. Teaching Students Software Architecture Decision Making. In Anton Jansen, Ivano Malavolta, Henry Muccini, Ipek Ozkaya, and Olaf Zimmermann, editors, *Software Architecture*, pages 231–246, Cham, 2020. Springer International Publishing.

[9] Lenna Carballo Muñoz and Ivette Núñez Barrientos. Propuesta para evaluar arquitecturas de software. *Revista Universidad y Ciencia*, 7(3):159–174, aug 2018.

[10] Paolo Ciancarini, Stefano Russo, and Vincenzo Sabbatino. A Course on Software Architecture for Defense Applications. In Paolo Ciancarini, Alberto Sillitti, Giancarlo Succi, and Angelo Messina, editors, *Proceedings of 4th International Conference in Software Engineering for Defence Applications*, pages 321–330, Cham, 2016. Springer International Publishing.

[11] Orges Cico and Letizia Jaccheri. Industry trends in software engineering education: A systematic mapping study. In *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019*, pages 292–293, Montreal Quebec Canada, may 2019. IEEE.

[12] Paul Clements and Len Bass. Using business goals to inform software architecture. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, RE2010*, pages 69–78, Sydney, NSW, 2010. IEEE, IEEE Computer Society.

[13] Remco C De Boer, Rik Farenhorst, and Hans van Vliet. A Community of Learners Approach to Software Architecture Education. In *Proceedings 22nd Conference on Software Engineering Education and Training*, pages 190–197, USA, 2009. IEEE Computer Society.

[14] Cesar De la Torre Llorente, Unai Zorrilla Castro, Javier Calvarro Nelson, and Miguel Angel Ramos. *Guia de arquitectura de N capas orientada al dominio con .Net 4.0.* Krassis Press, Madrid, Spain, 2010.

[15] Liliana Dobrica and Eila Niemela. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[16] Anabel Fraga and Juan Llorens. The Challenge of Training New Architects: an Ontological and Reinforcement-Learning Methodology. *Journal of Software*, 2:24–28, 2007.

[17] Matthias Galster and Samuil Angelov. What makes teaching software architecture difficult? In *Proceedings - International Conference on Software Engineering*, pages 356–359, Austin Texas, 2016. Association for Computing MachineryNew YorkNYUnited States.

[18] Vahid Garousi, Görkem Giray, Eray Tüzün, Cagatay Catal, and Michael Felderer. Closing the gap between software engineering education and industrial needs. *IEEE Software*, 37:68–77, 2020.

[19] Kevin Gary. Project-Based Learning. *Computer*, 48(9):98–100, sep 2015.

[20] Neil B Harrison and Paris Avgeriou. How do architecture patterns and tactics interact? A model and annotation. *Journal of Systems and Software*, 83(10):1735–1758, 2010.

[21] Orit Hazzan. The reflective practitioner perspective in software engineering education. *Journal of Systems and Software*, 63:161–171, 2002.

[22] Thomas Hilburn and Donald Bagert. A software engineering curriculum model. In *FIE'99 Frontiers in Education. 29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings*

(IEEE Cat. No.99CH37011, volume 1, pages 12A4/6 – 12A411 vol.1, San Juan, Puerto Rico, 1999. IEEE Computer Society.

[23] IEEE. Ieee 1471-2000 - ieee recommended practice for architectural description for software-intensive systems. Technical report, 2000. `https://ieeexplore.ieee.org/document/875998`.

[24] Zhenyan Ji and Jing Song. Improved Teaching Model for Software Architecture Course. In *Proceedings of the 2015 International Conference on Education, Management, Information and Medicine*, pages 333–338, No City, 2015. Atlantis Press.

[25] Joint Task Force on Computing Curricula. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM and IEEE, USA, 2013.

[26] Lina Khalid. *Software Architecture for Business*. Springer, 2020.

[27] Barbara Kitchenham, O Pearl Brereton, David Budgen, Mark Turner, John Bailey, and Stephen Linkman. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15, 2009.

[28] Arvind W Kiwelekar and Hansaraj S Wankhede. Learning objectives for a course on software architecture. In *European Conference on Software Architecture*, pages 169–180. Springer, 2015.

[29] Christian Köppe and Leo Pruijt. Teaching Software Architecture Concepts with HUSACCT. In *Conference: SPLASH-EAt: Pittsburgh, USA*, page 4, Pittsburgh, USA, 2015. University of Applied Sciences Utrecht.

[30] Philippe Kruchten. What do software architects really do? *Journal of Systems and Software*, 81(12):2413–2416, 2008.

[31] Patricia Lago and Hans Van Vliet. Teaching a Course on Software Architecture. In *18th Conference on Software Engineering Education Training (CSEETÓ5)*, pages 35–42, Ottawa, ON, Canada, 2005. IEEE.

[32] Paul M Leidig and Lillian Cassel. Acm taskforce efforts on computing competencies for undergraduate data science curricula. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 519–520, 2020.

[33] Weigang Li. 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)*, volume 352, pages 17–21, Wuhan, China, 2019. Atlantis Press.

[34] X. Li, T. Xu, and C. Zhang. Discussion about an undergraduate course on software architecture. In *2008 International Conference on Computer Science and Software Engineering*, volume 5, pages 616–619, 730 Massachusetts Ave., NW Washington, DCUnited States, 2008. IEEE Computer Society.

[35] Zheng Li. 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*, pages 1–11, Seoul South Korea, 2020. Association for Computing Machinery.

[36] Ouh Eng Lieh and Yunghans Irawan. Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, San Jose, CA, USA, 2018. IEEE.

[37] Ouh Eng Lieh and Yunghans Irawan. Teaching adult learners on software architecture design skills. In *Proceedings - Frontiers in Education Conference, FIE*, volume 2018-Octob, pages 1–9, Uppsala, Sweden, 2019. IEEE.

[38] Eng Lieh Ouh, Benjamin Kok Siew Gan, and Yunghans Irawan. Did our Course

Design on Software Architecture meet our Student's Learning Expectations? In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Uppsala, Sweden, 2020. IEEE.

[39] Lars Lundberg, Jan Bosch, Daniel Häggander, and Per-Olof Bengtsson. Quality attributes in software architecture design. In *Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications*, pages 353–362. Citeseer, 1999.

[40] T Mannisto, J Savolainen, and V Myllarniemi. Teaching Software Architecture Design. In *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pages 117–124, Vancouver, BC, Canada, 2008. ACM.

[41] Elkin Moreno Vélez. Arquisoft90 Formación profesional y capacitación. `https://www.linkedin.com/showcase/arquisoft90-entrenamiento-den-arquitectura-de-software`, 2020. Accecced: 01-06-2020.

[42] Dewayne E Perry and Alexander L Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*, 17(4):40–52, 1992.

[43] Mark Richards and Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach 1st Edicion.* O'Reilly Media, Inc., Canada, 2020.

[44] Nick Rozanski and Eoin Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives.* Addison-Wesley, New Jersey, USA, 2012.

[45] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2008.

[46] Chandan R. Rupakheti and Stephen V. Chenoweth. Teaching Software Architecture to Undergraduate Students: An Experience Report. In *Proceedings - International Conference on Software Engineering*, volume 2, pages 445–454, Florence Italy, 2015. IEEE Press.

[47] Ahmed E Sabry. Decision model for software architectural tactics selection based on quality attributes requirements. *Procedia Computer Science*, 65:422–431, 2015.

[48] Muhammad Arif Shah, Iftikhar Ahmed, and Muhammad Shafi. Role of Software Architect: A Pakistani Software Industry Perspective. *Research Journal of Recent Sciences*, 3:48–52, 2014.

[49] Sofia Sherman and Naomi Unkelos-Shpigel. What do software architects think they (should) do? Research in progress. In *Advanced Information Systems Engineering Workshops*, volume 178, pages 219–225, Cham, 2014. Springer International Publishing.

[50] SyndicodeTeam.
The role, skills, and duties of a software architect. `https://syndicode.com/blog/the-role-skills-and-duties-of-a-software-architect/`, 2017. Accecced: 01-08-2020.

[51] The Joint Task Force on Computing. Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Technical report, ACM and IEEE, New York, NY, USA, 2004.

[52] Arie Van Deursen, Mauricio Aniche, Joop Aué, Rogier Slag, Michael De Jong, Alex Nederlof, and Eric Bouwers. A Collaborative Approach to Teaching Software Architecture. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '17, pages 591–596, New York, NY, USA, 2017. Association for Computing Machinery.

[53] Arie Van Deursen, Maurício Aniche, Joop Aué, Rogier Slag, Michael De Jong, Alex Nederlof, and Eric Bouwers. A Collaborative approach to teaching software architecture. In *Proceedings of the Conference on Integrating Technology into Computer*

Science Education, ITiCSE, pages 591–596, Seattle Washington USA, 2017. ACM.

[54] Bingyang Wei, Yihao Li, Lin Deng, and Nicholas Visalli. *Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application*, volume 845, pages 43–54. Springer International Publishing, Cham, 2020.

[55] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, New York, NY, USA, 2014. Association for Computing Machinery.

[56] Li Zhang, Yanxu Li, and Ning Ge. Exploration on theoretical and practical projects of software architecture course. In *15th International Conference on Computer Science and Education, ICCSE 2020*, number 61732019, pages 391–395, Delft, Netherlands, 2020. IEEE.