Towards a Software Architecture Training Pattern Language

WILSON LIBARDO PANTOJA YEPEZ, Universidad del Cauca
JULIO ARIEL HURTADO, Universidad del Cauca
LUIS MARIANO BIBBO, Universidad Nacional de la Plata
ALEJANDRO FERNANDEZ, Universidad Nacional de la Plata
BANDI AJAY, Northwest Missouri State University

Purpose: This article provides a Software Architecture (SA) training pattern that allows professors and trainers to design and execute courses at the undergraduate level that develop students' competencies according to the software industry's expectations.
Methods: The training patterns were extracted from a literature review based on reports of SA course experiences. In this review, we looked for recurrent challenges in SA teaching and the solutions found and experienced by professors. The first training pattern was socialized and refined through a focus group with professors and researchers with expertise in patterns and software architecture.
Results: We propose seven training patterns that could help professors create and improve SA courses by developing competencies close to industry needs.
Conclusion: A SA course aligned with industry needs is essential in computer science, systems engineering, and related programs curricula. However, training undergraduate students with the skills demanded by industry has many challenges. To design and execute an SA course, we propose seven training patterns that could facilitate the achievement of fundamental competencies of the undergraduate student in the creation and documentation of SA.

Categories and Subject Descriptors: D.2.11 [**Software Engineering**]: Software Architectures—*Training*

General Terms: Training

Additional Key Words and Phrases: Catalog, training patterns, software architecture

ACM Reference Format:

Pantoja, W. and Hurtado, J. and Bibbo L. 2023. Pattern of training in software architecture. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 22 (October 2023), 33 pages.

1. INTRODUCTION

Software architecture is defined as a structure or structures of a system, which organize software elements and their relationships [Clements and Bass 2010]. The emphasis is on the externally visible properties of the pieces of software and how they are related. Software Architecture provides a mechanism for establishing, documenting, and communicating a system's main design decisions. Some of the recurring architectural concepts in industrial practices and courses on software architecture are quality attributes, architectural tactics, architectural patterns, and architectural decisions.

Quality attributes are the characteristics that a software product shall satisfy. Each quality attribute is associated with specific metrics defining the quality levels for a software product [Sabry 2015]. A few examples of quality attributes include security, modifiability, reliability, performance, interoperability, and others. *Architecture Patterns* are common solution structures to a similar design problem that are well understood and documented [Harrison and Avgeriou 2010]. Examples of Architecture patterns include Layers, Pipelines, Event-driven, Microkernel,

and Microservices. 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. *Architectural tactics* are high-level abstractions that capture decisions 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 [Harrison and Avgeriou 2010]. *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 the justification or rationale behind the selection. The choices may be about the structure of the application or system, the selection of a technology to implement the design or a tradeoff between quality attributes.

Software Architecture (SA), encompassing software design, documentation, and evaluation, is receiving growing attention in industries [Li 2019]. The focus on quality attributes during software system development has contributed to this trend. Recognizing the importance of SA in the industry, several academic institutions have incorporated a SA course into their undergraduate programs in Software Engineering [Rupakheti and Chenoweth 2015] [Li 2020] [Chatley and Field 2017] [Zhang et al. 2020] [Wang 2011] [Wu and Wang 2012] [Wedemann 2018] [Angelov and de Beer 2017] [de Beer and Angelov 2015]. Nevertheless, instructors of this course encounter various challenges. These difficulties arise from applying SA in real-world settings and teaching these concepts in an academic environment [Rupakheti and Chenoweth 2015] [Zhang et al. 2020].

This article presents seven Software Architecture education patterns that allow professors and instructors to design and execute undergraduate-level courses that develop students' competencies according to the software industry's expectations. In addition, these training patterns could have a larger target population, as these could be used to train software architects in industries.

A SA course that meets industry needs is essential in the curriculum of computer science and related programs [Li 2020]. However, training undergraduate students with the skills demanded by the industry poses many challenges [Rupakheti and Chenoweth 2015]. Some of these challenges include: the abstract nature of architectures, the foundational knowledge required by students, the difficulty in recreating projects and environments in the classroom with characteristics similar to those of the industry, team collaboration difficulties, lack of updated resources and content, lack of experience of professors in real projects, among others [Rupakheti and Chenoweth 2015] [Zhang et al. 2020] [Lieh and Irawan 2018] [Angelov and de Beer 2017] [Van Deursen et al. 2017] [Lieh and Irawan 2019]. To design and execute a SA course, we propose a first education pattern (from a catalog of seven patterns) that could help instructors create and improve SA courses by developing competencies close to the needs of the industry.

The presented education pattern in this paper was extracted from the literature review "Training Software Architects Suiting Software Industry Needs: A Literature Review"' [Pantoja et al. 2023a] based on reports of SA course experiences. In this review, we analyzed 56 articles reporting on teaching experiences focused specifically on software architectures or focused on software engineering in general but discussing SA. The main contributions of this work include identifying strategies used in educating SA students aligned with the needs of the software industry. These strategies include short design projects, large development projects, and projects with actual clients.

Inspired by the goals of the GoF design patterns [Gamma et al. 1994], we defined the following objectives for education patterns:

—Provide catalogs of reusable elements in the design of SA courses.

—Avoid repetition in searching for solutions to previously known and solved problems.

—Formalize a common vocabulary among SA course designers.

—Standardize the way course designs are made.

—Facilitate learning of new SA course proposals by condensing existing knowledge.

To arrive at the training patterns, we followed the following steps:

(1) We conducted a systematic mapping to search for SA education experiences.
(2) We defined the form to specify the patterns.
(3) We performed the extraction of the patterns.
(4) We preliminarily validated the pattern with academic experts using the focus group technique.

As a result, we obtained seven training patterns that can be seen in Table I.

Table I. Training patterns found

No	Pattern name	Summary
1	Mini-Projects-based training	When students reach their first SA course, they have already seen programming and software development courses. At this level, students have skills in creating simple applications but working only with functional requirements. To develop a SA, the professional must consider quality attributes (scalability, performance, security, etc.) and satisfy functional requirements. Working with mini projects, students can practice architecture patterns and architecture tactics to favor the fulfillment of quality attributes.
2	Large project-based training	When students have acquired sufficient software development skills to attack medium to large development projects in complex domains, the professor could develop a course that connects the theoretical foundations of software architecture with practice through a large project of some complexity. Students learn SA through a complete real project where they can see the results of their architectural design as a product.
3	Open-source projects-based training	When recent graduates join the software industry, one of the initial challenges they face is developing software components on existing and usually large projects. Students work with an open-source project to have the unique opportunity to learn attitudes only present in real-world scenarios, which can increase their skills and confidence.
4	In-house project-based training	Despite the advantages of confronting students with modifying real systems, working with open-source projects can become too complex for students and professors. Therefore, an alternative is for professors to have their open-source application (In-house project) to teach SA with enough complexity of an industrial system
5	Cases-based training	Making correct architectural decisions while constructing a software system is one of the significant skills a future software architect must develop. Case-based training focuses on designing and analyzing real software projects from companies. Students experience and use the theory and technology of SA design applied to specific projects to improve the teaching effect. Cases are usually taught through the flipped classroom.
6	Problem-solving-based training	Making architecture decisions as a team while constructing a new software system is one of the most critical skills a future software architect must develop. The most important decisions are made at the beginning of the creation of an application: technologies to be used, architecture patterns, and trade-offs between quality attributes, among others. After comes the development and maintenance of the application. It is an activity where the goal is to design the architecture of a system close to reality. The activity is usually done in teams, where each group is assigned an exercise to be solved in a given time.
7	Games-based training	Teaching SA is complex because the architect's role is multifaceted. The architect requires developing technical, analytical, and communication skills. Most talented architects have acquired extensive knowledge over many years of experience. Professors need fun teaching methods for shortened training time related to SA decision-making. Game-based training is an educational strategy that uses elements of games to foster student engagement, participation, and learning.

The structure of this paper is as follows: Section 2 analyses the context of SA Teaching Pattern Language. Section 3 describes the catalog of training patterns. Section 4 describes a preliminary validation using a focus group. Finally, section 5 wraps up with conclusions and ideas for future work.

2. THE CONTEXT OF SOFTWARE ARCHITECTURE TEACHING PATTERN LANGUAGE

We search and compare different ways of specifying patterns in areas similar to our object of study. Specifically, we rely on the works of "Gang of Four" (GoF) [Gamma et al. 1994], "Pattern-Oriented Software Architecture (POSA)" [Buschmann et al. 1996] and "Towards a Pattern Language for Learning Management Systems"[1] [Avgeriou et al. 2003]. Considering the templates proposed by these authors and the mandatory fields suggested by Meszaros et al. in "A pattern language for pattern writing" [Meszaros and Doble 1998], we offer a structure to describe the training patterns. We take the fields that fit our training patterns; some are new and do not come from the world of software patterns. The format we chose to document our training pattern is as follows:

—**Name**: It is a short phrase describing the pattern's name.

—**Problematic context**: It describes the problem situation or circumstance in which the pattern is applied.

—**Forces**: They represent a concrete scenario that motivates the use of the pattern.

—**Solution**: It describes the solution to the problem by applying the pattern.

—**Prerequisites**: They are prerequisites for students to apply the pattern.

—**Example**: It is an example of how to use the pattern.

—**Competencies**: List of competencies that students develop by applying the pattern.

—**Variants**: They describe alternative cases or situations on how to apply the pattern.

—**Advantages**: Positive consequences of applying the pattern.

—**Disadvantages**: Negative consequences of applying the pattern.

—**Reported experiences using the pattern**: References of papers that applied the pattern in some SA training experience.

—**Related patterns**: indicates the connections and relationships between different training patterns. These relationships help designers and developers understand how different patterns can be combined and applied together.

Alexander et al. [Alexander 1977] define a pattern language as "a collection of related patterns that captures the entire design process and can guide the designer through step-by-step design guidelines." Following this concept, we have proposed a Software Architecture Pattern Language to denote a set of related patterns that collaborate within the boundaries of SA course design.

Figure 1 shows graphically the proposed language where the rectangles show each of the seven training patterns, and the arc shows the relationship between them. The relationship *frequently used* means that one training pattern can be used with another in the same AS course. This way, a professor can choose several training patterns for their course. Each pattern will allow the development of a set of AS skills. The training patterns are divided into two distinct groups. The first group corresponds to those that extend from a project-development-driven pattern. The second group is those patterns that extend from a decision-making-driven pattern. There are mutually exclusive training patterns, which implies that, due to the effort required for their implementation by both the professor and the students, it is not possible to apply more than one simultaneously. This restriction is the case of project-driven patterns. On the other hand, the training patterns of the decision-making hierarchy do not demand excessive effort for their implementation, thus allowing their combination with any different training pattern. This

[1] This paper presents an approach of recording design experience in the form of design patterns for Learning Management Systems and aims at developing a pattern language for these systems.

second group of patterns can be easily integrated into a single class session. In this way, a professor can apply, for example, the “large project-based training” pattern and other patterns such as “problem-solving training,” “cases and flipped classroom training,” and “games-based training.”

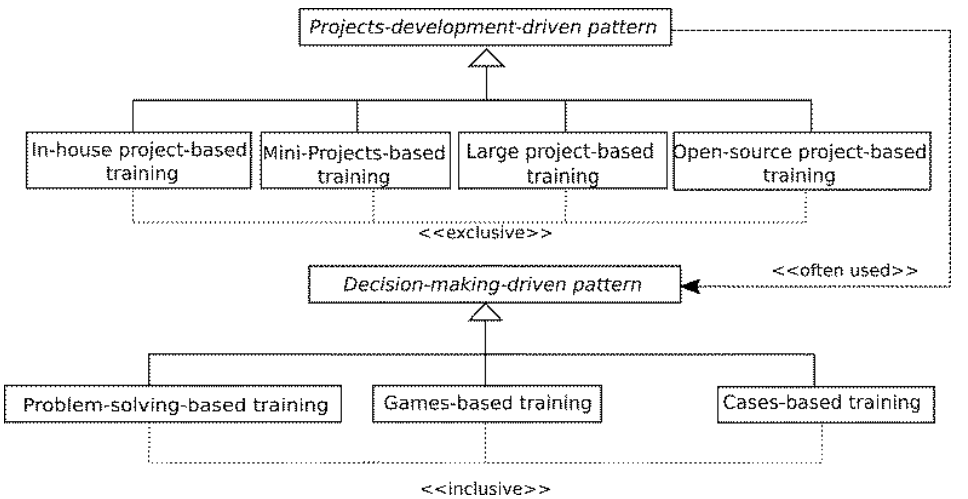

Fig. 1. Software Architecture Teaching Pattern Language

3. PATTERNS CATALOG

In this section, we describe the seven training patterns.

3.1 Training pattern 1

Name: Mini-Projects-based training.

Problematic context: By the time students reach their first SA course, they have already seen programming and software development courses. At this level, students have skills in creating simple applications but working only with functional requirements. To develop a SA, the professional must consider quality attributes (scalability, performance, security, etc.) and satisfy functional requirements. The software industry then expects software architects to design the structure of applications to satisfy the system’s quality attributes and constraints [Bass et al. 2012]. According to Sherman [Sherman and Unkelos-Shpigel 2014], the architect is responsible for the design and technical decisions in the software development process and solves a problem by defining the structures of a system that can be implemented using certain technologies. However, finding the right balance between software quality attributes such as security, performance, usability, availability, maintainability, and interoperability, among others, takes a lot of work. These attributes can conflict with each other, so the software architect needs to know tactics, patterns, and principles that help them make the right decisions [Lieh and Irawan 2018].

At the same time, software projects are becoming more and more demanding. Systems are required to be easy to use, providing a good user experience; to work on different devices, including mobile devices; to be secure and maintain privacy; to be able to integrate with other systems and facilitate interoperability; to be able to process large volumes of data; and so on. All of this means that courses related to SA need to provide more life-like experiences for students to prepare them for these demands. Teaching SA to work with real-world-like projects means that students must learn about and apply new topics such as quality attributes, architecture styles, and tactics.

From this perspective, students do not yet have sufficient software development skills to cope with demanding development projects in complex domains [Chatley and Field 2017]. Students need to add experience to adapt to situations that may arise in current software industry projects.

On the other hand, the professor seeks to develop a SA course, hoping to connect the student with the theoretical and practical fundamentals of SA. The objective is to enable students to gain confidence in the area without dealing with the complexity of an extensive system. The most important skills to develop will be identifying and specifying the software system's quality attributes, choosing architecture styles that favor those quality attributes, and diagramming the architecture. Students are expected to develop a substantial architectural design experience considering the students' time and resource constraints and the universities' infrastructure.

Forces:

—By applying architectural styles and tactics, the professor wants to train students to achieve software quality attributes (such as scalability, performance, and security).

—The industry expects students to know how to promote specific quality attributes by applying architectural styles.

—The professor wants to develop a practical course developing students' skills in building a new software system with good architecture searching for a balance between breadth and depth of knowledge.

—Students do not have experience working on complex development projects. On the contrary, through mini projects, they can learn how to design the architecture of a system and gain experience to face future more extensive and more complex design projects.

Solution: The professor works his course with mini projects so that students can practice the concepts of architectural patterns and architectural tactics to favor the fulfillment of quality attributes [Rupakheti and Chenoweth 2015; Li 2020].

Students develop in teams one or several short projects of one or two weeks each. Each mini-project mainly favors one quality attribute(s). A short project involves developing a project with few (two or three) functional requirements.

For example, do a mini-project that favors modifiability by applying design principles and a layered architecture; then develop a mini-project that favors application performance, another that facilitates scalability, and another that prioritizes security. The projects should be simple (In a domain understandable to students), seeking that the student's effort is in understanding the quality attributes, tactics, and architectonics involved in the solution. [Li 2020].

We recommend that the projects and examples be developed in technologies that students and professors are familiar with from their previous courses, java, others in C, C++, Python, etc. The professor can have examples of projects for each quality attribute, which can be consulted by the students in a repository of examples.

By the end of the semester, students will have developed several projects, each favoring a quality attribute. Figure 2 provides the main elements of the pattern solution.

Each mini-project can be described in a document with the following structure:

(1) Introduction or context of the software system to be developed.
(2) Learning goals to be developed in the students (or competencies).
(3) Description of functional requirements.
(4) Description of the non-functional requirement.
(5) Definition of deliverables.
(6) Project evaluation rubric.

Student prerequisites:

Mandatory:

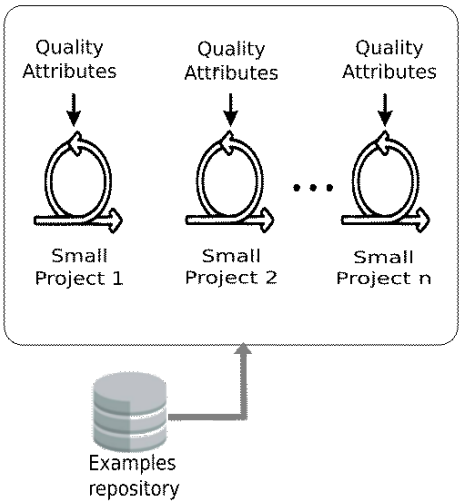

Fig. 2. Mini-Projects-based training pattern

—Object Oriented Programming.

—Database design.

Desirable:

—Operating systems.

—Distributed Systems.

—Software Engineering.

Example of pattern use: The professor can choose the number of mini-projects for his course; it can be between two and six (depending on the duration of the course in weeks). The first mini-project can be approached with a monolithic structure with a layered architecture that favors the quality attribute of modifiability. Students must apply a layered design using principles and some software design patterns. The professor can also provide them with an example project so the students can build on a reference.

In the second mini-project, the professor can work with a microkernel architecture to favor the quality attributes of extensibility and modularity. The microkernel architectural pattern is also known as the plugin architectural pattern. It is generally used when software teams create systems with interchangeable components, and it applies to software systems that must be able to adapt to changing system requirements. It separates a minimal functional core from extended and customer-specific parts. The architecture also serves as a plug to connect these extensions and coordinate their collaboration.

The professor can work with a client/server architecture in the third mini-project. The professor can implement a single server serving simultaneous requests from several clients. In this case, the attribute of performance can be measured.

The professor can work on the event-driven architecture in the fourth mini-project with a broker topology that favors performance, scalability, and fault tolerance quality attributes. This topology is proper when you have a relatively simple event processing flow and do not need central event orchestration and coordination. As a lightweight message broker technology, technologies such as RabbitMQ, ActiveMQ, HornetQ, etc., can be used.

In the fifth mini-project, the professor can work with a microservices architecture that favors the attributes of scalability and elasticity because the monolithic application is divided into small independent applications that

communicate with each other. We recommend working with a framework specialized in microservices to make the implementation easier. In the case of Java, frameworks such as Spring Boot, MicroProfile, WildFly Thorntail, Cricket, etc., can be used.
Competencies (or learning outcomes) addressed: With this training pattern, the following competencies are developed (See Appendix 2):
Mandatory competencies: C2, C5, C8, C12, C18, C23.
Optional competencies: C3, C4, C9, C14, C21.

Variants:
One approach to SA training is to start with a large project (with many requirements), address a small set of high-priority requirements for the client in each mini-project, and change the quality attributes. In this way, each mini-project will require redesign and involve new architectural styles.

Advantages:

—Students will have the opportunity to experiment with various styles of architecture and see their applicability through mini development projects.

Disadvantages:

—Students do not participate in an entirely real project with genuine clients.
—This pattern requires the professors to have example projects in source code to avoid students spending too much time-solving implementation issues from scratch.

Reported experiences with the use of the formation pattern:

—Teaching Software Architecture to Undergraduate Students: An Experience Report [Rupakheti and Chenoweth 2015].
—Using Public and Free Platform-as-a-Service (PaaS) based Lightweight Projects for Software Architecture Education [Li 2020].
—Lean learning - Applying lean techniques to improve software engineering education [Chatley and Field 2017]

Related patterns:
Mini-Projects-based training pattern can be combined in an SA course with other training patterns that involve little student effort and allow the development of various decision-making competencies. For example, Case-based training, Problem-solving-based training, and Games-Based Training.

3.2 Training pattern 2

Name: Large Project-Based training.

Problematic context: Students have sufficient skills in software development to face medium to large development projects and in complex domains [Chatley and Field 2017]. Therefore, students are at an intermediate level of training in SA topics.
In this way, the professor intends to develop a SA course to connect the students with the theoretical and practical fundamentals. This situation will allow them to gain confidence in the area through a large project of sufficient complexity that solves an enterprise problem. It is important to emphasize that the professor must have experience and preparation in dealing with software projects of greater complexity to guide his students.

Forces:

—The professor wants to train students in software architecture through a complete real project of some degree of complexity, working in an unknown domain and where students can see the results of their architectural design as a product.

—The professor wants to develop a hands-on course for students to understand and apply SA theory.

—The industry expects students to know how to design the architecture of a real and complex software system, be able to work in a team making decisions, and see the consequences of those decisions.

—Students have some experience working on complex development projects and need to develop team decision-making skills, working on projects of similar complexity to those in the real world. This situation involves the project having various functional and non-functional requirements, constraints, and adequate management.

Solution: Students learn SA through an actual completed project where they can see the results of their architectural design as a product. We recommend working in teams of between 3 and 5 students. It is essential to clarify that the evaluation of equal contributions of team members is not a primary objective of this pattern. Evaluating team contributions is a broader challenge encompassing several models and approaches.

Working with real clients by providing meaningful and realistic industrial scenarios and problems for the projects requires the University to have agreements with companies [de Beer and Angelov 2015] and a project bank. Many universities have biannual calls for proposals where companies apply for project ideas, and faculty evaluate and choose suitable projects. In this case, students will be able to do requirements engineering comprehensively, specify quality attributes, and apply methods to concert with stakeholders the attributes that will define the architecture [Rupakheti and Chenoweth 2015]. The actual customers could participate in requirements capture, prioritization of requirements in each deliverable, and attend the delivery meeting of each iteration.

The project to be chosen must be sufficiently limited to be able to be worked on in one academic period. Projects of low, high or medium architectural complexity can be chosen depending on the student's skills. The professor should have the intuition and experience of what kind of projects can be feasible according to the level of their students. Figure 3 shows the main elements of the pattern solution.

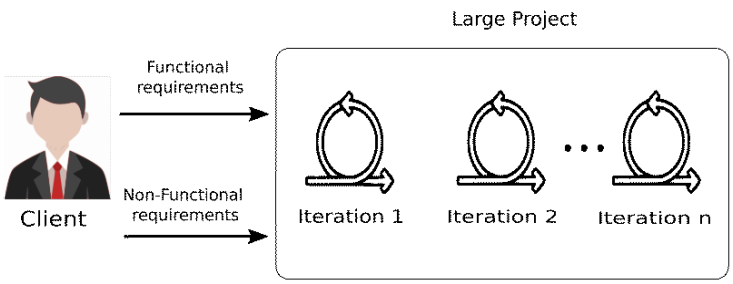

Fig. 3. Large Project-based training pattern

Projects may involve real customers or industry experts providing requirements and evaluating demonstrations. In addition to assisting with the evaluation workload, industry experts provide students with practical feedback based on current industry best practices [Weerawarana et al. 2012]. Therefore, a relationship between university and business is required. This relationships can be created by taking advantage of customers who are graduates of the same educational institutions [Rupakheti and Chenoweth 2015].

Universities can sign agreements with companies seeking partners. Initially, the companies enter a probationary period, after which they become partners supporting the education process [de Beer and Angelov 2015].

It is common for clients to be reluctant to participate in class projects because they are short on time due to their work schedules. The project could ensure that client participation is manageable for their time. These alternatives could include regular project status updates, well-structured feedback sessions, or utilizing representatives from the client organization who can liaise with students on a more frequent basis.

Student prerequisites
Mandatroy:

—Object Oriented Programming.

—Database design.

—Small scale web development.

—Small scale movil development.

Desirable:

—Operating systems.

—Distributed Systems.

—Software Engineering.

Example of pattern use: The Colombian Red Cross needs to develop a software system to connect people with blood collectors at the right time and place [Palacin-Silva et al. 2017]. The fundamental objective is to help save human lives. It requires designing a web application where people register and provide basic information, such as blood type and area of residence. The algorithm to be developed will automatically notify donors when their blood is needed in their area, and donors can reserve a date for blood donation. The health center's application offers tracking of all appointments and provides a channel to notify about the need for blood. The application should have a secure authentication and authorization system and be scalable if it starts to be used in other cities across the country.

The project described above is an example of a real project. Students will have to interact with real customers, capture functional and non-functional requirements, propose an architecture from the quality attributes selected as drivers, make decisions about which technology is the most appropriate (and other types of decisions), evaluate the chosen architecture, and make an implementation by iterations.

Competencies (or learning outcomes) addressed: With this training pattern, the following competencies are developed (See Appendix 2):
Mandatory competencies: C1, C2, C5, C8, C18, C22, C23.
Optional competencies: C3, C4, C9, C20, C24.

Variants: None.

Advantages:

—Students can work on a real-life project with real clients and develop various SA skills.

—Working on real projects with real clients could allow students to receive financial compensation or be linked to work with the company shortly.

Disadvantages:

—Students will not be able to experiment with many architectural styles, as they will have to choose the most appropriate style for the selected project.

—Working with real clients involves partnerships between the University and the companies.
—It is not easy to involve real clients in class projects, as business people are busy.

Reported experiences with the use of the formation pattern:

—Exploration on Theoretical and Practical Projects of Software Architecture Course [Zhang et al. 2020].
—Extensive Evaluation of Using a Game Project in a Software Architecture Course [Wang 2011].
—Using game development to teach software architecture [Rupakheti and Chenoweth 2015].
—Comparison of learning software architecture by developing social applications versus games on the android platform [Wu and Wang 2012].
—Scrum as a Method of Teaching Software Architecture [Wedemann 2018].
—Designing and applying an approach to software architecting in agile projects in education [Angelov and de Beer 2017].
—Fontys ICT, Partners in Education Program: Intensifying Collaborations Between Higher Education and Software Industry [de Beer and Angelov 2015]

Related patterns:
Large Project-based training pattern can be combined in an SA course with other training patterns that involve little student effort and allow the development of various decision-making competencies. For example, Case-based training, Problem-solving-based training, and Games-Based training.

3.3 Training pattern 3

Name: Open-source-projects training.

Problematic context:
Using open-source software projects in a software engineering course has many advantages. For example, it allows students to learn good coding practices from real-world projects and gives them an insight into a real project. However, it is difficult for instructors and students to contribute to such projects. One of the first challenges is identifying and selecting the right project with the right size and complexity. Other challenges are the students' inexperience, the course's limited duration, and the product's informal practices [Hu et al. 2018].

When recent graduates join the software industry, one of the initial challenges they face is developing software components on existing and usually large projects [Weerawarana et al. 2012]. In colloquial software engineering jargon, this is known as a "brownfield scenario," instead of a "greenfield scenario" in which a software engineering team starts developing a project from scratch. Students find completely new scenario projects easy to approach. In such scenarios, students are more confident as they only need to understand the requirements and have complete control over the software system's architecture, design, and structure. In contrast, existing project scenarios tend to intimidate students as they require skills in dealing with systems made by other teams, reading and understanding thousands of lines of source code, and understanding the models and architecture decisions that others made.

Forces:

—The professor wants students to develop skills for active contributions to open-source projects.
—The professor wants to train in SA through a pre-existing software project to develop skills to face systems developed by other teams.
—The industry expects students to know how to modify a software architecture for a real pre-existing system.
—The professor wants to develop a practical course for students to learn how to modify the architecture of an existing system.

—Students have some experience working on complex development projects.

Solution: Work with an open-source project to give students the unique opportunity to learn attitudes only present in real-world scenarios, which can increase their skills and self-confidence. Students can make components and extensions with open-source projects, fix bugs, or analyze the architecture. In addition, students can interact with other architects and developers, extract GitHub issues, receive community feedback, and study design and architecture documents [Van Deursen et al. 2017]. Figure 4 shows the main elements of the pattern solution.

The professor can previously select open-source projects, taking into account their architectural complexity and students' skills. In addition, students can have the opportunity to choose the project of their choice and thus increase their level of motivation in the learning process.

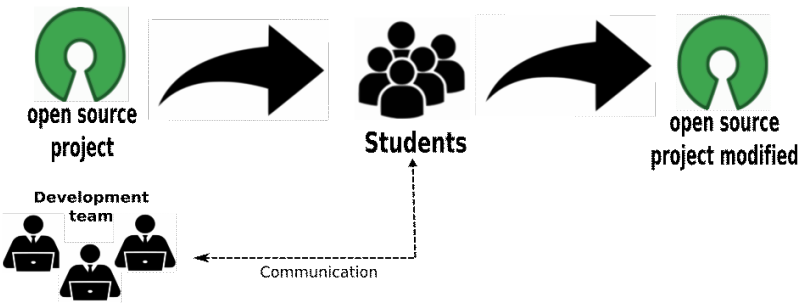

Fig. 4. Open-source project-based training pattern

Student prerequisites
Mandatory:

—Object Oriented Programming.

—Database design.

—Small scale web development.

—Small scale movil development.

Desirable:

—Operating systems.

—Distributed Systems.

—Software Engineering.

Example of pattern use: The professor prepares a list of candidate open-source projects to be worked on by the students during the course as shown in Table II [Pinto et al. 2019]:

Table II. Sample list of open-source projects

No	Project	Language	Domain
1	Catch-the-pigeon	Java	Android game
2	Jabref	Java	BibTeX manager
3	Gnome-music	Python	Music player
4	L.Office Impress	Object-C	Office suite
5	Noosfero	JavaScript	Content Management System
6	Prezento	Ruby	Web interface tool
7	Diaspora	Ruby	Social network
8	Amadeus	Python	Online learning system
9	Kalibro	Ruby	Source code analyzer
10	Gestorpsi	Python	Clinic organization system
11	Analizo	Perl	Source code analyzer
12	Cakephp	PHP	Web framework
13	Liferay-portal	Java	Web platform for building business
14	Joomla!	PHP	Content Management System
15	Teammates	Java	Education management tool

Then, students, individually or in teams, choose the project where they will make the contribution guided by the instructor and their interests.

After choosing the project, the next step is to select the task the student will work on in the course [Pinto et al. 2019]. This work can be agile and democratic; for example, the students and the instructor can meet one day each week, open the list of project problems and discuss what can be worked on. Contributions can be of 4 types according to Hattori and LanzaHattori2008:

(1) *Forward engineering*, by adding new features. For example, a contribution to the LibreOffice Impress repository is to allow the user to change slides using a smartphone.

(2) *Reengineering*, e.g., refactoring activities. Example: refactoring some layers that are not quite appropriate according to Android best practices.

(3) *Corrective*, e.g., bug fixes. For example, an import process to a database works only with MySQL but does not work when using Postgres.

(4) *Management*, e.g., updating documentation, reporting bugs, adding tags on issues, following scrum processes (sprints, user stories, planning poker, etc.).

Finally, the professor follows up on the open-source projects. Even if the instructors are not experts in open-source projects, their active participation is essential, for example, by researching or contributing to the open-source project.

Competencies (or learning outcomes) addressed: With this training pattern, the following competencies are developed (See Appendix 2):
Mandatory competencies: C11, C22, C23.
Optional competencies: C24, C14.

Variants: None.

Advantages:

—The students work with real projects.

—Skills are generated by interacting with version control systems.

—Students become members of an active development community.

—In large systems of thousands of lines of code, there is a need for architecture and simple models that help understand the system's complexity [Ciancarini et al. 2016].

Disadvantages:

—Students and instructors have to deal with the complexity of a large project's architecture and source code organization.

—Interacting with the community can be complicated; for example, interaction through a mailing list is complex as you don't know who is who and who will respond.

—At first, students must deal with the complexities of understanding and configuring the software development environment, the change control system, command-line skills, etc.

—This strategy proves more complex for instructors to orient students properly.

Reported experiences with the use of the formation pattern:

—Training software engineers using open-source software: the students' perspective [Pinto et al. 2019].

—A Collaborative approach to teaching software architecture [Van Deursen et al. 2017].

—Promoting creativity, innovation, and engineering excellence [Weerawarana et al. 2012]

Related patterns: Open-source-projects training can be combined in an SA course with other training patterns that involve little student effort and allow the development of various decision-making competencies. For example, Case-based training, Problem-solving-based training, and Games-Based training.

3.4 Training pattern 4

Name: In-house project-based training.

Problematic context: Working with real-world open-source projects involves challenges for students and instructors, which are (i) the complexity of the source code as it requires understanding the structure of the entire project, (ii) interaction with the open-source project community is done through mailing lists and not knowing people (iii) understanding and configuring the software development environment involves knowing operating systems such as Linux, version control system, working with command line, and (iv) the lack of time to contribute, sometimes the duration of the course is not enough to know the project and then make contributions [Pinto et al. 2019].

Despite the advantages of confronting students with modifying real systems, working with open-source projects can become too complex for students and professors. Therefore, an alternative is for professors to have their open-source application to teach SA with enough complexity of an industrial system.

Forces:

—The professor wants to train in SA through an open-source project he created to solve some of today's common architectural problems.

—The professor wants to develop a hands-on course for students to learn architecture concepts from an existing system.

—The industry expects students to know how to modify a SA for a real pre-existing system.

—Students do not have experience developing complex development projects.

—Students need a custom open-source application designed for the classroom to create a more engaging and personalized learning experience, which may not be fully achieved through external open-source projects.

Solution: Professors can have their open-source application to teach software architecture with enough complexity of an industrial system. For example, professors can use their E-commerce B2C system as a pedagogical tool to work on availability, security, and scalability attributes [Wei et al. 2020]. These projects may come from industry or may have been built by Professors. Periodically, these types of projects can be reviewed by the industry to see if they comply with the characteristics of an industrial system. (see Figure 5).

In this way, students are introduced to best practices widely used in industry to solve some of today's common architectural problems [Wei et al. 2020]. Using a concrete and realistic case study of a familiar area gives students a better context for applying the architectural principles learned in the lesson. A concrete application scenario in this open-source project supports each concept and theoretical principle students learn in theory. Students can download the projects to their machines, study the source code, read the manuals, and run and test them.

By joining the efforts of several faculty members from different universities, a repository of realistic open-source projects from industry sectors can be created so that instructors can refer to them to enhance the SA learning experience, even if the instructors are not software engineers in practice.

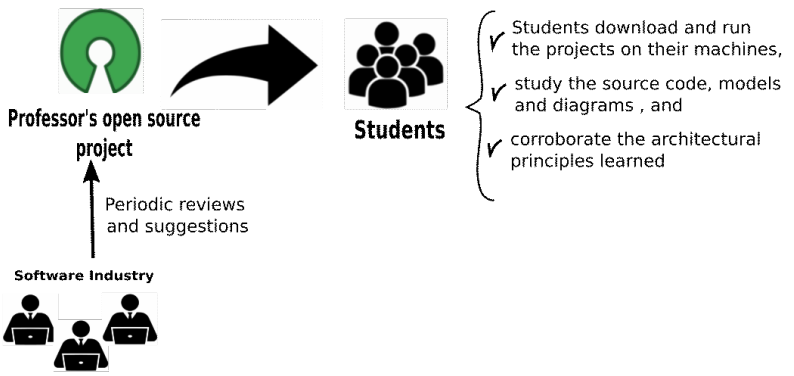

Fig. 5. In-house project-based training pattern

To maintain this type of application, the professor can host them in a code repository on GitHub and help collaborators (instructors, industry friends, master students) to make updates and improvements. It would be ideal to have an extensive repository of several open-source projects dedicated to training. Industry can be an excellent partner for universities to know if, in time, the application is still valid as a business application.

Student prerequisites

Mandatory:

—Object Oriented Programming.

—Database design.

—Small scale web development.

—Small scale movil development.

Desirable:

—Operating systems.

—Distributed Systems.

—Software Engineering.

Example of pattern use: At the beginning of the course, the professor introduces his students to the open-source e-commerce application (developed by the professor) that will support the course's theoretical concepts.

The course introduces the traditional monolithic layered architecture for building a web application and then discusses the disadvantages and possible improvements. In a monolithic system, applications are deployed as a web server unit. For minor traffic, one server may be sufficient. But when the application server receives a lot of traffic during the season of high request traffic, it will be necessary to duplicate the projects on more servers. These systems are easy to develop by students. The e-commerce system example project allows students to understand the above concepts.

Having explained the advantages and disadvantages of traditional architecture, we can move on to distributed architecture. Each module in the traditional architecture must be removed from the monolithic system, and a separate system is developed. Each module will run in a separate Docker container and communicate with other modules through RESTful web services. Separating these modules can also facilitate better team collaboration and project management.

Deployment and operations can also be part of this course. During the first half of the course, the VMWare virtual environment is used to simulate deployment. Then, the students can switch to cloud deployment. The DevOps approach can also be introduced.

Competencies (or learning outcomes) addressed: With this training pattern, the following competencies are developed (See Appendix 2):
Mandatory competencies: C01, C05, C08.
Optional competencies: C11, C12.

Variants: None.

Advantages:

—The professor has an open-source application for the needs of his course.

—Students can download and run the projects on their computers, study the source code, models, diagrams, and manuals.

—The students in the application evidence item every concept learned about SA.

Disadvantages:

—For the professor, it can be complex to develop or have examples of open-source applications tailored to the course.

—The sample applications must be updated over time according to technological advances.

—The project may not be motivating for the student.

Reported experiences with the use of the formation pattern:

—Teaching Distributed SA by Building an Industrial Level E-Commerce Application [Wei et al. 2020].

—A Collaborative Approach to Teaching Software Architecture [Van Deursen et al. 2017]

Related patterns:

Open-source project-based training pattern can be combined in an SA course with other training patterns that involve little student effort and allow the development of various decision-making competencies. For example,

Case-based training, Problem-solving-based training, and Games-Based Training.

3.5 Training pattern 5

Name: Cases-based training.

Problematic context: Making correct architectural decisions while constructing a software system is one of the significant skills a future software architect must develop. In addition, decisions provide a choice made by the software architect in a specific context, along with its justification or rationale.

Decisions may relate to choosing the structure of the application or system, selecting a technology to implement the design, or a trade-off between quality attributes. Whatever the context, an exemplary architecture decision helps development teams make the right technical decisions. Therefore, an architecture decision should be explained in terms of the following characteristics:

—Available design alternatives.

—Justification of the decision.

—Document the decision.

—Effectively communicate the decision to stakeholders.

The professor must provide, during the course development, the necessary conditions for their students to learn how to make decisions.

The following is a concrete scenario where the student must make software architecture decisions. When designing a new software system, based on the quality attributes of the new system (scalability, availability, security, performance, fault tolerance, elasticity, among others), the architect must first select a small set of attributes that will be the most relevant to be satisfied by the system, and that will become the architecture drivers (it is not possible to design a system that helps all attributes). Next, a decision must be made regarding which architectural style or styles favor these quality attributes. For example, a microservices style tends to scalability, elasticity, and evolution, but at the same time, fault tolerance and reliability suffer when too much inter-service communication is used; in a pipeline architecture style, overall cost, simplicity, and modularity are its main strengths as it is monolithic, but elasticity and scalability are deficient; in an event-driven architecture style, performance, scalability, and fault tolerance are its main strengths, but simplicity and testability are relatively low [Richards and Ford 2020]. In summary, the architectural styles chosen must be justified according to the application's requirements. In addition, an architect must decide the type of application to be developed: web, web single page, desktop, mobile, hybrid (web and mobile). Finally, the architect must choose the appropriate technologies for the system by answering the questions: Which technologies help to implement the selected architectural styles? Which technologies allow the implementation of the selected application type? Which technologies help to meet the specified non-functional requirements?

Forces:

—The professor wants to develop skills related to making software system architecture decisions in his students.

—The industry expects students to know how to make SA decisions.

—The professor wants to avoid going to the development of software projects because they are too time-consuming.

—Students do not have sufficient skills to carry out the implementation of a software project.

Solution: Case studies are real-life business scenarios that allow students to analyze, apply the concepts taught, and apply trade-offs within a realistic context.

Case-based instruction focuses on the design and analysis of real business software projects. Students experience and use the theory and technology of SA design applied to specific projects to enhance the teaching effect [Lieh Ouh et al. 2020]. The main steps of this formation pattern can be seen in Figure 6.

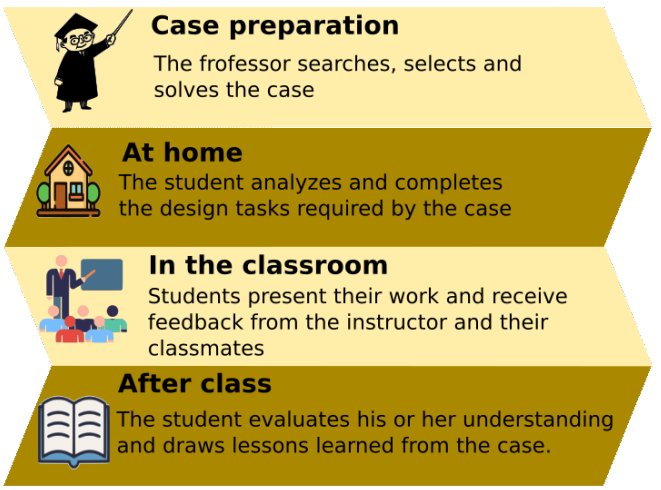

Fig. 6. Cases-based training pattern

Cases are typically taught using the flipped classroom. Reversed classroom learning promotes greater immersion in a topic than traditional teaching, improving student engagement, interaction, and cooperation by providing students with the content before the classroom [Gonçalves et al. 2020]. The inverted classroom has three moments before the encounter with the professor, during and after.

The training pattern proposes conducting industry-related workshops in case studies, which can guide and take participants through the thought processes of a junior architect [Lieh Ouh et al. 2020]. These workshops could include examples of real projects showing situations that a solution architect goes through in a real working environment. The professor could draw these workshops from the experiences of architects in the industry, architecture blogs, and others.

The general steps to be followed to conduct a case are [Oliveira et al. 2022]

(1) First, the professor teaches the topics of SA.
(2) The professor defines the topic and learning objectives of the case. The professor should establish clear criteria for selecting cases. These criteria include the relevance of the case to the learning objectives, the appropriate level of difficulty for the students, authenticity, and applicability of the case to real-world situations.
(3) The professor searches and selects the case. The professor can search for cases from various sources, such as textbooks, academic journals, specialized websites, educational case databases, and other online resources. They can also consider creating their cases based on real-world situations.
(4) Adaptation and customization. In some instances, the professor may need to adapt the case better to suit the needs of their students. This adaptation might involve simplifying or expanding parts of the case or adjusting information to make it more relevant to the educational context.
(5) Preparation of didactic material. Once the case has been selected and adapted, the professor should create the material to present to the students. It could include detailed case descriptions, relevant data, reflection, discussion questions, and additional resources to help students understand the context.
(6) Individual or group work. Students work individually or in groups to analyze the case, identify problems, propose solutions, and discuss their findings. This phase encourages active participation and critical thinking.

(7) Discussion and analysis. The professor facilitates class discussions where students share their analysis, proposed solutions, and reasoning. This situation can lead to enriching discussions and a deeper understanding of the concepts involved.
(8) Synthesis and conclusion. At the end of the process, the professor summarizes the key lessons that can be drawn from the case and its relevance in the broader learning context.

Case-based learning is an effective educational strategy for developing decision-making skills and fostering critical thinking in students. Some ways instructors can ensure that the case study effectively addresses these skills are:

—Selection of relevant and challenging cases. Instructors should choose pertinent cases to the course objectives and challenges for students. Cases should reflect real-world situations in which students will face complex decisions.
—Clear definition of objectives. Before presenting the case, instructors should set clear objectives for what students should learn and accomplish. This provides clear direction and ensures that the case is aligned with the desired learning outcomes.
—Stimulating discussion. Cases should be designed so that there is no single correct answer. This discussion will encourage student debate and discussion, fostering critical thinking by considering different perspectives and solutions.
—Provide limited information. Cases should present limited information, thus simulating real situations where decision-makers often must work with incomplete or ambiguous information. This situation will help students develop skills in identifying and gathering the information required to make informed decisions.
—Promoting reflection. After analyzing the case, students should be encouraged to reflect on their decisions and how they arrived at those conclusions. Questions such as “Why did they choose that option¿‘, “What other alternatives did they consider?”, and “What did they learn from this process?” can encourage self-reflection.
—Provide constructive feedback. Instructors should provide constructive feedback on students’ decisions and analysis. This feedback not only validates their effort but also provides them with ideas for improving their decision-making skills in the future.
—Linking to theory and concepts. After analyzing the case, linking students’ conclusions and decisions to relevant SA theories and concepts is essential. This linking helps students understand how theory applies to practical situations.

Professors may primarily conduct case-based instruction as a “flipped classroom.” Before the face-to-face encounter, each student has some time to analyze the case and complete the design tasks required by the case. During the face-to-face meeting, students present their work and receive feedback from the instructor and their peers. After the face-to-face meeting, the student evaluates their understanding and draws lessons from the case.

Some activities that the student can do before the face-to-face encounter are:

—Students can receive pre-recorded videos, readings, or multimedia resources from the professor. This material can explain key concepts, demonstrate processes, present examples, and provide context for the case to be addressed in a face-to-face meeting.
—Students may read textbook chapters, scholarly articles, or papers related to the case to be covered in class.
—Students may complete exercises, quizzes, or assignments related to the case. These activities can help them assess their understanding and prepare for the face-to-face meeting.
—Students may be encouraged to research online or in other sources to delve deeper into the topic and discover additional information.
—Students may be invited to generate questions or concerns based on previous content. These questions can serve as a starting point for discussion in the face-to-face meeting.

—Students can participate in online forums or platforms where they discuss previous content with their peers, answer questions posed by the professor, or generate debates.
—Students can write reflections, summaries, or outlines on previous content to organize their thoughts and prepare for class interaction.

Student prerequisites
Mandatory:

—Object Oriented Programming.
—Database design.

Desirable:

—Operating systems.
—Distributed Systems.
—Software Engineering.

Example of pattern use: The professor selects the following case to work with his students using the flipped classroom. A nationwide health system has been designed to monitor students' health in primary, secondary, and tertiary education institutions. The student is required to design a distributed system that addresses the security, performance, maintainability, and scalability qualities of this health system. Decisions about the architectural design may favor one of the qualities but will likely offset another. For example, the learner will need to decide whether to retain healthcare data in the storage available at each institution during the triage process or to access a centralized remote system to retain the data directly. Adopting the former may allow better system decoupling but risks data inconsistency across institutions. On the other hand, adopting the latter may achieve better architecture and data consistency maintainability but risks a single point of failure at each institution. For each tradeoff, learners should be able to recommend mitigating actions [Ouh and Irawan 2019].

Competencies (or learning outcomes) addressed: With this training pattern, the following competencies are developed (See Appendix 2):
Mandatory competencies: C01, C02, C05, C08, C12, C18, C22, C23.
Optional competencies: C03.

Variants: One way to work with cases is through lectures with guests from the software industry. The professor can organize from one to four lectures or talks throughout the course. In each lecture, the guest speaker tells the details of a real architectural case: the context, problem, decisions, and results. In this way, students learn from the architects' lived experience.

Advantages:

—The cases focus on architecture issues, and the code implementation is left aside.
—The cases allow a short time to develop SA decision-making skills.

Disadvantages:

—The solution to the cases is not unique; how the SA is defined ultimately depends on the context, stakeholders, concerns, and the architecture's purpose [Lieh and Irawan 2018]. Therefore, the professor has a significant challenge when solving the case.

Reported experiences with the use of the formation pattern:

—Applying case-based learning for a postgraduate Software Architecture course [Ouh and Irawan 2019].

—Did our Course Design on Software Architecture meet our Student's Learning Expectations? [Lieh Ouh et al. 2020].

—Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course [Lieh and Irawan 2018].

—Improved Teaching Model for Software Architecture Course [Ji and Song 2015].

—Flipped Classroom Applied to Software Architecture Teaching [Gonçalves et al. 2020].

Related patterns:
Cases-based training pattern can be combined in an SA course with other training patterns that involve software project development. For example, Mini-Projects-based training, Large Project-Based Training, Open-Source project-based training, and In-house project-based.

3.6 Training pattern 6

Name: Problem-solving-based training.

Problematic context: Making architecture decisions as a team while constructing a new software system is one of the most critical skills a future software architect must develop. The most important decisions are made at the beginning of the creation of an application: technologies to be used, architecture patterns, and trade-offs between quality attributes, among others. After comes the development and maintenance of the application.

The professor must provide, during the course development, the necessary conditions for students to learn to make such decisions as a team.

When students work in teams and are faced with complex decision-making, many challenges present themselves:

—Differences in team dynamics. Differences in personality, work style, and communication among team members. Group dynamics can affect the efficiency and effectiveness of the decision-making process.

—Communication difficulties. Inefficient communication can lead to misunderstandings, lack of clarity, and coordination problems. Effective communication is essential for sharing ideas, discussing solutions, and reaching a consensus.

—Time management. Working in teams can require effective time coordination, especially when tackling complex problems. Planning and time management can be challenging, as students must balance multiple responsibilities and tasks.

—Conflicts and disagreements. When students work in teams and face complex decisions, disagreements and conflicts over the best solutions will likely arise. Managing these challenges can be complicated.

—Making complex decisions. Problems often do not have clear, single answers. Making decisions in an environment of uncertainty can be challenging for students as they must evaluate different options and consider multiple perspectives.

—Equity in contribution. Ensuring that all team members participate equally and contribute meaningfully can be challenging. Some students may be less inclined to express their ideas or may be dominant in the process.

—Pressure to reach consensus: Reaching consensus can be difficult when there are differences of opinion on the team. Some students may feel pressure to compromise on their views, which can affect the quality of decision-making.

Forces:

—The professor wants to develop skills in his students related to team decision-making in architecting a new software system.

—The industry expects students to know how to make SA decisions.

—The professor does not want to go to the development of software projects because they are too time-consuming.

—Students do not have sufficient skills to carry out the implementation of a software project.

—When students work in teams and are faced with complex decision-making, many challenges arise, such as personality differences, communication difficulties, conflicts and disagreements, time management, and fairness in contribution, among others. To overcome these challenges, instructors must expose their students to exercises that allow them to develop teamwork and decision-making skills.

Solution: The problem-based learning approach allows working in teams of students, solving real or fictitious architectural problems, and instructors play a minimal role and do not interfere in the discussion. As students explore difficulties, instructors can act as facilitators and use guiding questions to bring them back to the main learning objective. For example, one of the subgroups explains their design solution.

The steps to apply the problem-based approach are:

(1) Problem identification and selection. The instructor identifies a realistic, relevant, stimulating, and challenging problem for the students related to the AS course's objectives.

(2) Presentation of the problem. The instructor presents the problem to the students clearly and concisely. Provides the information necessary for students to understand the context of the problem.

(3) Team building. The instructor organizes students into collaborative teams. You can do this randomly or consider individual strengths and abilities. It is important to encourage diversity in the teams to promote different perspectives.

(4) Analysis and understanding of the problem. The instructor invites teams to analyze and fully understand the problem. In addition, the instructor encourages students to raise questions, identify missing information, and define the objectives of the problem.

(5) Idea generation and discussion. Teams generate ideas and possible solutions to address the problem. Discussion is encouraged in teams to explore different approaches.

(6) Analysis and development of solutions. Teams analyze different proposed solutions and evaluate their pros and cons. This activity encourages critical thinking by considering the solutions' ethical, social, and technical aspects.

(7) Presentation and feedback. Each team socializes the different solutions in front of the others. The instructor gives constructive feedback on the solutions and the decision-making process.

An example of the problem-based approach is the Katas of Architecture. Kata is taken from Karate and refers to an individual training exercise. An Architecture Kata is an activity defined by Ted NewardTed Neward is a freelance software development architect and mentor in Sacramento, California where the goal is to design the architecture of a system close to reality. The activity is usually done in teams, where each group is assigned an exercise to be solved in a given time. There is a person with the role of moderator, who acts as a client, project manager, and end user. The moderator has the task of clarifying any concerns that may arise in the kata (see Figure 8).

The steps that can be followed to apply the Architectural Katas are:

(1) The instructor forms work teams of 3 to 5 students per group.

(2) The exercise to be solved is randomly assigned. Ted Neward defined an initial list of exercises for architectural katas on the website `architecturalkatas.com`. The instructor can ask the site to select a kata randomly (see Figure 7). This list of exercises is extensive, and the instructor can choose any of them. Each kata

consists of functional requirements, non-functional requirements, and constraints. When there are doubts about the requirements, the instructor can be consulted. The students can make assumptions about missing requirements to make their design decisions.

(3) Next comes the discussion, for which the teams can be given 45 minutes to propose an architectural solution to the problem. Since the problems are short in wording, assumptions should be made about some missing requirements or technologies to use. We recommend that students use the C4 model to design their proposals. For the choice of architecture patterns according to the quality requirements of the problem, we recommend chapters 10 to 18 of Mark Richards' book Fundamentals of SA.

(4) Next comes the presentation of the proposals. One or two people per team are chosen to present their proposals.

(5) Finally comes the "voting." The rest of the teams vote based on the presentation:
—*Good job (thumbs up):* The exercise was solved positively, the solution was coherent, and reasonable technologies were selected.
—*Bad job (thumbs down):* Student made critical assumptions without validity.
—*Could have been better (neutral or horizontal hand): n*o clear vision of the project, and essential aspects were forgotten.

Pantoja et al. show an example of a kata application in an SA course [Pantoja et al. 2023b].

Fig. 7. Example of a architectural kata generated by the site `architecturalkatas.com`

Student prerequisites
Mandatory:

—Object Oriented Programming.
—Database design.

Desirable:

—Operating systems.
—Distributed Systems.

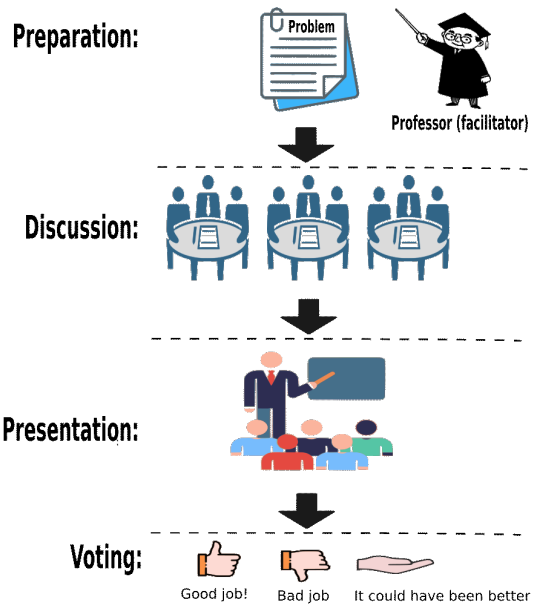

Fig. 8. Problem-solving-based training pattern

—Software Engineering.

Example of pattern use: Develop in teams of three students the following Architecture Kata. The discussion phase should last 45 minutes. Use the C4 model to design the solutions.
The Road Warrior: A primary travel agency wants to build a next-generation travel management dashboard that allows travelers to view all their existing bookings, organized by trip, either online or via a mobile device. The system must support over 10 thousand registered users worldwide.

The requirements for this system are:

—Must connect to the agency's existing airline, hotel, and car rental system. The connection must allow reservations to be automatically loaded through frequent flyer accounts, hotel points accounts, and car rental rewards accounts.

—Customers can manually add existing reservations.

—Items in the dashboard can be grouped by trip, and once a trip is completed, items are automatically removed from the dashboard.

—Users can also share their trip information through social networks.

—Richest possible user interface across all platforms.

Additional context:

—Must integrate seamlessly with existing travel systems.

—Partnerships are being negotiated so that there are "favored" suppliers.

—Must operate internationally.

Competencies (or learning outcomes) addressed: With this training pattern, the following competencies are developed (See Appendix 2):
Mandatory competencies: C01, C02, C05, C08, C12, C18, C22, C23.
Optional competencies: C03.

Variants: None

Advantages:

—The problems are focused on architectural issues, and the implementation is not done in code.
—The problems allow a short time to develop skills in the software architect.

Disadvantages:

—There are no one-size-fits-all solutions to problems; the professor requires experience to guide students' proposals.

Reported experiences with the use of the formation pattern:

—Teaching adult learners on software architecture design skills [Lieh and Irawan 2019] [Gonçalves et al. 2020].
—Applying case-based learning for a postgraduate software architecture course [Ouh and Irawan 2019].
—Aligning Software Architecture Training with Software Industry [Pantoja et al. 2023b].

Related patterns:
Problem-solving-based training pattern can be combined in an SA course with other training patterns that involve software project development. For example, Mini-Projects-based training, Large Project-Based Training, Open-Source project-based training, and In-house project-based.

3.7 Training pattern 7

Name: Games-Based training.

Problematic context: Teaching SA is complex because the architect's role is multifaceted. The architect requires developing technical, analytical, and communication skills. Most talented architects have acquired extensive knowledge over many years of experience. However, if we want the architectural design to be systematic and reproducible, we must improve teaching methods. It is not acceptable to wait for an aspiring architect to accumulate 10 or 20 years of experience if we consider software engineering a real engineering discipline. [Cervantes et al. 2016].

Professors need fun teaching methods for shortened training time related to SA decision-making. Game-based training is an educational strategy that uses elements of games to foster student engagement, participation, and learning. While it is effective for many, it also presents challenges for educators. Here are some of those challenges:

—Designing effective educational games. Creating games that are educational and engaging in SA topics can be tricky. Games must effectively balance fun with educational content, ensuring that learning objectives are met without sacrificing the game's appeal.
—Alignment with learning objectives. Ensuring that games address specific learning objectives can be challenging. Instructors should design games that relate directly to the topics and skills being taught.
—Assessing learning. Determining how to assess student learning through games can be complex. Instructors must develop assessment methods that are appropriate and that reflect the knowledge and skills gained through the game experience.

—Difficulty in progression design. Educational games should have an appropriate difficulty curve for students. Students may lose interest or become frustrated if the game is too easy or difficult.

Forces:

—The professor wants to develop skills in their students related to making architectural decisions for a new software system in a fun way, encouraging engagement, participation, and learning.

—The professor does not want to go to the development of software projects because they are too time-consuming.

—The industry expects students to be able to make complex SA-related decisions and work in teams..

—Students do not have sufficient skills to carry out the implementation of a software project.

Solution:

Games can provide a valuable illustration of the design decision-making process and teach students the power of team interaction to make good decisions [Lago et al. 2019]. Games allow software design skills to be developed in a fun and engaging way for students (see Figure 9).

The game is not a substitute for “traditional” instruction on design but rather a complement to such education [Lago et al. 2019]. Consequently, players are not expected to learn in detail how to design or make optimal design decisions simply by playing the game. Instead, the game can be used as a starting point for more in-depth discussions about the complexities of architectural design or to practice various aspects of the design process. Game participants can understand, in a short time, entertainingly, and compellingly, how design is done and the different concepts and activities associated with this crucial activity.

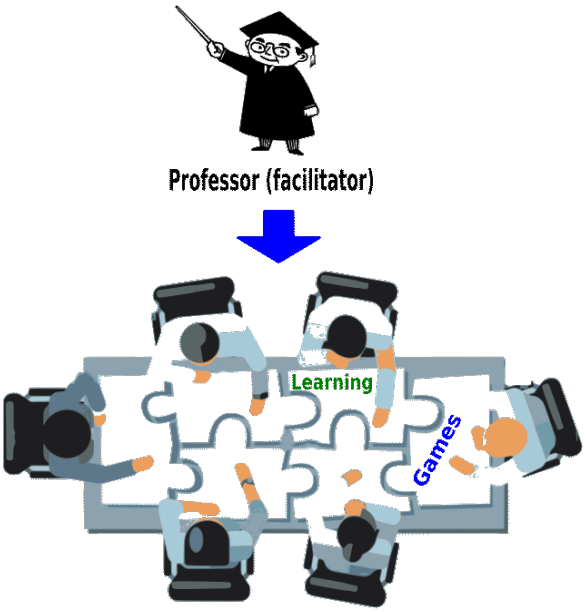

Fig. 9. Games-based training pattern

The instructor can apply games to teach SA decision-making, such as:

(1) Design architectures that meet the quality attributes necessary for system success and guide designers to make informed decisions consistent with project requirements and expectations. This situation involves identifying the most critical quality attributes for the system. These attributes vary according to the context and requirements of the project. Next, attributes are prioritized according to their importance to the system. Depending on the software domain and user needs, some attributes may be more critical than others.

(2) Evaluate and analyze software architectures in terms of quality attributes such as performance, scalability, availability, security, usability, maintainability, and other factors relevant to the system. Architects and development teams must understand how architectural decisions impact quality attributes and how trade-offs can influence the final architecture.

The “Example of pattern use” section illustrates games that help to make the decisions described above.

Student prerequisites

Mandatory:

—Object Oriented Programming.

—Database design.

Desirable:

—Operating systems.

—Distributed Systems.

—Software Engineering.

Example of pattern use: Below, we describe three games related to SA training and decision-making.

Smart Decision is an architectural design game. This game’s core is applying the Attribute-Driven Design (ADD) method. ADD focuses on translating the essential requirements of the software system (also called architectural drivers) into a set of structures from which the system is developed. The translation of the architectural drivers into structures is the architectural design process. ADD is usually iterative: a subset of drivers is selected at the beginning of an iteration, and then design decisions are made to identify elements and create structures from them to satisfy the chosen drivers. Other drivers are then selected, more structures are established, or existing structures are refined until an initial architecture is created. The design process involves making decisions, often involving choosing among proven, documented solutions to recurring design problems. These proven solutions, which we call design concepts, are the building blocks of the design. Design concepts, such as design patterns or tactics, can be conceptual or more concrete, such as application frameworks.

Smart Decision game mechanics include game rules, steps, and scoring. Smart Decisions requires a facilitator to guide players to understand the game mechanics through a presentation. The game requires a minimum of two players and six players competing against each other. These players can be individuals or teams. The game is played in a series of rounds, each representing an iteration in designing a greenfield system. The details and mechanics of the game are described in [Cervantes et al. 2016].

DecidArch - Playing Cards as Software Architects is a game developed to achieve three learning objectives: 1) create awareness of the logic involved in making design decisions, 2) enable appreciation of the reasoning behind candidate design decisions proposed by others, and 3) create awareness of the interdependencies between design decisions [Lago et al. 2019]. DecidArch is a board game that is inexpensive and easy to introduce in the classroom to teach undergraduate students about the concept of SA design decision-making: examining trade-offs and trade-offs between stakeholder demands and critical quality attributes in the face of moderate uncertainty. The details and mechanics of the game are described in [Lago et al. 2019].

RPG Role Playing Game is a game to support the teaching of ATAM (Architecture Trade-off Analysis Method) to computer science students either in the classroom or remotely. In this game, students assume a stakeholder role, prioritize and examine quality attributes, negotiate the priority and difficulty of scenarios, and agree on a final architecture. This game exercises negotiation skills. The details and mechanics of the game are described in [Montenegro and Astudillo 2014].

Competencies (or learning outcomes) addressed: With this training pattern, the following competencies are developed (See Appendix 2):
Mandatory competencies: C01, C08, C18.
Optional competencies: None.

Variants: None

Advantages:

—The games focus on architecture topics, and code implementation is left aside.

—Games are a fun way for students to develop SA skills.

Disadvantages:

—No interaction with real systems.

Reported experiences with the use of the formation pattern:

—Smart Decisions: An Architectural Design Game [Cervantes et al. 2016]

—A role-playing game to teach ATAM (Architecture Trade-off Analysis Method) a simulation tool and case study [Montenegro and Astudillo 2014].

—DecidArch: Playing cards as software architects [Lago et al. 2019]y [Pantoja et al. 2023b].

Related patterns:
Games-based training pattern can be combined in an SA course with other training patterns that involve software project development. For example, Mini-Projects-based training, Large Project-Based Training, Open-Source project-based training, and In-house project-based.

4. PRELIMINARY VALIDATION

We subjected the Mini-Projects-based training pattern to a preliminary validation using a focus group with expert professors from the Universidad Nacional de la Plata, Argentina. We chose a group of four professors with expertise in software patterns and SA. With this evaluation, we sought to receive feedback on the following elements:

—That the pattern is easy to read, understand and apply.

—That the name of the pattern is appropriate.

—That the patterns' structure is complete, i.e., they check if there are missing or excess fields in the proposed format.

—That the relationship between the problem and the solution is coherent.

—And some general comments by the evaluators.

The Focus group had three phases. The first was planning the activity, choosing the profile of the people and the logistics, and preparing a document with the training pattern, the place, the date, and the appropriate time. The second phase was the execution, in which we conducted the focus group in person. The execution had four parts:

(1) Presentation of each of the participants.
(2) Reading of a paragraph of the document by the authors as an introductory mechanism to the focus group.
(3) Mention the positive aspects of the pattern by the evaluators.
(4) To discuss the aspects of the document.
(5) Formulation of questions from the authors to the evaluators.
(6) Closing and thanks to all reviewers.

Finally, the third phase of the focus group was the analysis of the observations and recommendations given by the evaluators. As a result of this activity, we made the following adjustments to the training pattern:

—We improved the section of the pattern example that had unintelligible and inconsistent parts.
—We created a background section to prepare the reader for the topics related to SA.
—We rearrange the order of the parts of the pattern template.
—We improved aspects of form, such as wording.

The evaluators' recommendations for the first training pattern were used to improve the other six patterns.

5. CONCLUSIONS AND FUTURE WORK

A SA course in line with industry needs is an essential part of today's world's computer science program curricula. However, training undergraduate students in SA subjects with the skills demanded by industry has many challenges.

To design and execute a high-quality SA course, we propose to incorporate a series of training strategies that allow, on the one hand, to recreate the problems and the way of working in the classroom similar to the environments used by the industry, and on the other hand, to organize the architectural knowledge so that the classroom activities can be structured and facilitate incremental learning.

Based on the challenges of the SA courses, a catalog of training patterns could guide professors on what and how to teach, achieving the development of the most relevant competencies for the current and future industry related to the creation, evaluation, and documentation of SA in potential graduates of computer science programs.

The "Mini-Projects-based training" pattern was preliminarily evaluated through a focus group involving expert professors' feedback in SA and patterns. This activity made it possible to socialize and improve the template to make it easy to read, understand, and apply.

We will refine the other training patterns in the catalog in future work. In addition, we will create a guide that integrates the patterns and guides professors in creating and improving SA courses. Subsequently, we will evaluate the effectiveness of the training patterns present in the catalog through some case studies. We will continue researching the formation patterns, obtaining more empirical evidence that will allow us to disseminate the patterns, improve and complement the catalog, and establish more formality in the language. New patterns may emerge. We will create a guide that integrates the patterns and guides professors in creating and improving SA courses. The guide and the catalog could be combined with other tools to create a training center to train software architects in a short time according to the needs of the industry.

REFERENCES

Christopher Alexander. 1977. *A pattern language: towns, buildings, construction*. Oxford university press.

S Angelov and P de Beer. 2017. Designing and Applying an Approach to Software Architecting in Agile Projects in Education. *Journal of Systems and Software* 127, C (2017), 78–90. DOI:http://dx.doi.org/10.1016/j.jss.2017.01.029

Paris Avgeriou, Andreas Papasalouros, Symeon Retalis, and Emmanuel Skordalakis. 2003. Towards a Pattern Language for Learning Management Systems. *Educational Technology & Society* 6 (2003).

Len Bass, Paul Clements, and Rick Kazman. 2012. *Software architecture in practice, third Edition.* Pearson Education, Massachusetts, USA. https://www.amazon.com/Software-Architecture-Practice-3rd-Engineering/dp/0321815734

Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Chichester, UK. https://www.safaribooksonline.com/library/view/pattern-oriented-software-architecture/9781118725269/

Humberto Cervantes, Serge Haziyev, Olha Hrytsay, and Rick Kazman. 2016. Smart Decisions: An Architectural Design Game. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. 327–335.

Robert Chatley and Tony Field. 2017. Lean learning - Applying lean techniques to improve software engineering education. In *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET 2017*. IEEE Press, Buenos Aires, 117–126. DOI:http://dx.doi.org/10.1109/ICSE-SEET.2017.5

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

Paul Clements and Len Bass. 2010. Using business goals to inform software architecture. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference, RE2010*. IEEE, IEEE Computer Society, Sydney, NSW, 69–78. DOI:http://dx.doi.org/10.1109/RE.2010.18

Patrick de Beer and Samuil Angelov. 2015. Fontys ICT, Partners in Education Program: Intensifying Collaborations Between Higher Education and Software Industry. In *Proceedings of the 2015 European Conference on Software Architecture Workshops (ECSAW '15)*. Association for Computing Machinery, New York, NY, USA, 4. DOI:http://dx.doi.org/10.1145/2797433.2797468

Erich Gamma, Richard Helm, Ralph Johnson, and John M Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software* (1 ed.). Addison-Wesley Professional, Boston, USA. http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/ref=ntt_at_ep_dpi_1

Anderson Cavalcante Gonçalves, Valdemar Vicente Graciano Neto, Deller James Ferreira, and Uyara Ferreira Silva. 2020. Flipped Classroom Applied to Software Architecture Teaching. In *2020 IEEE Frontiers in Education Conference (FIE)*. 1–8. DOI:http://dx.doi.org/10.1109/FIE44824.2020.9274255

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

Zhewei Hu, Yang Song, and Edward F Gehringer. 2018. Open-Source Software in Class: Students' Common Mistakes. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '18)*. Association for Computing Machinery, New York, NY, USA, 40–48. DOI:http://dx.doi.org/10.1145/3183377.3183394

Zhenyan Ji and Jing Song. 2015. Improved Teaching Model for Software Architecture Course. In *Proceedings of the 2015 International Conference on Education, Management, Information and Medicine*. Atlantis Press, No City, 333–338. DOI:http://dx.doi.org/10.2991/emim-15.2015.65

Patricia Lago, Jia F. Cai, Philippe Kruchten, Remco C. de Boer, and Roberto Verdecchia. 2019. Decidarch: Playing cards as software architects. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, Vol. 2019-Janua. 7815–7824. DOI:http://dx.doi.org/10.24251/hicss.2019.940

Weigang Li. 2019. Teaching Reform and Practice of Software Architecture Design Course under the Background of Engineering Education. In *Proceedings of the 2019 International Conference on Advanced Education, Management and Humanities (AEMH 2019)*, Vol. 352. Atlantis Press, Wuhan, China, 17–21. DOI:http://dx.doi.org/10.2991/aemh-19.2019.4

Zheng Li. 2020. Using Public and Free Platform-as-a-Service (PaaS) based Lightweight Projects for Software Architecture Education. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training*. Association for Computing Machinery, Seoul South Korea, 1–11. DOI:http://dx.doi.org/10.1145/3377814.3381704

Ouh Eng Lieh and Yunghans Irawan. 2018. Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, San Jose, CA, USA, 1–9. DOI:http://dx.doi.org/10.1109/FIE.2018.8659200

Ouh Eng Lieh and Yunghans Irawan. 2019. Teaching adult learners on software architecture design skills. In *Proceedings - Frontiers in Education Conference, FIE*, Vol. 2018-Octob. IEEE, Uppsala, Sweden, 1–9. DOI:http://dx.doi.org/10.1109/FIE.2018.8658714

Eng Lieh Ouh, Benjamin Kok Siew Gan, and Yunghans Irawan. 2020. Did our Course Design on Software Architecture meet our Student's Learning Expectations?. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, Uppsala, Sweden, 1–9. DOI:http://dx.doi.org/10.1109/FIE44824.2020.9274014

Gerard Meszaros and Jim Doble. 1998. A pattern language for pattern writing. *Pattern languages of program design* 3 (1998), 529–574.

Claudia Hidalgo Montenegro and Hernán Astudillo. 2014. A role-playing game to teach ATAM (Architecture Trade-off Analysis Method) a simulation tool and case study. In *2014 IEEE ANDESCON*. 1. DOI:http://dx.doi.org/10.1109/ANDESCON.2014.7098541

Brauner R N Oliveira, Lina Garcés, Kamila T Lyra, Daniel S Santos, Seiji Isotani, and Elisa Y Nakagawa. 2022. An Overview of Software Architecture Education. In *Anais do XXV Congresso Ibero-Americano em Engenharia de Software*. SBC, 76–90.

Eng Lieh Ouh and Yunghans Irawan. 2019. Applying Case-Based Learning for a Postgraduate Software Architecture Course. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. Association for Computing Machinery, New York, NY, USA, 457–463. DOI:http://dx.doi.org/10.1145/3304221.3319737

Maria Palacin-Silva, Jayden Khakurel, Ari Happonen, Timo Hynninen, and Jari Porras. 2017. Infusing Design Thinking into a Software Engineering Capstone Course. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)*. IEEE, Savannah, Georgia, USA, 212–221. DOI:http://dx.doi.org/10.1109/CSEET.2017.41

W. Libardo Pantoja, Julio Ariel Hurtado, Bandi Ajay, and Arvind W Kiwelekar. 2023a. Training Software Architects Suiting Software Industry Needs: A Literature Review. *Education and Information Technologies* (2023). DOI:http://dx.doi.org/https://doi.org/10.1007/s10639-023-12149-x

Wilson Libardo Pantoja, Julio Ariel Hurtado, and Arvind W Kiwelekar. 2023b. Aligning Software Architecture Training with Software Industry Requirements. *International Journal of Software Engineering and Knowledge Engineering* 33 (2023), 435–460. DOI:http://dx.doi.org/https://doi.org/10.1142/S0218194023500031

Gustavo Pinto, Clarice Ferreira, Cleice Souza, Igor Steinmacher, and Paulo Meirelles. 2019. Training software engineers using open-source software: The students' perspective. In *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET 2019*. IEEE, Montreal Quebec Canada, 147–157. DOI:http://dx.doi.org/10.1109/ICSE-SEET.2019.00024

Mark Richards and Neal Ford. 2020. *Fundamentals of Software Architecture: An Engineering Approach 1st Edicion*. O'Reilly Media, Inc., Canada. 383 pages. https://www.amazon.com/Fundamentals-Software-Architecture-Comprehensive-Characteristics/dp/1492043451

Chandan R. Rupakheti and Stephen V. Chenoweth. 2015. Teaching Software Architecture to Undergraduate Students: An Experience Report. In *Proceedings - International Conference on Software Engineering*, Vol. 2. IEEE Press, Florence Italy, 445–454. DOI:http://dx.doi.org/10.1109/ICSE.2015.177

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

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

Arie Van Deursen, Maurício Aniche, Joop Aué, Rogier Slag, Michael De Jong, Alex Nederlof, and Eric Bouwers. 2017. A Collaborative approach to teaching software architecture. In *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*. ACM, Seattle Washington USA, 591–596. DOI:http://dx.doi.org/10.1145/3017680.3017737

Alf Inge Wang. 2011. Extensive Evaluation of Using a Game Project in a Software Architecture Course. *ACM Trans. Comput. Educ.* 11, 1 (2011), 28. DOI:http://dx.doi.org/10.1145/1921607.1921612

Gero Wedemann. 2018. Scrum as a Method of Teaching Software Architecture. In *Proceedings of the 3rd European Conference of Software Engineering Education (ECSEE'18)*. Association for Computing Machinery, New York, NY, USA, 108–112. DOI:http://dx.doi.org/10.1145/3209087.3209096

Shahani Markus Weerawarana, Amal Shehan Perera, and Vishaka Nanayakkara. 2012. Promoting creativity, innovation and engineering excellence. In *Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE) 2012*. IEEE, Wuhan, China, T1C–12–T1C–17. DOI:http://dx.doi.org/10.1109/TALE.2012.6360374

Bingyang Wei, Yihao Li, Lin Deng, and Nicholas Visalli. 2020. *Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application*. Vol. 845. Springer International Publishing, Cham, 43–54. DOI:http://dx.doi.org/10.1007/978-3-030-24344-9_3

Bian Wu and Alf Inge Wang. 2012. Comparison of Learning Software Architecture by Developing Social Applications versus Games on the Android Platform. *Int. J. Comput. Games Technol.* 2012 (2012). DOI:http://dx.doi.org/10.1155/2012/494232

Li Zhang, Yanxu Li, and Ning Ge. 2020. Exploration on theoretical and practical projects of software architecture course. In *15th International Conference on Computer Science and Education, ICCSE 2020*, Editor (Ed.). IEEE, Delft, Netherlands, 391–395. DOI:http://dx.doi.org/10.1109/ICCSE49874.2020.9201748

APPENDIX

Received May 2023; revised September 2023; accepted February 2024

2. TABLE OF SOFTWARE ARCHITECT COMPETENCIES

These competencies were obtained from a literature review, and we assigned an identifier C1, C2, C... to each one. In addition, in a workshop, we invited SA professors and industry architects, and we made a classification of the architecture into three groups. mandatory, optional and out of scope for an undergraduate course.

Creation of an Architecture:

— C1 : Identifies the relevant software quality attributes that will drive the architecture of a software system to be constructed.
— C2 : Consistently design the software architecture by defining how components interact with each other.
— C3 : Makes relevant design decisions about how a system should be built involving the choices an architect faces when designing a software system.
— C4 : EIt carefully expands the details of the design, refining it to converge in the final design.

Analysis and Evaluation of an Architecture:

— C5 : Independently evaluates a software architecture to determine functional and non-functional requirements satisfaction.
— C6 : Frequently reviews component designs proposed by junior engineers verifying compliance with the architecture.
— C7 : Systematically applies value-based architectural techniques to evaluate architectural decisions.
— C8 : Impartially performs a trade-off analysis to evaluate architectures.

Architectural Documentation:

— C9 : Organized preparation of architectural documents and presentations useful for stakeholders.
— C10 : Produces documentation standards that include variability and dynamic behavior.

Working with Existing Systems:

— C11 : Easily maintains existing systems and their architecture to evolve software systems.
— C12 : Redesigns existing architectures for migration to recent technologies and platforms.

Other Competencies:

— C13 : Proactively provides architectural guidelines for software design activities.
— C14 : Enthusiastically leads architecture improvement activities in a software development organization.
— C15 : Actively participates in defining and improving software processes in an organization.
— C16 : Reflectively defines the philosophy and principles for global architecture.
— C17 : Collaboratively provides architecture oversight support for software development projects.

Requirements Management:

— C18 : Critically analyzes functional and quality attribute software requirements.
— C19 : Understands business and customer needs quickly to ensure that requirements meet these needs.
— C20 : Systematically captures the architecture's customer, organizational, and business requirements.
— 21 : Creates clear software specifications from business requirements.

Product Implementation:

— C22 : Periodically reviews the source code written by the development team.
— C23 : Develops reusable software components.
— C24 : Develops solutions based on existing reusable components.

— C25 : Ensures compliance with coding guidelines by the development team.
— C26 : Recommends development methodologies for the development team.
— C27 : Monitors the work of consultants and external suppliers.

Product Testing:

— C28 : Establishes test procedures considering architectural aspects (types of components/services, integration).
— C29 : Builds the product by facilitating the identification and correction of faults.

Evaluation of Future Technologies:

— C30 : Explicitly evaluates enterprise software solutions and makes recommendations.
— C31 : Carefully manages introducing new software solutions in an organization.
— C32 : Objectively analyzes the current IT environment and recommends solutions for the deficiencies found.
— C33 : Develops quality technical documents and presents them to organizational stakeholders.

Selection of Tools and Technology:

— C34 : Performs reliable technical feasibility studies of recent technologies and architectures for the organization.
— C35 : Objectively evaluates commercial tools and software components from an architectural perspective.