Integrando La Formación En Arquitecturas De Software En Programas De Ingeniería De Sistemas Y Afines Con Las Necesidades De La Industria De Software

Wilson Libardo Pantoja Yépez

Tesis de Doctorado

Director:
Dr., Julio Ariel Hurtado Alegría

Co-Director:
Dr., Ajay Bandi

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Línea de investigación en Ingeniería de Software
Popayán, Diciembre de 2023

Wilson Libardo Pantoja Yépez

Integrando La Formación En Arquitecturas De Software En Programas De Ingeniería De Sistemas Y Afines Con Las Necesidades De La Industria De Software

Tesis presentada a la Facultad de Ingeniería Electrónica y Telecomunicaciones
de la Universidad del Cauca para la obtención del título de:
Doctor en Ciencias de la Electrónica

Director:
Dr., Julio Ariel Hurtado

Co-Director:
Dr., Bandi Ajay

Popayán, Colombia
2023

Dedicatoria

Siento tu ausencia en cada paso de este camino, mi amado Juan Manuel, tu luz sigue guiando mi búsqueda de conocimiento, a ti, mi eterna inspiración dedico este logro de mi vida... A mi hijo Juan Manuel quien fue y será mi inspiración y motivación, a mi sobrina María Isabel, a mi compañera Adriana, a mi madre, a mi hermana, a mi abuela, a mi tío Julio César, a mi tía Mercedes Torres, a mi familia entera, a mis amigos más cercanos que motivaron a estudiar y emprender este gran reto.

La gota de agua perfora la roca no por su fuerza, sino por su constancia, el que persevera todo lo alcanza (Ovidio).

Agradecimientos

Un agradecimiento especial a mis amigos Julio Ariel Hurtado y César Alberto Collazos por haberme motivado y animado a estudiar, quienes estuvieron pendientes del proceso, aportándome buenas ideas. Un agradecimiento especial a los profesores Andrés Solano de la Universidad Autónoma de Occidente, a Luis Mariano Bibbó y Alejandro Fernández de la Universidad de La Plata, Argentina por haber hecho posible las pasantías de investigación y sus aportes a mi tesis.

Resumen

Propósito: La capacidad de definir, evaluar e implementar arquitecturas de software es una habilidad fundamental para los ingenieros de software. Sin embargo, enseñar Arquitecturas de Software (AS) puede suponer un reto, requiere que los estudiantes participen en proyectos de contexto real con altos grados de complejidad. El propósito de este documento es presentar un catálogo de patrones de formación (reportados por la literatura y evaluados por la industria) de arquitectura de software para estudiantes de pregrado, que orienten al docente sobre el qué y el cómo enseñar, permitirán lograr significativamente el desarrollo de las competencias más relevantes para la industria de software actual y futura, relacionadas con la creación, evaluación y documentación de una arquitectura software, en potenciales egresados de programas de ingeniería de sistemas y afines.

Métodos: La primera versión del catálogo fue extraído de las estrategias recurrentes a partir de un mapeo sistemático de la literatura. Las competencias de referencia fueron establecidas a través de grupos focales con profesores y profesionales de la industria. El catálogo fue validado a través de dos estudios de caso exploratorios, un experimento controlado y un estudio de caso confirmatorio.

Resultados: Como resultado tenemos un catálogo de siete patrones de formación en AS, articulados bajo una metodología que permitiría a los docentes proponer y mejorar cursos de AS que desarrollan competencias más cercanas a las necesidades de la industria.

Conclusión: Un curso de AS acorde a las necesidades de la industria es algo esencial en los planes de estudio de programas de informática, ingeniería de sistemas y afines. Para diseñar y ejecutar un curso de AS de alta calidad, proponemos un catálogo de patrones de formación que permiten recrear los problemas y la forma de trabajo en el aula similares a los ambientes utilizados por la industria, y por otro lado, organizar el conocimiento arquitectónico para que las actividades de clase se puedan estructurar y facilitar el aprendizaje incremental.

Palabras clave: Catálogo, patrones de formación, arquitectura de software, industria, competencias.

Abstract

Purpose: The ability to define, evaluate and implement software architectures is a fundamental skill for software engineers. However, teaching Software Architectures (SA) can be challenging, requiring students to participate in real context projects with high degrees of complexity. The purpose of this paper is to present a catalog of training patterns (reported by the literature and evaluated by the industry) of software architecture for undergraduate

students to guide the teacher on what and how to teach AS. The purpose of this document is to present a catalog of training patterns (reported by the literature and evaluated by the industry) of software architecture for undergraduate students, that will guide the teacher on what and how to teach, will allow to significantly achieve the development of the most relevant competencies for the current and future software industry, related to the creation, evaluation and documentation of a software architecture, in potential graduates of systems engineering and related programs.

Methods: The first version of the catalog was extracted from the recurrent strategies based on a systematic mapping of the literature. Benchmark competencies were established through focus groups with teachers and industry professionals. The catalog was validated through two exploratory case studies, a controlled experiment and a confirmatory case study.

Results: As a result, we have a catalog of seven AS training patterns articulated under a methodology that would allow teachers to propose and improve AS courses that develop competencies closer to industry needs.

Conclusion: An AS course according to the needs of the industry is essential in the curricula of computer science, systems engineering and related programs. To design and execute a high quality AS course, we propose a catalog of training patterns that allow 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 class activities can be structured and facilitate incremental learning.

Keywords: Catalog, training patterns, software architecture, industry, competencies

Contenido

Lista de Figuras

Lista de Tablas

1. Introducción

Resumen

La capacidad de definir, evaluar e implementar arquitecturas de software es una habilidad fundamental para los ingenieros de software. Sin embargo, enseñar Arquitecturas de Software (AS) puede suponer un reto, requiere que los estudiantes participen en proyectos de contexto real con altos grados de complejidad. Esto implica tomar decisiones de compromiso entre varios atributos de calidad. Además, la percepción académica de la arquitectura de software difiere del punto de vista industrial. Este capítulo presenta una introducción al tema de investigación de la dificultad de formar en AS, describe el problema a abordar, los objetivos y la hipótesis.

1.1. Motivación

Por analogía a las disciplinas propias de la ingeniería de la construcción de edificios, Garlan y Shaw [41], introducen en el contexto de la Ingeniería de Software, el concepto de arquitectura de software, un concepto de diseño más allá de las estructuras de datos y los algoritmos. Para dar respuesta a la calidad, vista como un conjunto de propiedades visibles externamente de los sistemas de software [94], la arquitectura de software nace como un mecanismo de articulación de las cualidades en la solución desde las primeras descomposiciones del sistema [71], [33]. Los profesionales [59] también han enfatizado continuamente la importancia de la arquitectura de software para alcanzar los objetivos comerciales. En caso de falla del software, las personas y las empresas experimentan inconvenientes con consecuencias catastróficas. Por lo tanto, el software debe cumplir con las expectativas de calidad y respaldarlas siempre que el software esté operativo y orquestando los procesos de negocio [17].

Los arquitectos de software son los encargados de proponer la estructura de las aplicaciones software, de tal forma que satisfagan los atributos de calidad y restricciones del sistema, así como lograr que las aplicaciones se adapten fácilmente a los cambios del negocio. De acuerdo con Sherman and Unkelos-Shpigel [109], el rol de arquitecto es responsable del diseño y las decisiones técnicas en el proceso de desarrollo de software, tiene la función de resolver un problema definiendo las estructuras de un sistema que pueda ser implementado utilizando ciertas tecnologías. Sin embardo, encontrar el balance adecuado entre atributos del software como seguridad, desempeño, usabilidad, disponibilidad, mantenibilidad, interoperabilidad,

entre otros, es una tarea muy compleja. Estos atributos pueden entrar en conflicto unos con otros, por ello el arquitecto de software requiere conocer tácticas, patrones y principios que le ayuden a tomar adecuadamente las decisiones de diseño. Por tanto, los arquitectos deben definir sistemas aplicando conocimiento abstracto, métodos probados y un conjunto de tecnologías con el objetivo de crear una solución que responda a un balance entre las expectativas de calidad. Por lo tanto, los deberes de los arquitectos también incluyen habilidades blandas, tales como liderazgo, comunicación, el entrenamiento, entre otros [108]. Por ejemplo, un arquitecto de software debe comprender las expectativas de los interesados y comunicar las decisiones a diseñadores, programadores, testers, instaladores y demás usuarios de la especificación arquitectónica del sistema.

El rol del arquitecto es muy desafiante en cualquier proyecto de software. Un arquitecto de software podría ser una persona, equipo u organización que diseñe la arquitectura del sistema (norma IEEE 1471-2000 [52]). Un arquitecto de software comprende el proceso de desarrollo, tiene el conocimiento del dominio de negocio y, además, cuenta con las habilidades de análisis y programación. Un arquitecto es un buen comunicador, conoce las políticas de la organización y juega un papel importante en la toma de decisiones durante el proyecto. Un arquitecto desempeña el papel de un catalizador para mejorar la comunicación y desarrollar el entendimiento entre clientes y desarrolladores [100] [108]. Se considera que un arquitecto de software es un líder técnico del proyecto de software, ya que en todas las decisiones técnicas es imprescindible su participación como mediador de intereses [68].

El presente documento analiza las dificultades y desafíos de la formación de arquitecturas de software (AS) en programas ingeniería de sistemas y carreras afines y propone unos patrones de formación que ayudan a los docentes a diseñar cursos de AS alineados con las necesidades de la industria. En este capítulo se presenta el planteamiento del problema de la investigación, el objetivo general, los objetivos específicos y la hipótesis de la solución. El capítulo 2 presenta los antecedentes y algunos conceptos de AS. El capítulo 3 presenta la revisión de la literatura. El capítulo 4 contiene la Guía de Diseño de Cursos de AS (SAGITA - Software Architecture: GuIdeline for TrAining) junto al catálogo de patrones de formación. El capítulo 5 presenta la validación de SAGITA y los patrones de formación. Finalmente, el capítulo 6 contiene las conclusiones y recomendaciones.

1.2. Planteamiento del problema

Tradicionalmente, los programas de pregrado en ingeniería, como Ciencias de la Computación e Ingeniería, Tecnología de la Información, Ingeniería de Software e Ingeniería de Sistemas, incluyen muchos cursos en los cuales se desarrollan habilidades y se imparten conocimientos técnicos relacionados con la construcción de software a través del uso de lenguajes de programación y plataformas de desarrollo [64]. Sin embargo, los estudiantes de estos programas

presentan poco conocimiento sobre arquitectura de software y problemas de diseño a pesar de su creciente importancia para la industria de software [79]. Especialmente los recién graduados carecen de las habilidades suficientes en la toma adecuada de decisiones de diseño, prácticas y conocimientos relacionados al diseño de software en el contexto empresarial [79].

La literatura existente sobre la educación y la práctica de la arquitectura de software apuntan a diferentes razones para esta falta de habilidades relacionadas con la arquitectura y el diseño de software [42]. En primer lugar, existe una diferencia considerable entre la percepción académica de la arquitectura de software y su práctica en el sector industrial [69]. A veces, los problemas que las industrias consideran más críticos y desafiantes no reciben la debida importancia para la investigación o la formación en las universidades [11]. En segundo lugar, existe una brecha entre las habilidades que la industria espera de los graduados en áreas de Ingeniería de Software y las habilidades que se enseñan en los planes de estudios [4]. En tercer lugar, muchas instituciones no tienen una visión clara sobre los temas en los que se debe capacitar a los arquitectos de software [79].

Aunque existen numerosas plataformas virtuales que capacitan en entornos de desarrollo tecnológico y lenguajes de programación, no se imparten cursos que ayuden a formar el pensamiento sistémico y estratégico de un arquitecto de software [79]. Esto ha hecho que la mayoría de personas interesadas en temas de arquitecturas de software tengan que emprender un largo camino de auto estudio.

Las decisiones que se toman alrededor de las arquitecturas de software, van de la mano de temas de tecnología de información y la comunicación. Un arquitecto de software debe tener un amplio conocimiento de tecnologías para tomar decisiones sobre qué plataformas, frameworks y lenguajes favorecen ciertas cualidades y restricciones técnicas y de negocio. El problema radica en la rapidez con que estas tecnologías evolucionan. Junto con el rápido desarrollo de la tecnología de la información y la comunicación (TIC), las habilidades laborales requeridas por las industrias de las TIC también están evolucionando muy rápidamente. Se vuelve difícil para todos los actores evaluar la brecha entre sus habilidades y las habilidades requeridas en ese contexto de evolución [68]. Aunque las universidades realizan evaluaciones periódicas del plan de estudios, la brecha de tiempo entre las evaluaciones y sus actualizaciones, hace que el plan de estudios se desactualice fácilmente, ya que no puede hacer frente a los cambios acelerados y rápidos que ocurren en la industria y en el medio [4]. Además, el problema no es sólo conocer la brecha, sino trabajar en estrategias concretas para reducirla.

A través de una revisión se buscó indagar a fondo las razones de la complejidad asociada a la enseñanza de la arquitectura de software. Como resultado, se encontraron varias causas de la dificultad que involucra la formación en temas de arquitectura de software a nivel de pregrado (ver Figura **1-1**) [87]. Estas causas las clasificamos por categorías definidas a través de la lectura de los resúmenes: (1) Industria de software, (2) Contenidos curriculares, (3) Docentes, (4) Naturaleza de las arquitecturas, (5) Recursos y (6) Estudiantes. Algunas causas podrían

encajar en más de una categoría, nosotros la ubicamos en aquella que a nuestro juicio tuviera mayor relevancia en la categoría. A continuación se describen las causas encontradas.

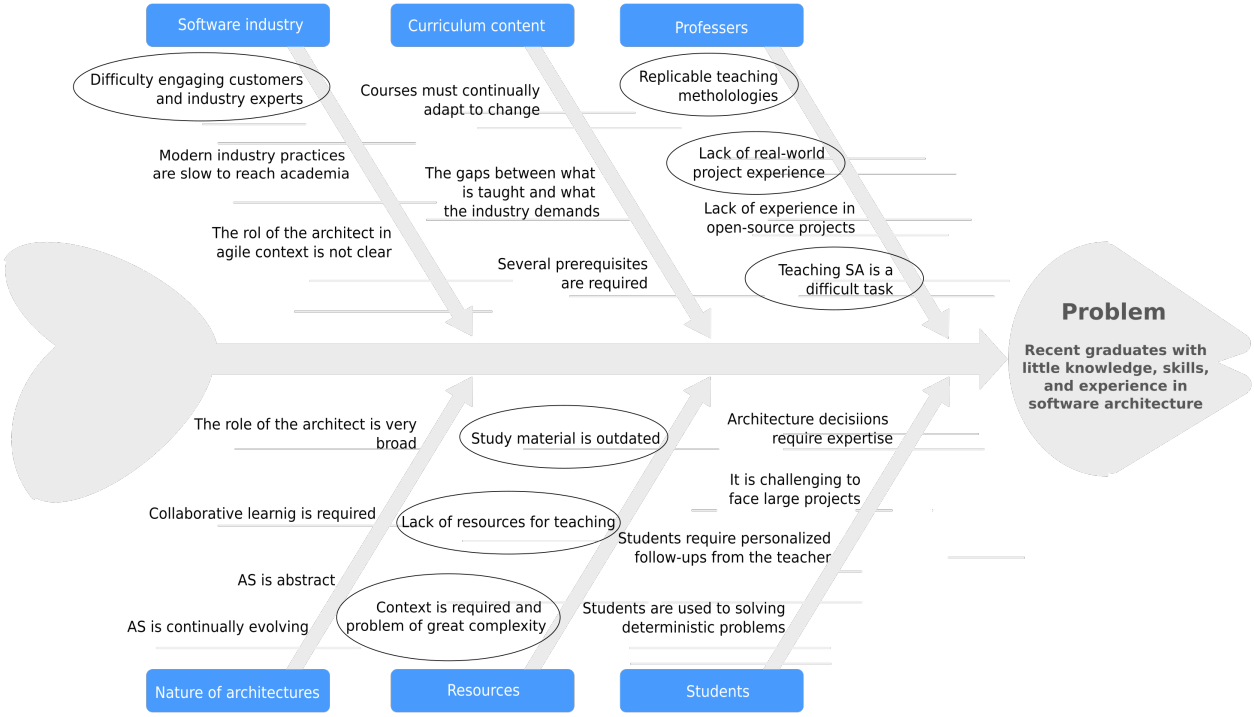

Figura 1-1.: Diagrama de espina de pescado, causas y efectos.

Industria de software:

1. Hay dificultad para involucrar clientes y expertos de la industria en los ambientes académicos. Este tipo de personas siempre están ocupadas [29].
2. Las prácticas modernas se desarrollan en la industria y tardan en llegar a la academia. La enseñanza en las aulas puede basarse en prácticas obsoletas o prácticas que muy pronto lo serán [116].
3. En metodologías ágiles, las más usadas hoy día por la industria, no está claro: i) Cuándo crear la arquitectura, ii) cuál es el rol del arquitecto, c) cómo debe ser la documentación de la arquitectura, iii) los métodos y las actividades, e) el valor y los costos [9].

Contenidos curriculares:

1. Los cursos de arquitectura de software se deben adaptar continuamente a los cambios en los requisitos, a los cambios en la industria, los avances tecnológicos o las expectativas de los estudiantes [73].
2. Existe una brecha de conocimiento entre la teoría aprendida en el aula y los requisitos reales exigidos por la industria [57].

3. Las actividades en cada etapa del diseño de arquitectura de software involucran integrar muchos conocimientos aprendidos en otros cursos (prerrequisitos) [65], como redes, bases de datos, sistemas operativos, sistemas distribuidos, ingeniería de software, etc. Si el estudiante no tiene buenas bases en estas áreas, no podría aprovechar adecuadamente un curso de arquitectura de software.

Docentes:

1. Los antecedentes de los docentes deben incluir una gran cantidad de experiencia diversa del mundo real. Desafortunadamente, muchos profesores siempre han sido académicos y no tienen experiencia en el diseño de productos reales con desafíos arquitectónico [104].
2. Los repositorios open-source brindan muchas ventajas en el aprendizaje de arquitecturas de software, pero la mayoría de docentes no tienen experiencia en este tipo de proyectos y productos [110] [50]. Por lo tanto, para los docentes es muy difícil trabajar en este tipo de escenarios.
3. Se requiere un gran esfuerzo para lograr un seguimiento personalizado de los estudiantes, y en especial cuando los cursos son muy grandes [116]. Por lo general, la cantidad de estudiantes en un curso de este tipo es elevada.
4. Los educadores deben crear un equilibrio entre la calidad, el alcance, la profundidad, la aplicabilidad, las habilidades blandas y duras, el aprendizaje individual y colaborativo en sus enseñanzas. Se requiere un contexto realista, trabajo en equipo, suficiente complejidad y entrenamiento práctico [86] [1] [9] [124]. Esto hace que planear y desarrollar un curso de arquitectura es una labor bastante compleja.

Naturaleza de las arquitecturas:

1. Enseñar arquitecturas de software va más allá de la forma tradicional de enseñar, se debe poner atención a la complejidad de las interacciones sociales, en particular, discutir cómo se produce el desarrollo de software en forma colaborativa y en un entorno del mundo real. Hay obstáculos para trabajar con aprendizaje colaborativo: (1) la resistencia al cambio de paradigma que requieren estudiantes e instructores, (2) contar con un diseño adecuado de la actividad colaborativa a aplicar, y (3) tener una buena solución tecnológica para apoyar la actividad (especialmente en entornos distribuidos) [97] [96] [99] [118] [85].
2. No sirven proyectos de "juguete", para proporcionar una experiencia práctica real, la enseñanza de actividades de arquitectura requiere un contexto adecuado y problemas de complejidad suficiente (es decir, lo suficientemente grandes involucrando varios atributos de calidad) [97] [20] [110] [39] [60].

Recursos:

1. Existe poco material sobre arquitectura de software disponible, la mayoría de recursos están desactualizados y su calidad no está asegurada [39].
2. No hay recursos disponibles de problemas con un contexto adecuado y una complejidad razonable que proporcionen al estudiante una experiencia práctica real [97] [20] [110] [39] [60] [60]

Estudiantes:

1. Quien esté acostumbrado a escribir código y compilar para obtener un resultado determinista debe cambiar la forma de pensar. Ahora, debe estructurar los componentes juntos en un diagrama, justificando sus decisiones esenciales con posibles compensaciones sin necesariamente escribir código. Además, el contexto de aprendizaje varía según las experiencias y antecedentes del estudiante, lo que hace que las soluciones de arquitectura de software desarrolladas por ellos sean demasiado abstractas y difusas [66] [128] [67] [115] [118] [68] [65] [60].
2. Los escenarios de proyectos finalizados son ventajosos en la formación de arquitectura de software, pero tienden a intimidar a los estudiantes para que realicen cambios. Por esto, los estudiantes prefieren proyectos completamente nuevos. Este segundo escenario es más fácil para los estudiantes porque comprenden los requisitos y tienen un control total sobre la arquitectura, el diseño y la estructura del sistema de software [123]. Sin embargo, esto no siempre corresponde con la realidad de la industria, donde un número significativo de proyectos son de mantenimiento y evolución que no inician de cero.

Por otro lado, tenemos los efectos del problema. Por razones prácticas, hemos considerado los efectos directos sobre los recién egresados de las universidades identificados por la consultora Arquisoft90: Entrenamiento de Arquitectura de Software [79].

1. Los recién egresados tienen pocos conocimientos, habilidades y experiencia en temas de arquitectura de software.
2. Los recién egresados que se enfrentan a sistemas en producción, no son capaces de tomar decisiones idóneas alrededor de las arquitecturas de software ni aplicar prácticas, patrones, ni principios de manera adecuada.
3. No está claro qué actividades debe desarrollar el arquitecto de software durante el proceso de desarrollo.
4. Las Universidades pierden credibilidad cuando sus egresados no están al nivel que demanda la industria de software.
5. Los egresados pierden oportunidades laborales, y además, deben invertir más tiempo y recursos en formarse como arquitectos de software.
6. La industria de software pierde competitividad.

Además, las empresas buscan incorporar el rol de arquitecto de software a sus equipos de desarrollo, pero este rol es aún difuso [100] [108].

Debido a que el problema es complejo y tienen muchas causas, en este investigación se decidió abordar aquellas causas que tenga mayor impacto en el problema. Según la regla de Paretto, sostiene que aproximadamente el 80 % de los problemas se derivan del 20 % de las causas. De esta manera, en la investigación se aborda una solución que involucre las causas encerradas en óvalos tal como lo muestra la Figura **1-1**. De esta forma se toman principalmente las causas de las categorías: recursos, docentes y una causa de la industria de software.

1.3. Objetivos

1.3.1. Objetivo general

- Establecer un catálogo de patrones de formación para que los profesores logren, en los potenciales egresados de programas de ingeniería de sistemas y afines, el desarrollo de las competencias más relevantes para la industria de software relacionadas con la creación, evaluación y documentación de arquitecturas de software.

1.3.2. Objetivos específicos

- Caracterizar las necesidades y estrategias de formación del rol de arquitecto de software acorde a las necesidades de la industria para entender las problemáticas de formación e identificar las formas comunes en la que las universidades las resuelven.
- Documentar un catálogo de patrones de formación de ingenieros de sistemas y afines a nivel de pregrado, que permitan abordar las causas de mayor impacto sobre el problema y su viabilidad, para ser corregidas a través de un trabajo integrado entre la academia e industria.
- Evaluar iterativamente la efectividad de los patrones de formación presentes en el catálogo a través de un enfoque de investigación-acción que involucre varias universidades y empresas de la región del sur-occidente colombiano.

1.4. Hipótesis de la solución

Teniendo en cuenta el problema de las dificultades en la formación de arquitecturas de software en programas de pregrado, sus causas y posibles efectos, nos hemos planteado la

siguiente pregunta de investigación:

¿Cómo lograr adecuadamente el desarrollo de competencias relacionadas con la creación, evaluación y documentación de arquitecturas de software en potenciales egresados de programas de ingeniería de sistemas y afines, mediante una guía de diseño y desarrollo de cursos basada en patrones de formación?

De esta forma planteamos la siguiente hipótesis.

Hipótesis: Un catálogo de patrones de formación (reportados por la literatura y evaluados por la industria) de arquitectura de software para estudiantes de pregrado, que orienten al docente sobre el qué y el cómo enseñar, permitirán lograr significativamente el desarrollo de las competencias más relevantes para la industria de software actual y futura, relacionadas con la creación, evaluación y documentación de una arquitectura software,en potenciales egresados de programas de ingeniería de sistemas y afines.

Hipótesis nula: Un catálogo de patrones de formación (reportados por la literatura y evaluados por la industria) de arquitectura de software para estudiantes de pregrado, que orienten al docente sobre el qué y el cómo enseñar, no es suficiente para lograr significativamente el desarrollo de las competencias más relevantes para la industria de software actual y futura, relacionadas con la creación, evaluación y documentación de una arquitectura software,en potenciales egresados de programas de ingeniería de software sistemas y afines.‘

Para los **patrones de formación** se definirá/adecuará un lenguaje descriptivo que potencialmente incluirán *escenarios*, *estrategias de formación*, *recursos*, *competencias previas*, *competencias previas*, *planes de desarrollo ejemplo*, entre otros. Un escenario tendrá en cuenta el contexto de cada curso, por ejemplo, la intensidad horaria, los requisitos que traen los estudiantes, la tecnología, las personas, profesores, la formación de los profesores, etc. A su vez, cada escenario involucra unas estrategias de formación. Las estrategias deben orientar al docente en qué enseñar y cómo enseñar. Qué enseñar no necesariamente involucra el contenido a detalle, pero si los temas gruesos y el orden en que se deben dictar según el escenario. El cómo enseñar involucra la combinación de varias estrategias que acercan el curso a las realidades del mundo laboral, por ejemplo: un proyecto de curso, conferencistas del sector de la industria, blogs de arquitectura, resolución de problemas de arquitectura, entre otros. Alrededor de éstos aspectos el patrón detallará la información necesaria para que un profesor pueda llevar a la práctica su curso de formación siguiendo el enfoque dado por el patrón de formación establecido.

2. Antecedentes y definición de conceptos

Resumen

Teniendo en cuenta la gran relevancia de la AS para la industria, muchas universidades e institutos académicos han introducido un curso sobre Diseño y Arquitectura de Software como parte de los programas de informática e ingeniería de sistemas. Sin embargo, los profesores que imparten este curso se enfrentan a numerosos retos. Algunos de estos retos se derivan de cómo se practica la AS en las industrias y otros de la enseñanza de AS en un entorno académico. Esta sección brinda unos antecedentes de la investigación y profundizamos en varios conceptos de arquitectura de software que aparecen con frecuencia en las prácticas industriales y en los cursos. Estos conceptos incluyen la definición de AS, los atributos de calidad, los patrones de arquitectura y las tácticas arquitectónicas. Estas ideas recurrentes se han identificado y definido debido a su aparición regular en la literatura. Reconocer y comprender estos conceptos es esencial a la hora de diseñar cursos de formación en arquitectura de software. También se da una definición de competencia y su relación con las habilidades y el conocimiento.

2.1. Arquitectura de Software

La arquitectura de software se define como una estructura o estructuras de un sistema, que organizan los elementos de software y sus relaciones [25]. El énfasis está en las propiedades visibles externamente de las piezas de software y cómo se relacionan. La arquitectura de software proporciona un mecanismo para el establecimiento, la documentación y la comunicación de las principales decisiones de diseño de un sistema. Por lo tanto, la arquitectura de software es un concepto significativo de múltiples maneras.

Algunas de las características de la arquitectura de software que se mencionan con frecuencia son [13]: (i) Es una abstracción primaria del sistema que: los involucrados usan para pensar, diseñar, codificar y comunicarse en términos de grandes bloques conceptuales, (ii) promueve la reutilización de alto nivel y la reutilización de componentes, (iii) influye en la productividad del desarrollo al reutilizar grandes marcos para respaldar la construcción de líneas de productos, (iv) asegura la calidad a lo largo del ciclo de vida del software al tratar explí-

citamente los atributos de calidad como la modificabilidad, la portabilidad, escalabilidad y seguridad. A medida que crece el tamaño del sistema, los desarrolladores pueden rastrear las soluciones a estos problemas mediante el análisis de la arquitectura.

El concepto de Arquitectura de Software ha venido evolucionando continuamente lo que ha hecho que existan distintas definiciones. Angelov and de Beer [9] hablan de la semántica del término Arquitectura de Software y analizan tres enfoques de definiciones:

1. Buschmann y Henney afirman que una arquitectura de software "puede significar cualquier cosa, desde un *esbozo de diseño de alto nivel* con poca relación con la tecnología, el código o el sistema real que se está construyendo, hasta un diseño inicial grande y rígido con muchas minucias de clase y nivel de código" [14]. Señalan que, en esencia, una arquitectura "es un servicio para guiar y desarrollar un sistema de software hacia el cumplimiento de un conjunto de objetivos comerciales y técnicos, expresados en la forma que mejor ayude a la comunicación y el intercambio".

2. Algunas definiciones se enfocan en el aspecto *estructura* de las arquitecturas. Por ejemplo, según Bas et al., la arquitectura del software es "el conjunto de estructuras necesarias para razonar sobre el sistema, que comprende los elementos del software, la relación entre ellos y las propiedades de ambos". [13]. Según IEEE 1471-2000, "la arquitectura de software es la organización fundamental de un sistema incorporada en sus componentes, sus relaciones entre sí y con el entorno, y los principios que guían su diseño y evolución".

3. Otras definiciones ven a la *decisión de diseño* como el foco de las arquitecturas de software [53]. Lago y Van Vliet afirman que una arquitectura de software refleja "las principales decisiones de diseño que se toman" [62], y de Boer y van Vliet que "la estructura real o el diseño arquitectónico es simplemente un reflejo de esas decisiones de diseño" [32]. Fowler considera que las principales decisiones de diseño son decisiones que son difíciles de cambiar, por ejemplo, decisiones relacionadas con la elección de plataformas tecnológicas o la elección del estilo de arquitectura [38]. Según Zimmermann, las decisiones arquitectónicas "capturan los problemas de diseño clave y la lógica detrás de las soluciones elegidas" [129]. Sin embargo, Bass et al. argumentan que en los proyectos ágiles, algunas decisiones se toman más tarde a lo largo del proyecto y es difícil justificar si una decisión es importante [13]. Abrahamsson et al., señalan que las decisiones arquitectónicas no deben mezclarse con las demás decisiones de diseño de software [3].

Algunos arquitectos se refieren a la arquitectura de software como el *blueprint* del sistema, mientras que otros la definen como la *hoja de ruta* para desarrollar un sistema [100]. La Figura **2-1** ilustra una definición, la arquitectura de software consiste en la *estructura* del sistema (indicada como las líneas negras gruesas que soportan la arquitectura), combinada con **características de la arquitectura** ("ilities") el sistema debe soportar *decisiones de*

arquitectura y, finalmente, principios de diseño [100]. La estructura del sistema se refiere al tipo de estilo (o estilos) de arquitectura implementado (como microservicios, capas o microkernel). Las características de la arquitectura definen los criterios de éxito de un sistema, que generalmente es ortogonal a la funcionalidad del sistema. Las decisiones de arquitectura definen las reglas sobre cómo debe construirse un sistema. Un principio de diseño difiere de una decisión de arquitectura en que un principio de diseño es una guía en lugar de una regla estricta.

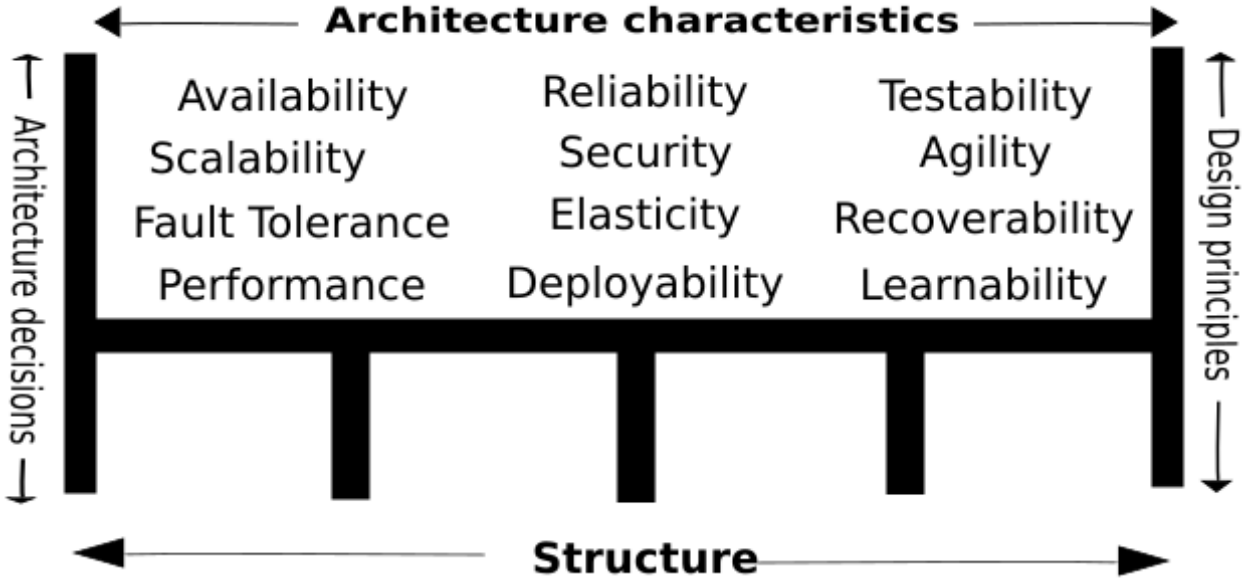

Figura 2-1.: La arquitectura de software consiste en la *estructura* del sistema, combinada con **características de la arquitectura** ("ilities") que el sistema debe admitir, *decisiones de la arquitectura* y, finalmente, los principios de diseño [100] .

2.2. Atributos de calidad

Los atributos de calidad son las características que debe satisfacer un producto de software. Cada atributo de calidad está asociado con métricas específicas que definen los niveles de calidad de un producto de software [105]. Algunos ejemplos de atributos de calidad incluyen seguridad, confiabilidad, modificabilidad, rendimiento, interoperabilidad y otros. Estos atributos surgen de reglas comerciales altamente complejas y problemas de calidad [35]. El manejo inadecuado de estos atributos de calidad plantea riesgos técnicos y comerciales. El arquitecto debe considerar los conflictos entre los atributos de calidad conocidos como compensaciones arquitectónicas. Un producto de software debe cumplir con varias cualidades relevantes, y las decisiones tomadas para satisfacer un atributo de calidad pueden afectar a otro atributo de calidad. Por ejemplo, diseñar sistemas de software para alta disponibilidad

puede afectar la seguridad, pues para dar mayor disponibilidad se necesita aumentar la redundancia de elementos, agregando más puntos de vulnerabilidad a la solución, afectando así la seguridad.

2.3. Patrones de arquitectura

Los patrones de arquitectura (o estilos de arquitectura) son estructuras de solución comunes a un problema de diseño similar que se entienden y documentan bien [47]. Cada patrón describe la estructura de un sistema de software general o el comportamiento de alto nivel desde una perspectiva y está destinado a satisfacer la funcionalidad, las cualidades y las restricciones de un producto de software. Por ejemplo, el patrón de capas describe una estructura desde la perspectiva de módulos de desarrollo, mientras que el patrón publicador/subscriptor describe una estructura desde la perspectiva de la interacción de componentes a través de conectores. Los patrones de arquitectura se eligen en respuesta a las primeras decisiones de diseño, incluidas las decisiones sobre la satisfacción de los requisitos funcionales, los requisitos no funcionales o los atributos de calidad, y las restricciones físicas, como la distancia física entre el usuario y el proveedor de servicios. La aplicación de un patrón generalmente se documenta como consecuencias, en las que se indican los aspectos positivos y negativos de la solución.

Muchos de los beneficios y responsabilidades se relacionan con el logro de los atributos de calidad. Por ejemplo, una ventaja del patrón de capas indica que las interfaces estandarizadas entre capas generalmente limitan los cambios de código en la capa que se cambia, lo que facilita la modificación del sistema.

2.4. Tácticas de arquitectura

Las tácticas arquitectónicas son abstracciones de alto nivel que capturan las decisiones tomadas para lograr objetivos de calidad. Las tácticas pueden ser en tiempo de diseño, como *ocultar información* para mejorar la modificabilidad, o pueden ser en tiempo de ejecución, como *administrar la concurrencia* para mejorar el rendimiento [47]. Las tácticas arquitectónicas impactan el diseño de software de varias maneras. En algunos casos, un desarrollador puede implementar rápidamente una táctica a través de un patrón de arquitectura particular.

Por ejemplo, el patrón *capas* es una forma de implementar la táctica de modificabilidad, que *restringe las rutas de comunicación*. Por otro lado, una táctica puede requerir cambios significativos en la estructura y comportamiento del patrón o necesitar estructuras y comportamiento completamente nuevos. En tales casos, la implementación de tácticas y el mantenimiento futuro de un sistema serán más complejos y tediosos.

2.5. Decisiones de arquitectura

Las decisiones de arquitectura definen las reglas sobre cómo debe construirse un sistema. Las decisiones de arquitectura generalmente involucran las elecciones que enfrenta un arquitecto al diseñar un sistema de software. Además, proporciona una selección realizada por el diseñador en un contexto específico junto con su justificación o fundamento detrás de la selección de la opción. Las opciones pueden ser sobre la estructura de la aplicación o el sistema, la selección de una tecnología para implementar el diseño o una compensación entre los atributos de calidad. Independientemente del contexto, una decisión de arquitectura ejemplar ayuda a los equipos de desarrollo a tomar las decisiones técnicas correctas [100]. Por lo tanto, las decisiones de arquitectura deben explicarse en términos de las siguientes características:

- Alternativas de diseño disponibles.
- Justificación de la decisión.
- Documentación de la decisión.
- Comunicación efectiva de esa decisión a las partes interesadas interesadas (stakeholders).

2.6. Competencias, habilidades y conocimientos

Una *competencia* se define como la *habilidad* para hacer algo [96]. El *conocimiento* puede ser entendido como el entendimiento teórico o práctico. Para un individuo, la competencia se compone de conocimientos y habilidades.

Una definición de competencia ampliamente aceptada en diversos contextos educativos y laborales es "la capacidad, habilidad o conjunto de conocimientos y destrezas que una persona posee para llevar a cabo una tarea o función específica de manera efectiva y con éxito". Implica la combinación de conocimientos teóricos, habilidades prácticas, actitudes y comportamientos necesarios para enfrentar desafíos y alcanzar objetivos en un contexto particular. Las competencias pueden ser adquiridas y desarrolladas a través de la educación, la formación y la experiencia laboral.

Según Bass en su libro Software Architecture in Practice [13], define deberes, habilidades y conocimientos. Los deberes, las habilidades y el conocimiento forman una tríada sobre la cual se apoyan las competencias arquitectónicas de los ingenieros. Las habilidades y conocimientos soportan la ejecución de las competencias tal como lo muestra la Figura **2-2**. Un ejemplo de estos tres conceptos se muestra a continuación:

- "Diseñar una arquitectura" es un deber (duty).

- "Pensar de manera abstracta" es una habilidad (skill).
- "Patrones y tácticas" es parte de un cuerpo de conocimiento (knowledge).

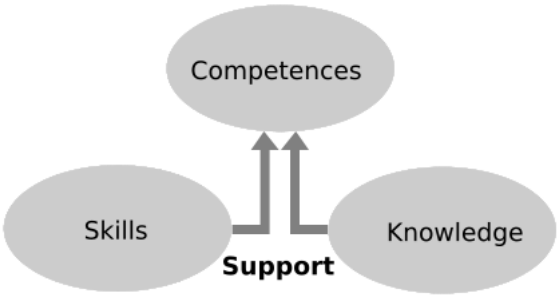

Figura 2-2.: Las habilidades y conocimientos soportan la ejecución de las competencias.

Como parte de esta investigación se obtuvo un listado de competencias en AS obtenidas de la revisión bibliográfica y analizadas por docentes e ingenieros de la industria de software (ver Apéndice C). Estas competencias están clasificadas en tres grupos: obligatorias, opcionales y fuera de alcance para un curso de AS de pregrado.

3. Revisión de la literatura sobre la formación en AS

Resumen

Esta sección describe el proceso y los resultados de la revisión de la literatura llevada a cabo para identificar y analizar las estrategias, los retos y las experiencias de los cursos utilizados para enseñar Arquitectura de Software. El estudio analizó 56 artículos primarios relacionados con la formación en arquitectura de software. Esta revisión apunta directamente al objetivo específico 1 de la presente investigación: "Caracterizar las necesidades y estrategias de formación del rol de arquitecto de software acorde a las necesidades de la industria para entender las problemáticas de formación e identificar las formas comunes en la que las universidades las resuelven".

3.1. El proceso de la Revisión de la Literatura

La revisión de la literatura siguió el método de estudio de mapeo propuesto por Petersen [95], que describe cómo llevar a cabo un estudio de mapeo sistemático en ingeniería de software. El método organiza sistemáticamente los conocimientos actuales en categorías, clasificaciones o taxonomías y resume sus resultados en relación con las preguntas de investigación. Este mapeo sistemático fue publicado en el artículo "Training Software Architects Suiting Software Industry Needs: A Literature Review" [88].

Las etapas esenciales del proceso del estudio de mapeo sistemático son las siguientes (véase Figura **3-1**): (1) definición de las preguntas de investigación, (2) búsqueda de estudios primarios, (3) revisión del texto completo de los artículos según criterios de inclusión/exclusión, (4) clasificación de documentos, y (5) extracción y mapeo de los datos recopilados. Cada etapa produce un resultado que se utiliza como entrada para las etapas siguientes.

Las siguientes subsecciones presentan el proceso de preparación del estudio de mapeo. Siguiendo estos pasos, podemos organizar y analizar sistemáticamente los conocimientos actuales en la enseñanza de la ingeniería de software e identificar estrategias para alinear la enseñanza de la arquitectura de software con las necesidades de la industria.

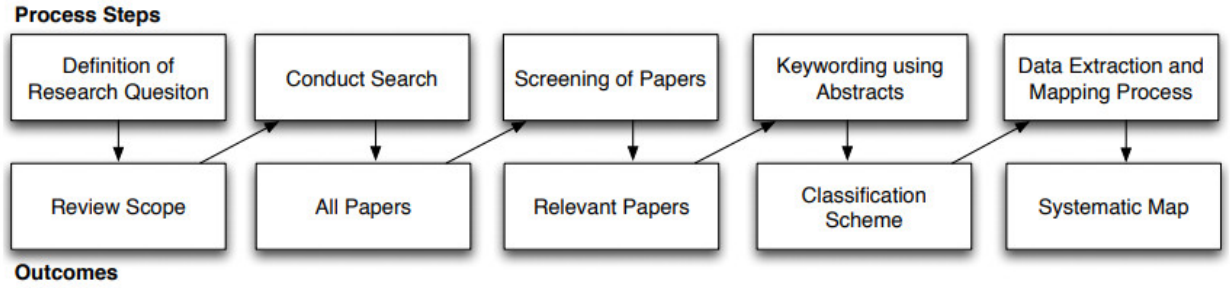

Figura 3-1.: Proceso para realizar el Mapeo Sistemático según Petersen et al. [95]

3.1.1. Preguntas de investigación

Las preguntas de investigación de este estudio pretenden identificar los vacíos en la investigación sobre formación en arquitectura de software, incluido el proceso de formación seguido por las universidades, sus estrategias de aprendizaje y el contenido. Las preguntas de investigación son las siguientes:

1. RQ1: ¿Cómo incorporan los cursos de AS técnicas de formación y metodologías de enseñanza para satisfacer las necesidades de la industria del software?
2. RQ2: ¿Qué contenidos de AS se incluyen en los cursos universitarios?
3. RQ3: ¿Cuáles son las habilidades clave que un estudiante en arquitectura de software debe desarrollar durante su formación?
4. RQ4: ¿Cuáles son los retos a los que se enfrentan los estudiantes durante la formación en arquitectura de software?
5. RQ5: ¿Qué métodos se han utilizado para validar las experiencias de formación y alcanzar los objetivos propuestos?

La Tabla **3-1** presenta las preguntas y su motivación. Al abordar estas preguntas de investigación, podemos seleccionar, analizar y categorizar la información encontrada en el área de estudio. Este enfoque nos permite identificar las áreas donde se pueden hacer mejoras en la educación de arquitectura de software para preparar de manera adecuada a los estudiantes para el éxito futuro en la industria del software.

3.1.2. Estrategia de búsqueda

Un aspecto fundamental de la realización de un estudio de mapeo sistemático es garantizar que las preguntas de investigación estén bien construidas. El método PICOC ayuda a describir los cinco elementos de una pregunta de investigación y significa Población, Intervención,

Tabla 3-1.: Preguntas de investigación utilizadas en el mapeo sistemático.

Pregunta	Motivación	Posibles respuestas
RQ1	Determinar las estrategias utilizadas para formar a los futuros arquitectos de software.	Trabajo en colaboración, proyectos de código abierto, aprendizaje basado en casos, juegos, aprendizaje basado en proyectos, estudios de casos,
RQ2	Determine los temas y los resultados del aprendizaje que abordan los cursos encontrados.	Contenidos de los cursos y resultados del aprendizaje.
RQ3	Determine cuáles son las competencias técnicas y blandas más esenciales que necesita un arquitecto de software para desempeñarse adecuadamente en el sector industrial.	Liderazgo, pensamiento crítico, comunicación y trabajo en equipo, pensamiento de diseño y abstracción, entre otras.
RQ4	Determinar los problemas a los que se enfrentan los estudiantes y que se reportan en la bibliografía sobre AS.	Falta de experiencia, complejidad del tema, etc.
RQ5	Evaluar cómo se ha validado la idoneidad del curso propuesto.	Estudio de casos, experimento, investigación-acción, etc.

Comparación, Resultado y Contexto:

- *Población*: Trabajos que reportan estudios sobre la enseñanza de las AS.
- *Intervención*: Estrategias usadas en estos estudios reportados que facilitan la enseñanza de las AS.
- *Comparación*: Experiencia directa del fenómeno a través del curso de ingeniería de software II de la Universidad del Cauca.
- *Resultado*: se espera poder clasificar las estrategias de enseñanza de acuerdo a las preguntas planteadas.
- *Contexto*: Enseñar AS es una tarea compleja para un docente. Varios investigadores reportan en sus artículos estrategias, métodos y técnicas que han usado para enseñar AS y desarrollar las habilidades que serán demandadas en la industria de software.

Para recopilar los datos de las bases de datos bibliográficas de forma sistemática, se utilizó

la siguiente cadena de búsqueda basada en los criterios PICOC [114]:

("educational needs" OR "knowledge needs" OR "desired skills" OR "essential competencies" OR "knowledge requirements" OR "skill requirements" OR "training" OR "teaching") AND ("software architectures" OR "Software Architecture" OR "software design") AND ("industrial software" OR "software industry") AND "course"

Aplicamos esta consulta al título, resumen y palabras clave de seis bases de datos bibliográficas: ACM Digital Library, EBSCO Services, IEEE Digital Library, Scopus, ScienceDirect y Google Scholar [102]. Aunque EBSCO no es una base de datos relacionada con la tecnología, la incluimos para incorporar artículos relevantes de fuentes fiables. Ajustamos la consulta a la sintaxis de cada buscador y la aplicamos a estudios publicados desde enero de 2011 hasta marzo de 2021.

3.1.3. Criterios de selección de los estudios primarios

Siguiendo las directrices propuestas por Petersen et al. [95], empleamos los siguientes criterios de inclusión y exclusión para identificar los estudios más relevantes para nuestra investigación y responder a las preguntas de investigación.

Criterios de inclusión: Los criterios de inclusión empleados para identificar estudios relevantes para nuestra investigación y responder a las preguntas de investigación fueron los siguientes:

1. Los artículos están escritos en inglés y se centran en la formación de los arquitectos de software para satisfacer las necesidades de la industria.
2. Los artículos son publicaciones de texto completo en revistas revisadas por pares, conferencias o simposios.
3. Los artículos se basan en datos empíricos y no únicamente en las opiniones de los autores.
4. Los lectores pueden acceder al texto completo del artículo a través de una fuente fiable.

Criterios de exclusión:

1. El estudio principal no presenta una solución que contribuya a que la formación de los arquitectos de software alineada con las necesidades de la industria.
2. Estudios no disponibles en texto completo.
3. Trabajos que presentan material no revisado por pares.

4. Estudios secundarios o terciarios que informan y analizan los resultados de otras investigaciones.
5. Publicaciones duplicadas del mismo autor.

Se consideró un estudio si cumplía cada criterio de inclusión y no cumplía ninguno de los criterios de exclusión.

3.1.4. Fase de ejecución

En nuestro estudio aplicamos la cadena de búsqueda a cinco bases de datos bibliográficas y obtuvimos 673 resultados. A continuación, se aplicaron los criterios de inclusión y exclusión a los resúmenes, las palabras clave y los títulos de los artículos. Después de dos rondas de revisión se obtuvieron 56 "estudios primarios candidatos". En la siguiente fase, se extrajeron los datos, se caracterizaron los estudios y se respondió a las preguntas de investigación propuestas.

Entre 2011 y 2021, se reportaron 56 experiencias de formación en AS que cumplieron con los criterios establecidos en esta revisión. Estas experiencias tenían como objetivo salvar la brecha entre la educación universitaria y las necesidades de la industria del software. Clasificamos estas experiencias en tres grupos: (i) experiencias centradas únicamente en la formación en arquitectura de software (37 %); (ii) experiencias centradas en la formación en ingeniería de software que incluían AS como uno de los temas particulares (57 %); y (iii) experiencias complementarias que utilizaban estrategias para apoyar los patrones de software y la formación en arquitectura (9 %). Figura **3-2** representa el número de experiencias en cada uno de estos tres grupos.

El Apéndice A muestra los artículos primarios encontrados.

El estudio identificó cuatro niveles de formación en arquitectura de software: pregrado, postgrado, formación continua y formación industrial. La formación de pregrado se refiere a los programas universitarios de informática, ingeniería de software y campos afines. La formación de posgrado se refiere a los programas de maestría, mientras que la formación continua se refiere a la enseñanza universitaria que pretende vincularse con el entorno a través de cursos, talleres y diplomados. La formación en la industria se refiere a la formación proporcionada por las empresas a sus empleados a través de cursos privados destinados a mejorar sus habilidades. Figura **3-3** representa la distribución de las experiencias en cada nivel de formación; la mayoría de los casos reportados corresponden a cursos de pregrado (39 experiencias, 70 %).

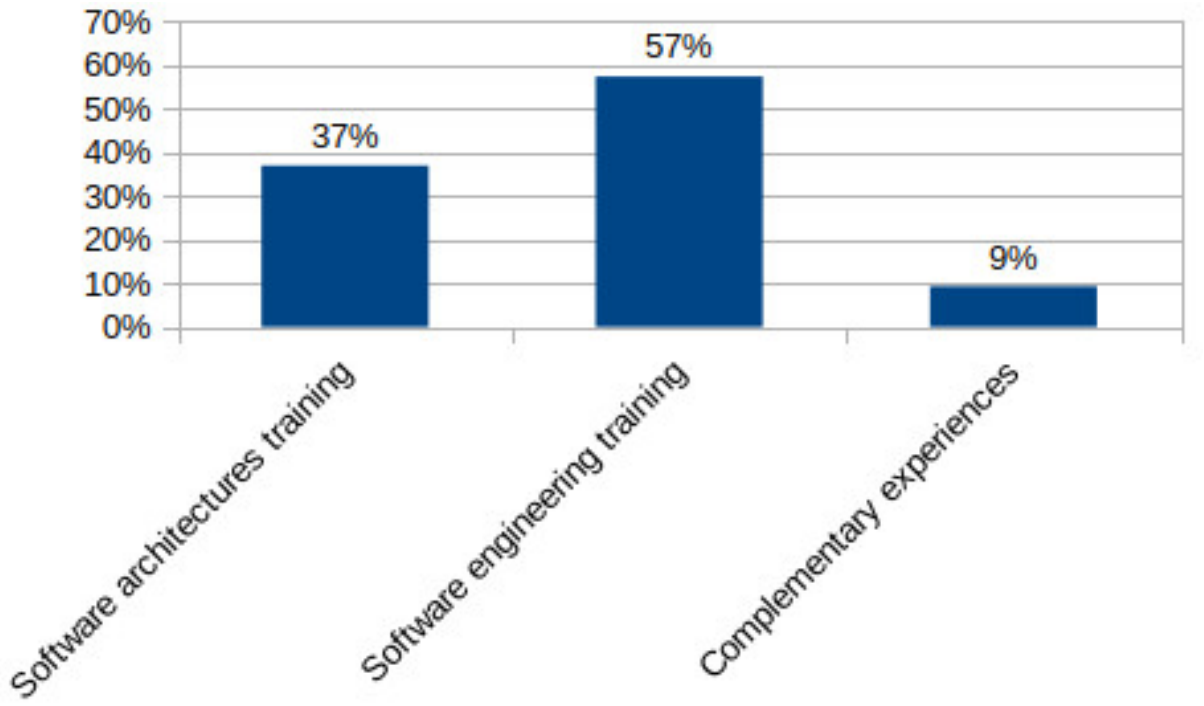

Figura 3-2.: Experiencias que soportan temas de formación en arquitectura de software.

3.2. Resultados y análisis de datos

La Figura **3-4** presenta la distribución anual de publicaciones sobre formación en arquitectura de software alineadas a las necesidades de la industria del software. En los últimos años, se ha producido un aumento significativo de las publicaciones, con el mayor número entre 2018 y 2020 (8 por año, con un total de 25, lo que representa el 44 %). Esta tendencia indica que la enseñanza de la AS es un área de interés para la comunidad investigadora. El año 2014 tuvo el menor número de publicaciones, con un solo artículo publicado. En 2021, sólo hay una publicación porque la recogida de datos se realizó hasta marzo.

A continuación vamos responder las preguntas de investigación planteadas anteriormente.

3.2.1. RQ1: ¿Cómo incorporan los cursos de arquitectura de software técnicas de formación y metodologías de enseñanza para satisfacer las necesidades de la industria del software?

La revisión de la literatura revela diversas estrategias de enseñanza para alinear los cursos de formación en arquitectura de software con las necesidades de la industria del software. Estas técnicas pueden clasificarse en dos categorías: las utilizadas para la formación de

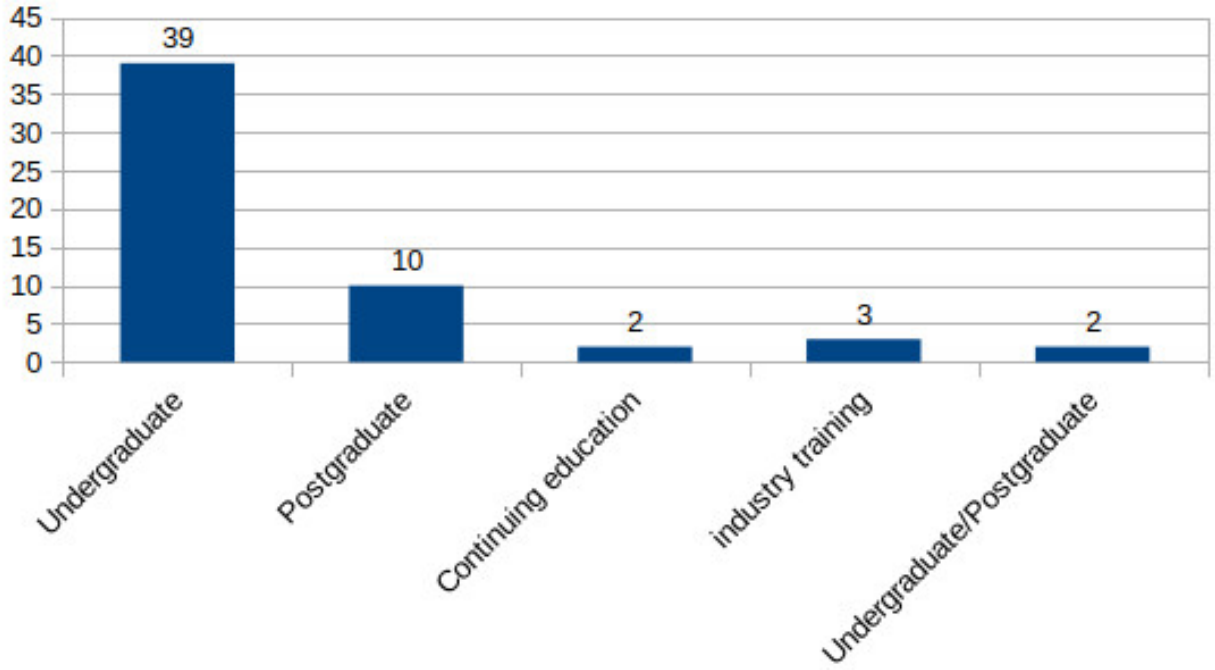

Figura 3-3.: Reporte de experiencias en cada nivel de formación.

arquitecturas de software y las utilizadas como metodologías de enseñanza y aprendizaje. Las primeras se refieren a las estrategias para recrear el entorno de trabajo de la industria en el aula, mientras que las segundas se refieren a las estrategias de enseñanza que utilizan los profesores para impartir los cursos de arquitectura de software. Los artículos Tabla **3-2** y Tabla **3-3** ofrecen una síntesis de estas estrategias.

En cuanto a las técnicas de formación, se han identificado varias estrategias. Una primera estrategia es la **formación basada en mini-proyectos**. Este enfoque implica el desarrollo de proyectos de una o dos semanas de duración, centrados en un atributo de calidad específico, como la mejora de la usabilidad de la interfaz de usuario, la mejora del rendimiento del sistema o el refuerzo de la seguridad. Este enfoque hace hincapié en ejercicios prácticos utilizando modernos servicios web y frameworks de software, en lugar de intentar realizar mejoras superficiales en proyectos junior [104]. Algunas experiencias han demostrado estrategias educativas que utilizan proyectos ligeros con recursos públicos y gratuitos para el aprendizaje colaborativo y experimental.

Estos cursos se diseñan utilizando un enfoque iterativo que implica planificar, ejecutar, evaluar y mejorar. Los proyectos están disponibles en plataformas basadas en la nube como GoogleAppEngine, IBM Cloud Lite y Dropbox. Desde la perspectiva del profesorado, las plataformas de acceso público facilitan la evaluación de los proyectos de los estudiantes al permitir a los usuarios examinar en línea el tiempo de ejecución de los sistemas o componen-

Tabla 3-2.: Técnicas de formación en AS alineadas con las necesidades de la industria del software

Técnicas de formación	Referencias
Formación basada en mini-proyectos	[104] [66] [20] [65] [77] [86]
Formación basada en proyectos grandes	[120] [74] [72] [50] [128] [2] [115] [92] [73] [9] [21] [44] [58]
Formación basada en proyectos Open-Source	[97] [110] [50] [118] [51] [124] [123] [75]
Clientes reales o expertos provenientes de la industria	[104] [2] [123]
Conferencias con invitados del sector de la industria del software	[120] [29] [128] [118] [65] [55] [26]
Visitas de campo	[26]
Instructores y colaboradores invitados en ingeniería de software del sector industrial	[51] [56]
Desarrollo de software en las condiciones que se dan en la industria	[77]
Desarrollo global de software	[50]
Conocimientos y habilidades exigidos en la industria	[121] [57] [115]
Anuncios de empleo para adaptar los planes de estudios a las necesidades de la industria.	[121]
Herramientas aplicadas en la industria	[85]
Estado del arte de la AS	[128] [55]
Pensamiento de diseño ()Design thinking)	[86]

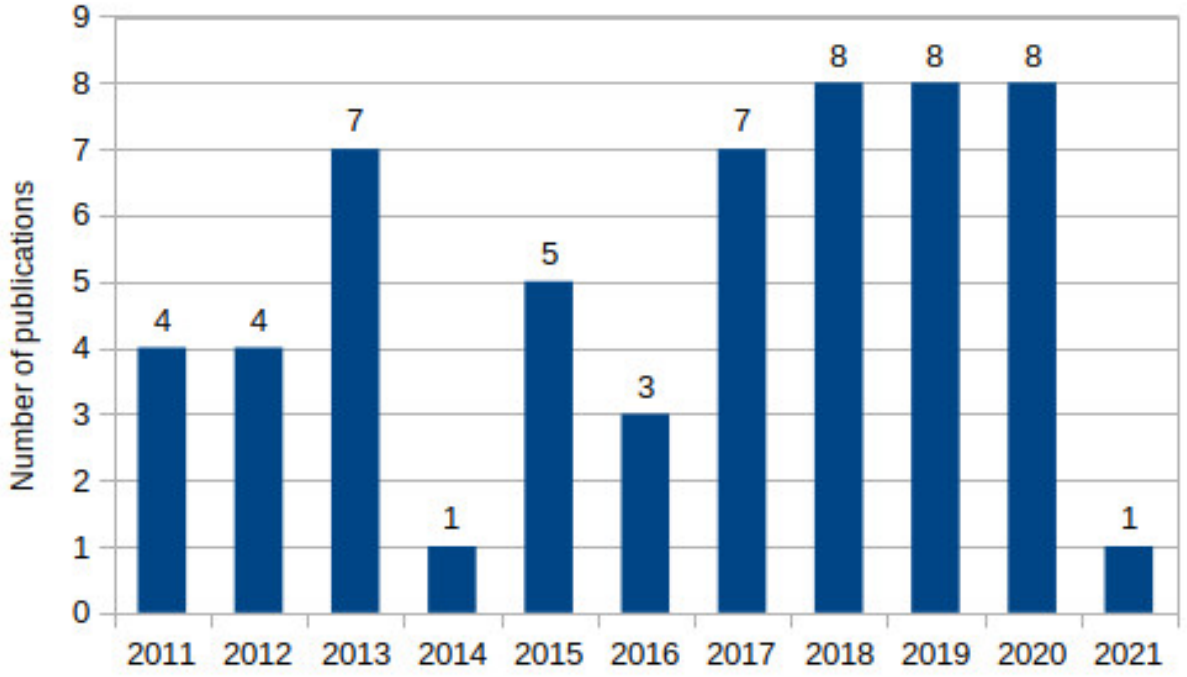

Figura 3-4.: Publicaciones de artículos al año centrados en la formación de arquitectos de software.

tes del proyecto [66]. El curso progresa desde el diseño de un gran sistema con sus requisitos y la discusión de las opciones de diseño a un conjunto de pequeños ejercicios semanales de codificación que implican la adición de nuevas características al sistema utilizando un patrón de diseño específico. Este enfoque proporciona a los estudiantes una experiencia de aprendizaje práctica [20].

En algunas experiencias, se proporcionó a los estudiantes el código fuente para analizar y evaluar el diseño arquitectónico [65]. En otros casos, los proyectos utilizaban un cliente simulado o real con una necesidad genuina [77]. Algunos cursos incorporaron métodos de Design Thinking y prácticas ágiles en el ciclo de vida del proyecto [86].

La formación basada en grandes proyectos implica proyectos de complejidad media y alta con múltiples requisitos que se centran en la arquitectura de software. Este enfoque permite a los estudiantes aprender arquitectura de software a través de un proyecto completo, permitiéndoles ver los resultados de su diseño arquitectónico en un producto final [120]. Los cursos de arquitectura e ingeniería de software suelen basarse en el aprendizaje por proyectos y en paradigmas colaborativos que simulan entornos industriales.

Algunos estudios incorporan prácticas ágiles de desarrollo en equipo [72], mientras que otros combinan la ingeniería global de software con proyectos de desarrollo de código abierto [50]. Estos proyectos permiten a los estudiantes aplicar conocimientos teóricos de arquitectura

de software a sistemas del mundo real [128]. Los estudiantes aprenden a diseñar sistemas de software que satisfagan las necesidades de las partes interesadas y proporcionen una experiencia de usuario positiva trabajando en retos reales [115] [92] [58]. Este enfoque exige que los estudiantes se comuniquen y colaboren eficazmente en equipo. Algunos cursos incluso proporcionan asistentes de investigación junior que actúan como tutores en las reuniones de equipo, ofreciendo asesoramiento y apoyo [73]. Los estudiantes también aprenden sobre la creación de arquitectura, el papel del arquitecto y la documentación que debe utilizarse para apoyar la arquitectura de software [9].

La formación en proyectos de código abierto (open-source) permite a los estudiantes interactuar con sistemas existentes, problemas reales y equipos de desarrollo de software reales para construir software de trabajo de alta calidad. Este enfoque proporciona a los estudiantes una oportunidad única de adquirir experiencia sólo presente en escenarios del mundo real, lo que puede aumentar sus habilidades y confianza en sí mismos [97]. Al trabajar en proyectos de código abierto, los estudiantes pueden realizar ampliaciones, corregir errores o analizar la arquitectura de los sistemas existentes. A continuación, ofrecemos diferentes alternativas para trabajar con proyectos de código abierto.

- Experiencias en la enseñanza del modelado orientado a objetos con UML (Unified Modeling Language) utilizando proyectos de código abierto son reportadas por [110].
- Los estudiantes analizan sistemas de código abierto y aprenden de los análisis realizados por otros equipos. Además, los estudiantes crean capítulos de libros basados en la documentación de la arquitectura de su sistema [118].
- Los proyectos de código abierto ofrecen a los estudiantes la oportunidad de adquirir experiencia a partir de errores comunes cometidos por desarrolladores principiantes, como la mala nombramiento de variables y métodos, colocar comentarios entre líneas, usar métodos largos, usar estructuras switch en lugar de polimorfismo, ausencia de pruebas unitarias y código duplicado [51].
- Algunas propuestas sugieren la creación de repositorios públicos de código que contengan proyectos reales de sectores industriales populares. Estos repositorios servirían de referencia a los instructores para mejorar la experiencia de aprendizaje de la arquitectura de software [124].
- Otras iniciativas proponen que los estudiantes desarrollen un componente, extensión o aplicación sobre un proyecto de software de código abierto ya existente [123].

En algunos cursos de AS participan **clientes reales o expertos del sector** que aportan requisitos y evalúan demostraciones. La universidad puede solicitar clientes candidatos a las empresas, siendo la mayoría de los candidatos titulados de las mismas instituciones educativas [104]. En otros casos, cada grupo de estudiantes tiene un mentor que actúa como cliente [2]. Además, algunos cursos recurren a expertos en software del sector para evaluar los proyectos

de clase mediante demostraciones. Además de ayudar con la carga de trabajo de evaluación, los expertos de la industria proporcionan a los estudiantes retroalimentación práctica basada en las mejores prácticas actuales de la industria [123].

Las conferencias con invitados del sector de la industria del software son valiosas en la enseñanza de la arquitectura de software. Arquitectos de software experimentados que han trabajado en el diseño de sistemas a gran escala comparten sus experiencias en cuestiones de arquitectura de software [128] [118] [65]. Los esfuerzos de colaboración entre el mundo académico y la industria son esenciales, y muchas facultades tienen acuerdos con empresas de desarrollo de software [29]. Algunos cursos ofrecen seminarios durante el semestre con invitados de la industria sobre temas innovadores [55]. Las conferencias suelen contar con la opinión de los estudiantes para planificar futuros eventos [26].

Las visitas de campo a empresas de desarrollo de software también proporcionan a los estudiantes una valiosa visión del mundo laboral. A través de estas visitas, los estudiantes pueden interactuar con desarrolladores, arquitectos y empresarios y ver cómo las empresas organizan sus proyectos, equipos de trabajo y tiempo. Este enfoque permite a los estudiantes tener un contacto temprano con la industria [26].

Los instructores y colaboradores invitados en ingeniería de software procedentes del sector industrial también pueden aportar valiosos conocimientos prácticos de ingeniería de software. A menudo están más cualificados para enseñar habilidades prácticas que los académicos puros y pueden dar a su material didáctico un valor agregado de interés para los estudiantes.

En algunos casos, a cada equipo de estudiantes se le asigna un mentor con experiencia en la industria y en proyectos, quien desempeña el papel de director de proyectos de la empresa[56].

Los estudiantes de AS aprenden el proceso de desarrollo de software **desarrollando software en las condiciones que se dan en la industria**. El proceso de desarrollo es esencial para organizar las funciones y actividades de todos los miembros del equipo.

El desarrollo global de software es otro enfoque, en el que los estudiantes forman equipos de diversas nacionalidades para desarrollar proyectos de software. Utilizan Scrum y están familiarizados con las aplicaciones Jira, Git y Moodle. La ingeniería de software global se puede combinar con proyectos de código abierto para proporcionar a los estudiantes una experiencia de la industria del software [50].

En el desarrollo de proyectos, los estudiantes pueden aplicar **conocimientos y habilidades demandados en la industria y desarrollar las habilidades esenciales** requeridas para carreras exitosas. Las habilidades adquiridas en los cursos se comparan con las habilidades blandas de la industria reportadas en la literatura [115]. Algunas experiencias implican usar los roles de la industria como analistas, arquitectos, diseñadores, probadores e integradores [57].

En el diseño de planes de estudio algunos utilizan los **anuncios de empleo para alinear los planes de estudio con las necesidades de la industria**. Los anuncios de empleo publican las competencias profesionales requeridas, lo que permite alinear los planes de estudio con las necesidades de la industria[121].

Otras experiencias se centran en el uso de **herramientas aplicadas a la industria** como bug tracking e integración continua (como Heroku y Vagrant), repositorios de código (como GitHub) [85]. El uso de máquinas virtuales (como Docker y Kubernetes) simplifica la instalación y configuración de estas herramientas. Para la arquitectura de software, los estudiantes deben familiarizarse con las herramientas de modelado y DevOps.

Para mantenerse al día con las prácticas actuales de AS, los estudiantes aprenden utilizando documentos y otros materiales del **estado del arte de la arquitectura de software** [128]. Hay muchas tareas en las que los estudiantes pueden trabajar [128]:

- En primer lugar, el profesor asigna a un estudiante la lectura de un artículo sobre arquitectura de software. Cada estudiante lee y responde a una serie de preguntas, incluyendo el problema, la solución, las contribuciones, los resultados y las lecciones aprendidas, entonces el estudiante hace un informe, utilizando diapositivas.
- Otra tarea requiere que los alumnos lean artículos en parejas y cooperen para crear notas de lectura que eviten los prejuicios individuales. Los estudiantes seleccionan artículos de revistas y conferencias reconocidas.
- En una tercera tarea, los estudiantes investigan tanto en el mundo académico como en la industria. Buscan grupos de investigación y empresas industriales en arquitectura de software, clasifican los diez mejores equipos u organizaciones y explican las razones de la clasificación en el informe.

Pensamiento de diseño (Design thinking) es un enfoque para resolver problemas complejos utilizando métodos creativos. Hace hincapié en la importancia de la empatía entre diseñadores y usuarios y fomenta la exploración compartida de problemas y soluciones. Durante la fase de ideación, los arquitectos de software evalúan la viabilidad de la solución considerada, eliminando la creación infructuosa de prototipos de soluciones según el estado actual de las tecnologías [86].

En resumen, las estrategias más utilizadas son la formación basada en grandes proyectos (12 citas), la formación en proyectos de código abierto (8 citas) y la formación basada en pequeños proyectos (7 citas).

A continuación se exponen las estrategias relacionadas con las metodologías de enseñanza.

Los profesores imparten **clases magistrales** en la enseñanza de la arquitectura de software. Los profesores explican y comparten sus conocimientos con los estudiantes en persona [120] [69]. Estas clases se centran en enseñar a los estudiantes a diseñar sistemas de software

Tabla 3-3.: Metodologías de enseñanza en la formación de arquitectos de software

Metodologías de enseñanza	Referencias
Clases magistrales	[120] [69] [92]
Aula invertida	[86] [92] [57] [60] [85]
Aprendizaje basado en problemas	[68] [60] [58]
Aprendizaje basado en casos	[69] [68] [65] [55] [16] [74]
Aprendizaje en línea	[125] [106]
Sistema tutorial inteligente	[68]

que resuelvan problemas del mundo real, cumplan los requisitos de las partes interesadas y proporcionen una experiencia de usuario positiva [92].

En algunas experiencias, los estudiantes han encontrado que el enfoque de **aula invertida** es mucho más eficaz y atractivo que el enfoque tradicional porque se centra en la resolución de problemas [57]. El aula invertida favorece las experiencias de aprendizaje inmersivas, colaborativas y activas [85]. Algunos cursos tienen un enfoque mixto que combina clases de aula invertida con instrucción tradicional [92].

El aprendizaje basado en problemas es una estrategia utilizada en la enseñanza de la AS en la que equipos de estudiantes abordan problemas arquitectónicos, y los instructores desempeñan un papel mínimo sin interferir en el debate. A medida que los estudiantes exploran los problemas, los instructores funcionan como facilitadores y guían las preguntas para que vuelvan al objetivo principal de aprendizaje. Por ejemplo, un subgrupo explica una determinada solución de diseño, mientras que el otro subgrupo comprende la solución de diseño de otro grupo y prepara las preguntas para la presentación propiamente dicha. Cada estudiante escucha las experiencias y el pensamiento de varios estudiantes aplicados en un escenario de la vida real [68].

El uso de proyectos y del aprendizaje basado en problemas fomenta el trabajo colaborativo [60] que examina el efecto del enfoque basado en problemas/proyectos en la adquisición de las habilidades de los estudiantes [58].

El aprendizaje basado en casos es otro enfoque en el que los estudios de casos son "escenarios de la vida real que permiten a los estudiantes analizar, aplicar los conceptos enseñados y mitigar las posibles compensaciones (trade-offs) dentro de un contexto realista" [69]. Se centra en el diseño y análisis de proyectos reales de software para empresas.

Con este enfoque, los estudiantes trabajan en equipos dentro de escenarios de la vida real. Los instructores desempeñan un papel mínimo y no interfieren en el debate. Cuando los alumnos se plantean dudas, los instructores actúan como facilitadores y utilizan preguntas orientadoras para que vuelvan al objetivo principal de aprendizaje [68].

Algunos profesores llevan a cabo una enseñanza basada en casos en un entorno de aula invertida. Los casos incluyen, por ejemplo, un sistema ATM estilo cliente-servidor, un sistema de venta online orientado a servicios, un sistema de monitorización de emergencias, un sistema de gestión automatizado basado en web y diseño de arquitectura de software en tiempo real. Cada estudiante analiza el caso y completa las tareas de diseño requeridas para la asignación en el plazo de una semana [65].

Según Capilla et al. [16], los estudiantes podrían trabajar en equipos de 5 o 6 miembros con funciones específicas asignadas a cada uno. Los arquitectos senior toman las decisiones de diseño arquitectónico, identifican los problemas de diseño y plasman las decisiones de diseño arquitectónico. Dos arquitectos cognitivos cuestionan las decisiones de los arquitectos superiores y plantean problemas cuando existen riesgos o información incompleta. Uno o dos arquitectos junior supervisan la comprensión y el modelado de las decisiones capturadas y proporcionan una arquitectura de solución.

El aprendizaje en línea en la enseñanza de la AS se imparte a través de dos modalidades fundamentales: MOOCs (Massive Open Online Courses) y SPOCs (Small Private Online Courses).

Wendt et al. [125] ofrecen un pequeño curso privado en línea (SPOC) sobre diseño de software, y tras completar el curso, aplican una encuesta en la que los participantes valoran el contenido del curso. La industria recibirá cursos SPOC para la formación del personal.

Schmidt et al. [106] describen las experiencias de producción e impartición de un MOOC titulado "Pattern-Oriented Software Architecture for Concurrent and Networked Software". También discuten "las implicaciones más amplias de los MOOC en el aprendizaje permanente y su papel en la mejora de la calidad y la productividad de los profesionales del software en el mundo académico y la industria" [106].

Otro enfoque del aprendizaje en línea en la enseñanza de la AS es el **Sistema Tutorial Inteligente**, que proporciona vías de aprendizaje personalizadas mediante el seguimiento del progreso de cada estudiante. Según Thevathayan et al. [116], los estudiantes interactúan con un Sistema Tutorial Inteligente (STI) que satisface necesidades variadas permitiendo diferentes modos de aprendizaje. Un STI funciona con cuestionarios de opción múltiple que plantean problemas de UML y de diseño. Elaborar preguntas que muestren conceptos o patrones específicos es una tarea difícil. Una de las ventajas de los STI basados en la nube es que el material didáctico desarrollado puede compartirse con otras instituciones de todo el mundo.

Se encontró que las estrategias más utilizadas son el aprendizaje basado en casos (6 citas) y las clases con aula invertida (4 citas).

La revisión de la literatura presenta experiencias de cursos relacionados con AS. Cada experiencia explora una o más estrategias de enseñanza desde una perspectiva pedagógica y

tecnológica. Sin embargo, la revisión también destaca las estrategias de enseñanza menos eficaces en la formación de arquitectos de software. La Tabla **3-4** ofrece un resumen de estas estrategias menos eficaces.

Tabla 3-4.: Estrategias que han resultado poco eficaces para la formación de arquitectos de software.

Categoría	Estrategia	Ref
Alcance	La revisión de la literatura muestra que un solo curso que abarque toda la complejidad de la arquitectura y el diseño de software detallado requiere mucho tiempo. Tratar toda la ingeniería de software en uno o dos cursos requiere tiempo y esfuerzo.	[104]
Alcance	Combinar la lectura de artículos científicos con el desarrollo de un proyecto de curso no es conveniente, ya que no permite a los estudiantes centrarse en el desarrollo del proyecto.	[50]
Proyecto	Trabajar con proyectos que no motivan a los alumnos es una estrategia inadecuada. En su lugar, podrían elegir el proyecto que les motive.	[118]
Pedagogía	Utilizar el tiempo de clase para enseñar la teoría abstracta del diseño de arquitecturas de software desmotiva a los alumnos. El profesor puede promover el aprendizaje activo mediante talleres, estudios de casos y debates.	[65]
Recursos	Los casos prácticos de los libros de texto son limitados e inadecuados para enseñar AS.	[65]
Contexto industrial	Las clases magistrales tradicionales y los proyectos "de juguete" son inadecuados en el contexto de una formación industrial o empresarial porque los estudiantes ya tienen experiencia práctica en grandes proyectos de software. Necesitan más tiempo para presentar, comparar y practicar diversas teorías arquitectónicas. Los ejemplos y ejercicios deben centrarse en sistemas de software existentes desarrollados por las empresas.	[22]

3.2.2. RQ2: ¿Qué contenidos de arquitectura de software suelen incluirse en los cursos universitarios?

Esta pregunta implica dos perspectivas: identificar los temas tratados y alinearlos con los objetivos de aprendizaje de los cursos. La Tabla **3-5** muestra los objetivos de aprendizaje identificados mediante el mapeo sistemático. Aunque todos estos cursos se centran en la arquitectura de software, tienen enfoques diferentes. Los cursos pueden ser básicos o avanzados y pueden basarse en proyectos existentes o en formulaciones de proyectos.

Tabla 3-5.: Objetivos de aprendizaje según la literatura

No	Objetivo de aprendizaje	Referencias
1	Definir y explicar los conceptos centrales de la arquitectura de software y utilizar y describir patrones de diseño/arquitectura, métodos para diseñar arquitecturas de software, métodos/técnicas para conseguir cualidades de software y métodos para documentar y evaluar la arquitectura de software.	[120]
2	Enseñanza de un conjunto de patrones de arquitectura distribuida.	[66] [10]
3	Solucionar los problemas arquitectónicos de los sistemas existentes.	[104]
4	Analizar y diseñar la arquitectura del software y adquirir la experiencia pertinente en sistemas a gran escala.	[128]
5	Revisar arquitecturas de sistemas de software y relacionar métodos de ingeniería de software para diseñar sistemas complejos de software intensivo.	[22]
6	Enseñar a los estudiantes métodos modernos de arquitectura de software y hacer que se conviertan en buenos arquitectos de sistemas en los 5 u 8 años siguientes a su graduación.	[55]
7	Explicar cómo afectan los atributos de calidad al diseño de la arquitectura del sistema.	[55]
8	Demostrar la capacidad de tener éxito en una experiencia real de diseño de software.	[77]

La Tabla **3-6** presenta los temas tratados en la revisión de la literatura. Al igual que ocurre con los objetivos de aprendizaje, el contenido varía de un curso a otro. Sin embargo, la literatura no proporciona información específica, como temas y subtemas, lo que hace difícil llevar a cabo examinar de forma más exhaustivo el contenido. No obstante, las experiencias reportadas comparten contenidos comunes, incluyendo fundamentos de arquitectura de software, patrones de arquitectura, atributos de calidad, principios de diseño, patrones de diseño, análisis, diseño y evaluación de arquitectura.

En resumen, los temas más citados incluyen patrones de arquitectura (6 citas), atributos de

Tabla 3-6.: Contenidos de los cursos según la revisión de la literatura.

No	Tema	Referencias
1	Patrones de arquitectura	[69] [118] [55] [16] [65]
2	Atributos de calidad	[69] [67] [125]
3	Estudios de caso sobre aplicaciones web, móviles y en la nube.	[69]
4	Fundamentos de las arquitecturas de software: principios y conceptos	[67] [55]
5	Vistas arquitectónicas	[118]
6	Principios de diseño	[68] [118] [55]
7	Líneas de productos	[118]
8	Deuda técnica	[118]
9	Análisis, diseño y evaluación de la arquitectura de software	[21] [16]
10	Patrones de diseño	[16] [55] [125]
11	Modelado de software y uso de diagramas UML	[125]
12	Refactorización	[125]

calidad (3 citas), principios de diseño (3 citas), análisis, diseño y evaluación de la arquitectura de software (3 citas) y patrones de diseño (3 citas).

3.2.3. RQ3: ¿Cuáles son las habilidades clave que un estudiante en arquitectura de software debe desarrollar durante su formación?

La Tabla **3-7** y la Tabla **3-8** ofrecen un resumen de las competencias técnicas y sociales que deben desarrollar los arquitectos de software. En esta sección, hablaremos de las habilidades blandas de los arquitectos de software.

Una de las habilidades clave que necesitan los arquitectos de software es la **comunicación** efectiva. Esto incluye la capacidad de hablar, escribir y presentar ideas de forma clara y

concisa, especialmente cuando se trata de problemas complejos que requieren una solución de diseño directa. Diversas fuentes bibliográficas destacan la importancia de las habilidades de comunicación para los arquitectos [104] [110] [50] [56] [115] [92] [99] [34] [123] [63] [54] [51] [77] [10].

Por otra parte, las habilidades de comunicación son cruciales para los estudiantes que siguen una carrera en la arquitectura de software [51]. Es vital garantizar que los estudiantes desarrollen habilidades comunicativas fundamentales, como leer, escribir, escuchar, hablar y comprender [77].

Los arquitectos de software se relacionan y comunican con expertos de diversas disciplinas para desempeñar eficazmente su función. Para ellos es importante comprender la terminología, los problemas comunes, las necesidades y los diversos enfoques culturales (incluidas las formas de trabajar y de pensar) [10].

La comunicación efectiva es crucial para los arquitectos, tanto externa (con los clientes) como interna (con los miembros del equipo) [58]. Los arquitectos deben tener habilidades para comunicarse con las partes interesadas y lograr el consenso en las decisiones [118]. Las habilidades de liderazgo son esenciales para que los arquitectos dirijan, presenten, negocien y justifiquen sus diseños y decisiones [68]. Deben escuchar y resolver problemas con los responsables de la aplicación del diseño [124].

El **trabajo en equipo** es otra habilidad fundamental para los arquitectos de software. Ellos colaboran estrechamente con otros miembros del equipo de desarrollo, como programadores [74] [77] [50] [128] [56] [86] [58] [115] [92] [34] [57] [65] [44] [63] [54]. El aprendizaje de la arquitectura implica un intenso trabajo en equipo [66], y el equipo debe analizar y tomar decisiones arquitectónicas conjuntamente [9].

Las habilidades de **liderazgo y negociación** son cruciales para que los arquitectos de software dirijan, presenten, negocien y justifiquen sus diseños y decisiones arquitectónicas [1] [68] [104]. Estas habilidades incluyen tomar decisiones independientes, tomar la iniciativa, demostrar un juicio independiente, ser influyente e imponer respeto [13].

Las **habilidades artísticas** también son necesarias para que los arquitectos de software creen diseños sencillos que sean fáciles de entender y resuelvan problemas complejos [104] [124]. Las habilidades artísticas implican el conocimiento de técnicas y herramientas de diseño únicas, cómo diseñar sistemas complejos multiproducto, análisis y diseño orientado a objetos, diagramas UML y modelado de análisis UML [13].

Junto con las habilidades blandas, los arquitectos de software poseen una serie de habilidades técnicas. Éstas se explican a continuación.

Estar familiarizado con múltiples tecnologías permite elegir la tecnología adecuada para un determinado proyecto. Aunque los arquitectos de software no tienen por qué ser

Tabla 3-7.: Habilidades interpersonales del arquitecto de software.

No	Habilidades blandas	Referencias
1	Capacidad de comunicación	[104] [110] [77] [50] [10] [56] [58] [115] [92] [99] [34] [123] [118] [51] [63] [54] [124]
2	Trabajo en equipo	[66] [74] [68] [77] [50] [128] [56] [86] [58] [115] [92] [34] [57] [9] [65] [44] [63] [54]
3	Liderazgo y negociación	[1] [68] [104]
4	Habilidades artísticas	[104] [124]

expertos en tecnología, mantenerse al día de las últimas tendencias tecnológicas es importante [104] [86] [124].

Como arquitecto de software, es crucial **comprender el dominio en el que funcionará un sistema**. El dominio significa comprender el entorno empresarial en el que funcionará el software para diseñar y desarrollar una solución adecuada. Implica comprender los procesos, reglas y operaciones empresariales específicos, así como las necesidades y requisitos de los usuarios y las partes interesadas. Este conocimiento es esencial en el diseño de una arquitectura que se alinea con las metas y objetivos de negocio [124].

Las **habilidades analíticas** son cruciales para que los arquitectos de software entiendan el problema, identifiquen las causas potenciales y tomen decisiones informadas para el proyecto [69] [57] [9] [66]. Los arquitectos deben identificar las causas de los problemas de alto nivel en los diseños existentes, como los problemas de rendimiento del sistema o las vulnerabilidades de seguridad [124].

Otra habilidad crítica para los arquitectos de software es la capacidad de **diseñar, modelar, analizar y evaluar arquitecturas de software** [118] [70] [111] [128] [67] [34] [54]. Los arquitectos deben ser expertos en la aplicación de patrones y marcos para crear aplicaciones de alta calidad.

El diseño de la arquitectura de sistemas de software complejos y a gran escala con numerosos requisitos y millones de líneas de código requiere habilidades excepcionales de modelado y abstracción [22].

Además de las habilidades técnicas, los arquitectos de software necesitan sólidas **habilida-**

des de investigación para gestionar y coordinar todos los elementos de un proyecto de aplicación con éxito. La capacidad de investigación permite a los arquitectos comprender situaciones complejas y resolver problemas. Antes de diseñar cualquier solución de software, los arquitectos investigan para comprender los requisitos y necesidades de la empresa. Esto incluye la realización de entrevistas, debates y análisis con los usuarios y las partes interesadas para entender lo que se espera del sistema. La investigación permite a los arquitectos captar los requisitos con precisión y definir una arquitectura que satisfaga las necesidades empresariales [128].

Aunque los arquitectos de software no tienen por qué ser expertos en programación, deben poseer **unos conocimientos mínimos de programación**. Las funciones de arquitectura suelen requerir experiencia en diseño y programación de software. Los arquitectos necesitan saber lo suficiente sobre programación para comunicarse eficazmente con los desarrolladores [66].

Para tomar **decisiones arquitectónicas** eficaces, el arquitecto de software debe recopilar y analizar la información pertinente para tomar decisiones con conocimiento de causa. Esto implica comprender los requisitos del sistema, las limitaciones técnicas, las necesidades empresariales y otros factores relevantes. Analizando esta información, los arquitectos pueden evaluar distintas opciones y tomar decisiones basadas en datos sólidos. En situaciones de incertidumbre, presión de tiempo o exploración limitada de alternativas, la capacidad de tomar decisiones de diseño informadas es fundamental para el éxito [69].

El pensamiento sistémico es una forma de entender y analizar las interacciones entre variables dentro de un sistema o múltiples subsistemas, expresadas en términos de retroalimentación. Este enfoque trata de analizar estas interacciones de forma ordenada y global, considerando todos los elementos interrelacionados. Los arquitectos de software necesitan poseer habilidades de pensamiento sistémico y holístico para abordar los problemas desde múltiples perspectivas [124].

En resumen, las competencias blandas más comunes para los arquitectos de software son la comunicación (18 citas) y el trabajo en equipo (15 citas). Estas habilidades son esenciales para una colaboración y comunicación eficaces con los miembros del equipo y las partes interesadas. Por otro lado, las habilidades técnicas más comunes para los arquitectos de software son la creación y evaluación de arquitecturas (9 citas), la capacidad de análisis (5 citas) y el conocimiento de múltiples tecnologías (4 citas).

Tabla 3-8.: Habilidades técnicas del arquitecto de software.

No	Habilidad técnica	Referencias
1	Estar familiarizado con múltiples tecnologías	[104] [86] [124]
2	Comprender el ámbito en el que funcionará un sistema	[104] [124]
3	Habilidades analíticas	[104] [69] [67] [57] [9] [66] [124]
4	Diseñar, modelar, analizar y evaluar arquitecturas de software	[70] [111] [77] [128] [67] [34] [118] [106] [22] [54]
5	Habilidades de investigación	[66] [128]
6	Habilidades mínimos de programación	[67] [66]
7	Toma de decisiones arquitectónicas	[69] [9]
8	Pensamiento sistémico	[124]

3.2.4. RQ4: ¿Cuáles son los retos a los que se enfrentan los estudiantes durante la formación en arquitectura de software?

El objetivo de esta pregunta es identificar los retos o desafíos que surgen durante el proceso de formación en arquitectura de software. La Tabla **3-9** presenta un resumen de dichos retos. A continuación, describimos cada uno de ellos:

1. *Falta de experiencia en el mundo real por parte de los instructores*. Para ser eficaces, los instructores deben poseer una amplia experiencia en el mundo real. Sin embargo, muchos instructores tienen formación académica y pueden tener una experiencia limitada en el diseño de proyectos a gran escala [104].

2. *La naturaleza abstracta y difusa* de la arquitectura de software plantea un reto importante para los estudiantes. Conceptos como principios de diseño, compensaciones,

patrones arquitectónicos y líneas de productos pueden ser difíciles de entender [66] [115] [118]. A diferencia de la programación, en la arquitectura de software no hay soluciones claras correctas o incorrectas. Uno de los principales retos de la enseñanza de la arquitectura de software es hacer que estos conceptos abstractos sean más accesibles para los estudiantes. Para los principiantes, esto requiere un cambio de mentalidad de la programación concreta de software al diseño arquitectónico abstracto. Los arquitectos acostumbrados a escribir código y conseguir resultados deterministas se adaptan a estructurar componentes en un diagrama y justificar sus decisiones basándose en compensaciones, ya que las soluciones perfectas no existen. Además, los aspectos y principios del entorno pueden variar en función de las experiencias y los antecedentes de los estudiantes, lo que hace que las soluciones de arquitectura de software sean abstractas y difíciles de comprender [68].

3. *La dificultad para involucrar a clientes y expertos de la industria* es un reto común en los entornos académicos [29]. Estas personas suelen tener agendas muy ocupadas, lo que dificulta su participación en la formación sobre arquitectura de software. [29].

4. *La enseñanza de la arquitectura de software requiere ir más allá de los métodos de enseñanza tradicionales* para tener en cuenta la complejidad de las interacciones sociales y el desarrollo colaborativo de software en un entorno real. A diferencia de la enseñanza universitaria tradicional, que se centra en impartir conocimientos, la arquitectura de software requiere la capacidad de aplicar los conocimientos de manera eficaz para obtener resultados [96] [99] [118]. Las clases tradicionales no motivan a los estudiantes de arquitectura de software [85].

5. *Los proyectos pequeños y sencillos son insuficientes para proporcionar a los estudiantes experiencia práctica*. Para enseñar arquitectura de software de manera eficaz, se necesitan problemas complejos con múltiples atributos de calidad y un contexto adecuado [39]. Los proyectos de clase suelen ser "proyectos de juguete" que carecen del alcance y la madurez necesarios para el desarrollo real de software [97]. Los académicos e investigadores no suelen trabajar en proyectos en entornos industriales, lo que dificulta el uso de proyectos desarrollados por la industria en el aula o la interacción con las empresas en el contexto educativo [110].

6. *Falta de uso de proyectos de código abierto*. Los repositorios de código abierto ofrecen beneficios para el aprendizaje de arquitecturas de software. Sin embargo, los instructores necesitan adquirir experiencia en este tipo de proyectos [110]. Trabajar en proyectos de código abierto es un reto para los profesores y sus alumnos, ya que requieren un profundo conocimiento y experiencia en tecnología. Es necesaria una etapa de aprendizaje antes de la ejecución de las tareas [50].

7. *El seguimiento personalizado* es necesario para cada alumno. Al igual que un estudiante de música, los estudiantes de arquitectura de software requieren una atención

personalizada por parte de sus profesores. Sin embargo, esto puede ser un reto para los equipos grandes [116].

8. *Los métodos modernos desarrollados en la industria pueden tardar en llegar al mundo académico*, lo que da lugar a enseñanzas en el aula que pronto pueden quedar obsoletas [116].

9. *Enseñar arquitecturas de software es una tarea desafiante* que requiere que los educadores equilibren calidad, alcance, profundidad, aplicabilidad, habilidades blandas y técnicas, y aprendizaje individual y colaborativo [86]. El aprendizaje requiere un contexto realista, trabajo en equipo, complejidad adecuada y formación práctica [9]. Los educadores necesitan encontrar el equilibrio adecuado entre "realismo" y control en el aula para los estudiantes de arquitectura de software [1]. La mayoría de los cursos de arquitectura se centran en conceptos teóricos o no pueden sumergir a los estudiantes en proyectos complejos de nivel industrial a menos que los instructores procedan del sector industrial [124].

10. *La necesidad de que los cursos se adapten a los cambios* es crucial para ofrecer a los estudiantes la mejor educación posible. Las universidades deben esforzarse por mejorar la experiencia de aprendizaje de los cursos y actualizar sus contenidos para satisfacer los requisitos cambiantes [73]. Las clases de arquitectura de software deben ser ajustables y actualizarse periódicamente a las fluctuaciones del negocio, los avances tecnológicos y las necesidades de los estudiantes, de modo que estos puedan recibir la instrucción más precisa y actual posible.

11. *Los escenarios de proyectos terminados son útiles en la formación de arquitectura de software, pero también pueden intimidar a los estudiantes*, que pueden sentirse obligados a hacer cambios. En cambio, los estudiantes prefieren nuevos proyectos que sean más accesibles y les permitan comprender los requisitos y tener un control total sobre la arquitectura, el diseño y la estructura del software [123].

12. *Existe un importante vacío entre las competencias de los estudiantes y las expectativas de los directivos de la industria o del personal contratado*. Este desfase impide que los estudiantes consigan ofertas de trabajo o cumplan las expectativas de la industria del software. Existe una brecha de conocimientos entre la teoría aprendida en el aula y los requisitos reales exigidos por la industria [57].

13. *Hay una necesidad de aclarar cómo trabajar con la arquitectura de software en metodologías ágiles*. En los métodos ágiles, no siempre está claro cuándo crear la arquitectura, el papel del arquitecto, cómo debe ser la documentación de la arquitectura, los métodos y actividades implicados, y el valor y los costes asociados a ellos [9].

14. *Las actividades en las etapas de arquitectura de software requieren que los estudiantes apliquen conocimientos* de cursos anteriores, como redes, bases de datos, sistemas ope-

rativos, sistemas distribuidos e ingeniería de software. Una cuidadosa consideración y aplicación de estos conocimientos son necesarios [65].

En resumen, las dificultades más mencionadas son la naturaleza abstracta y difusa de la arquitectura de software (8 citas), la falta de experiencia práctica real en proyectos pequeños y sencillos (4 citas), la necesidad de ir más allá de los métodos de enseñanza tradicionales a la hora de enseñar arquitectura de software (5 citas) y la naturaleza desafiante de la enseñanza de arquitecturas de software (4 citas). La naturaleza abstracta y difusas de la arquitectura de software es el obstáculo más discutido, lo que convierte la enseñanza de esta materia en una empresa considerable.

3.2.5. RQ5: ¿Qué métodos se han utilizado para validar las experiencias de formación y alcanzar los objetivos propuestos?

Cada experiencia presentada en la literatura detalla un enfoque específico para formar a los estudiantes en arquitecturas de software mediante la aplicación de estrategias de enseñanza. Cada enfoque requiere un diseño del curso y su posterior implementación. Una vez realizadas estas experiencias, es esencial comprender la eficacia de las estrategias de enseñanza empleadas.

Se han propuesto diversos métodos para determinar la eficacia de los procesos de enseñanza-aprendizaje. La Tabla **3-10** resume los métodos de validación utilizados para los cursos. A menudo se administran encuestas a los estudiantes al final del curso para medir la satisfacción utilizando el Modelo de Aceptación de la Tecnología (TAM). Otra técnica consiste en comparar la satisfacción con el curso propuesto con la de un curso tradicional (con estudios que utilizan la prueba de Kruskal-Wallis para verificar los datos). Una tercera técnica mide la calidad de los artefactos producidos, como especificaciones de requisitos, modelos de análisis, modelos de diseño y pruebas. En cuarto lugar, se analizan las evaluaciones de los alumnos durante el curso para comprender el proceso de aprendizaje y los objetivos. Por último, una quinta técnica consiste en analizar y comparar las competencias adquiridas en el curso con las que aparecen en la literatura.

Las técnicas más utilizadas fueron las encuestas a los estudiantes (20 citas), las métricas para analizar las evaluaciones de los estudiantes (4 citas) y las métricas para evaluar la calidad de los artefactos producidos (3 citas). La mayoría de las experiencias utilizaron la técnica de la encuesta (69 %, como se ve en Figura **3-5**) porque permite recoger las percepciones de los estudiantes y analizar sus comentarios. Sin embargo, esta técnica requiere más cuidado para manejar las amenazas de la validez.

Hemos analizado las técnicas de validación en conjunción con las estrategias de enseñanza de

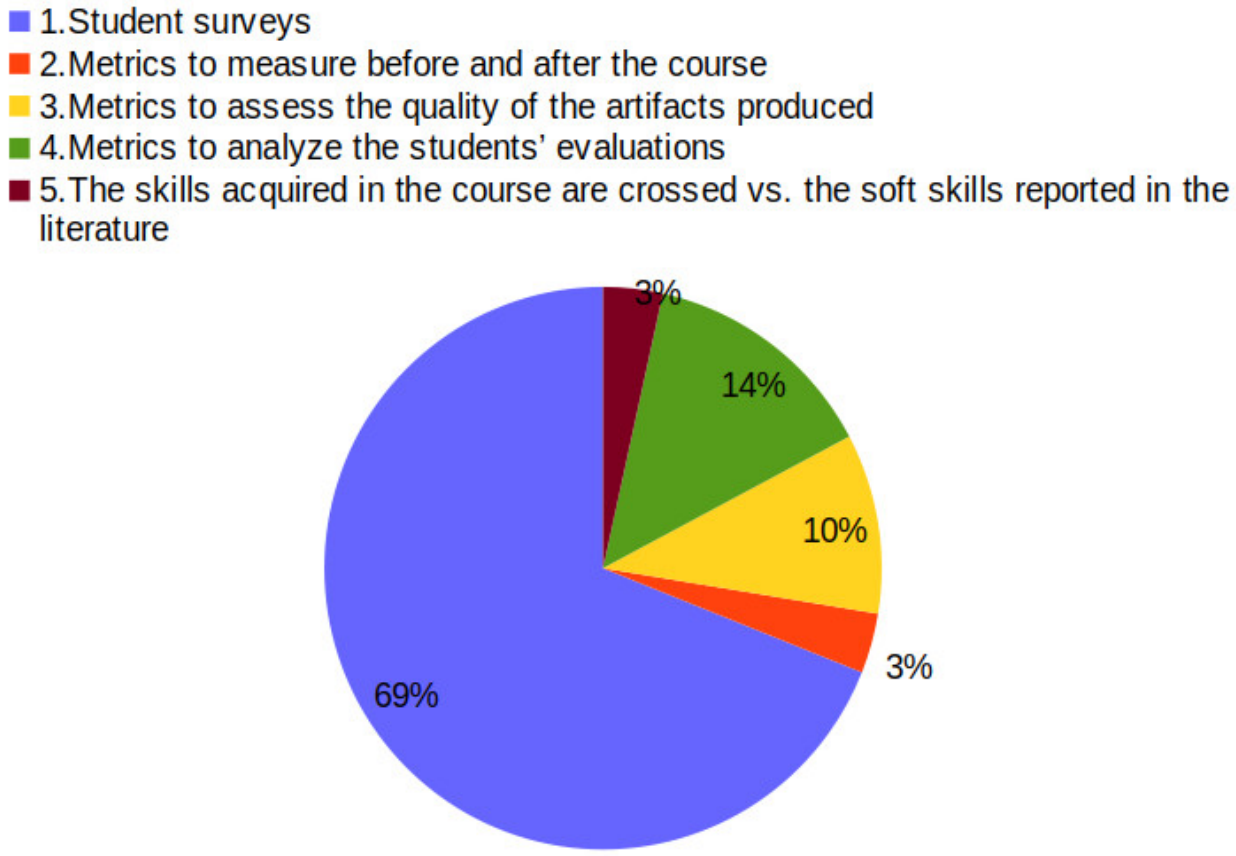

Figura 3-5.: Métodos de validación de los cursos reportados en la literatura.

la RQ1, y los resultados se muestran en Figura **3-6**, que presenta un diagrama de burbujas con tres dimensiones. El eje Y representa las estrategias de enseñanza, el eje X representa las técnicas de validación del curso, y la anchura de la burbuja corresponde al número de experiencias que utilizan la estrategia y la validación intersectadas. El diagrama revela que la técnica más utilizada es el cruce de proyectos cortos con validación mediante encuestas (seis experiencias). Además, algunas estrategias no tienen técnicas de validación asociadas, como el estado del arte de la arquitectura de software.

3.3. Discusión

Este mapeo sistemático analizó las estrategias, contenidos, habilidades, desafíos y métodos de validación de cursos para la formación de estudiantes en arquitectura de software a partir de estudios seleccionados. Los artículos fueron revisados a través de seis bases de datos, y se examinaron los documentos sobre la enseñanza de la arquitectura de software desde enero de 2011 hasta marzo de 2021. A partir de este análisis, se seleccionaron 56 estudios primarios

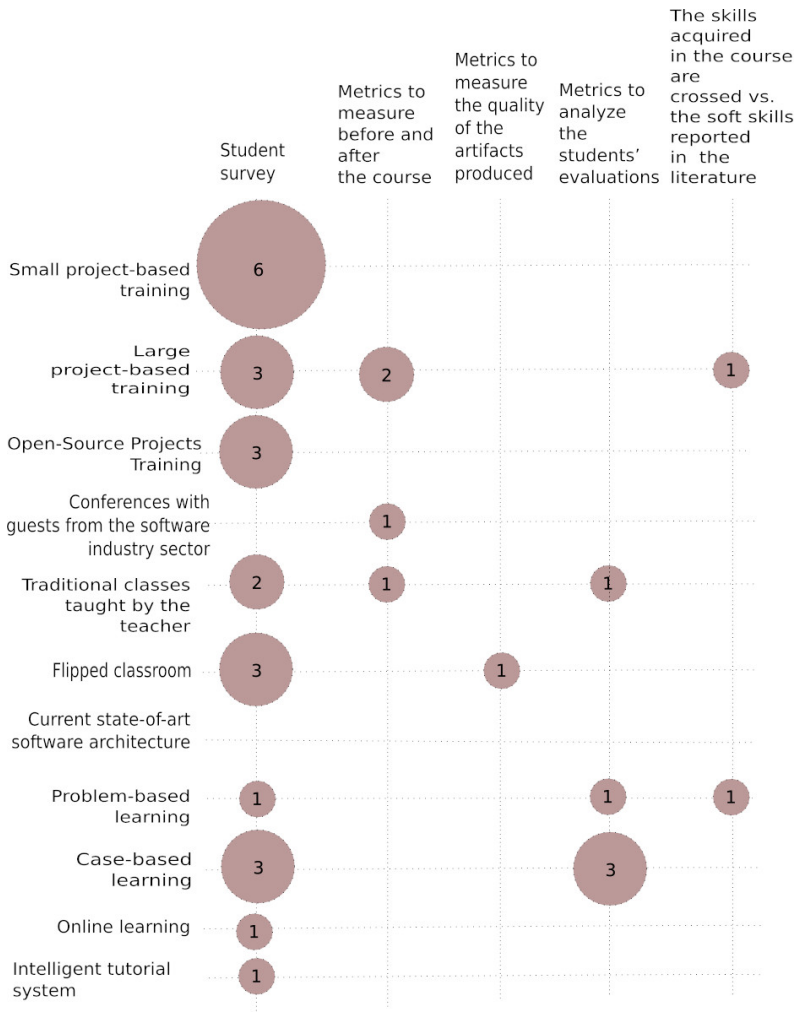

Figura 3-6.: Estrategias de enseñanza frente a técnicas de validación.

para una mayor extracción y análisis de datos, proporcionando una comprensión integral de cómo se está implementando el proceso de formación para la arquitectura de software en las universidades, teniendo en cuenta las necesidades de la industria. Los principales aspectos destacados de este mapeo sistemático se resumen a continuación:

1. La enseñanza de la arquitectura de software en los programas de ciencias de la computación e ingeniería de software es un tema pertinente en la comunidad científica, con un número constante de publicaciones en la última década. La mayor cantidad de publicaciones se registró entre 2018 y 2020, con un promedio de ocho publicaciones por año. Los países más interesados en este tema son Estados Unidos, India, China y Brasil, que representaron el 44 % del total de publicaciones en los últimos diez años.

2. La enseñanza de las arquitecturas de software presenta muchos retos para los educadores. Las dificultades más señaladas en la bibliografía incluyen la naturaleza abstracta y ambigua de la arquitectura de software (8 citas), la falta de experiencia práctica real en proyectos pequeños y sencillos (4 citas), la necesidad de ir más allá de los métodos de enseñanza tradicionales a la hora de enseñar arquitectura de software (5 citas) y la naturaleza desafiante de la formación en arquitecturas de software (4 citas).

3. Aunque el reto más importante señalado es la naturaleza abstracta y difusa de las arquitecturas de software, ninguna estrategia de formación se centra directamente en abordar este reto.

4. La revisión bibliográfica presenta diversas estrategias didácticas para adecuar los cursos de formación en arquitectura de software a las necesidades de la industria del software. Estas técnicas se han clasificado en dos categorías: formación de arquitectos de software y metodologías de enseñanza. Las estrategias más utilizadas para la formación de arquitectos de software son la formación basada en grandes proyectos (12 citas), la formación en proyectos de código abierto (8 citas) y la formación basada en mini-proyectos (7 citas). El aprendizaje basado en casos (6 citas) y el aula invertida (4 citas) son las estrategias más utilizadas relacionadas con las metodologías de enseñanza. A pesar de estas estrategias, ninguna solución completa aborda los retos de la formación de arquitectos de software. Los educadores siguen teniendo dificultades para formar a los arquitectos de software y confían en su experiencia o en las pruebas de ensayo y error. Los nuevos educadores necesitan directrices o patrones para diseñar planes de estudios de arquitectura de software.

5. Aunque en las experiencias de enseñanza se utilizan diversas estrategias de formación, existe un vacío sobre cómo combinar varias estrategias para que el proceso de enseñanza sea más eficaz.

6. La Tabla **3-3** muestra que el aprendizaje en línea tiene dos experiencias, y los sistemas tutoriales inteligentes sólo tienen una referencia. Estas dos líneas pueden ser un

contexto apasionante para la investigación en la formación en arquitectura de software.

7. Los estudiantes de arquitectura de software necesitan desarrollar sus capacidades gradualmente a lo largo del tiempo, y es necesario ponerse de acuerdo sobre los conjuntos de habilidades relevantes. Aunque el abanico de habilidades técnicas y sociales es amplio, las dos más mencionadas son la comunicación (18 veces) y la colaboración (15 veces).

8. La Tabla **3-7** y la Tabla **3-8** muestran las habilidades blandas y técnicas del arquitecto de software. Sin embargo, es necesario estudiar qué estrategias didácticas fomentan el desarrollo de estas habilidades.

9. Las experiencias reportadas cubren temas fundamentales como arquitecturas de software, patrones de arquitectura, patrones de diseño, atributos de calidad, principios de diseño, análisis, diseño y evaluación de arquitecturas.

10. Las técnicas de validación más utilizadas en los cursos de arquitectura de software son las encuestas a los estudiantes (20 citas), las métricas para analizar las evaluaciones de los estudiantes (4 citas) y las métricas para medir la calidad de los artefactos producidos (3 citas).

11. La Figura **3-6** muestra que la mayoría de las técnicas de validación son encuestas; sin embargo, ésta es la técnica menos objetiva disponible, ya que los estudiantes no pueden analizar objetivamente la calidad del curso. En este sentido, se necesitan validaciones que impliquen la participación de las competencias adquiridas a partir de las necesidades de la industria.

12. La enseñanza de las arquitecturas de software es un tema de investigación actual. Aunque los investigadores han realizado aportaciones notables, la complejidad de la arquitectura, las expectativas de la industria, la transformación del concepto de arquitectura, la aparición de tecnologías recientes y los criterios de calidad hacen que aún deban explorarse muchas facetas de este tema. Algunos estudios proporcionan información sobre temas como el rol del arquitecto, las lecciones aprendidas y dan detalles sobre qué y cómo impartir un curso de arquitectura de software.

Los resultados de este mapeo sistemático pueden ayudar a los investigadores y educadores a entender los retos de la enseñanza de la arquitectura de software, proponer nuevos cursos que construyan habilidades en arquitectura alineadas con las necesidades de la industria, y mejorar los cursos existentes incorporando y combinando estrategias reportadas en la literatura que han sido efectivas bajo ciertas condiciones.

3.4. Trabajos más recientes reportados por la literatura

El mapeo sistemático realizado incluyó los años 2011 a 2021. A continuación se muestran los trabajos recientes a partir de marzo de 2021 hasta el 2023, reportados por la literatura relacionados con formación en arquitectura de software.

El trabajo de Nasin [81] presenta el diseño y la implementación del taller de Kata de Architectura[1] realizado en el curso "Calidad y Arquitectura de Software" como un ejercicio de grupo para enseñar a diseñar, documentar y evaluar la AS. Los comentarios sobre este taller recogidos de los estudiantes mediante un cuestionario, mostraron que, además de adquirir habilidades para diseñar y evaluar diseños arquitectónicos, el taller también sirvió para identificar requisitos no funcionales, elaborar supuestos y trabajar en equipo.

Eigler et al., [37] presentan un trabajo donde se recopilan herramientas de la literatura científica que puedan utilizarse para enseñar patrones de diseño de software y patrones de arquitectura de software. De las herramientas encontradas se presenta la descripción general, la presentación de las funcionalidades, el apoyo de las herramientas en la enseñanza, así como una clasificación de las funcionalidades en las cinco áreas: descripción, detección, selección, aplicación e implementación.

Dobslaw et al., [36] llevan a cabo una comparación entre las palabras clave utilizadas en los programas de cursos de educación superior y las ofertas de empleo, con el objetivo de examinar la brecha de conocimientos desde una perspectiva centrada en la tecnología. Se presenta un análisis de las tendencias tecnológicas en los puestos de trabajo en Suecia durante los últimos seis años. Los resultados revelan un aumento en la demanda de habilidades en tecnología de nube y automatización, como Kubernetes y Docker, según los anuncios de empleo, mientras que esta tendencia no es tan pronunciada en los programas de estudios de educación superior. Se destaca la disparidad en el lenguaje utilizado, ya que los planes de estudio académicos enfatizan más los conceptos, mientras que las ofertas de empleo se centran en las tecnologías específicas. Se exploran posibles soluciones para abordar esta discrepancia y se plantea la posibilidad de obtener conclusiones más profundas en investigaciones futuras.

Aldenhoven y Engelschall [5] discuten el concepto de belleza en la arquitectura de software basados en avances observados en otras disciplinas, como la arquitectura civil y la cirugía plástica. Los profesionales consideran que la arquitectura de software adquiere belleza cuando simplifica el esfuerzo necesario para trabajar con ella. Asimismo, destacan que la belleza en la arquitectura de software ejerce una influencia significativa en la felicidad y motivación de los desarrolladores que la implementan. Además, proponen enfoques para la creación y enseñanza de una arquitectura de software bella. En proyectos de tamaño medio y grande, enfocarse en la belleza de la arquitectura de software se revela como esencial para incrementar

[1] La kata de arquitectura está basada en las Katas de código, es un ejercicio en grupo que permite mejorar las habilidades de diseño de AS, permitiendo a los participantes ganar experiencia y recibir realimentación.

la satisfacción de los desarrolladores y, por ende, la calidad del producto.

Hidalgo et al., [49] presentan un trabajo que aborda los desafíos que hay en la implementación de los juegos de rol en el proceso de aprendizaje de la Ingeniería de Software. Los juegos de rol permiten que los estudiantes asuman roles específicos e interactúen entre sí, simulando escenarios del mundo real y aplicando sus conocimientos teóricos en un contexto práctico. Los juegos de rol se presentan una ayuda valiosa para que los estudiantes desarrollen habilidades clave, tales como trabajo en equipo, resolución de problemas y pensamiento crítico. Asimismo, contribuyen a comprender las complejidades y desafíos asociados con el desarrollo de software, fomentando la apreciación de la importancia de la colaboración y la comunicación efectiva. Las conclusiones principales destacan que la mayoría de los juegos de rol se emplean para enseñar habilidades relacionadas con el desarrollo de software y competencias blandas como el trabajo en equipo. Además, se observa una menor asociación con áreas como el diseño de software, aseguramiento de la calidad y gestión de procesos, en comparación con la gestión de proyectos o el aseguramiento de la calidad.

Alidoosti et al., [8] presentan un estudio que explora los aspectos éticos de la arquitectura de software, abordando temas como (i) la conciencia de los profesionales sobre cuestiones éticas, (ii) la identificación de las partes interesadas con posibles preocupaciones éticas, (iii) el reconocimiento de los valores éticos durante el proceso de diseño arquitectónico, (iv) las dificultades éticas que enfrentan los profesionales, y (v) la cuantificación y validación de los valores éticos. El objetivo fundamental de esta investigación es mejorar las consideraciones éticas en el diseño de la arquitectura de software.

Bartel et al.,[12] presentan la evaluación de la enseñanza invertida en la ingeniería de software, comparándolo con los métodos tradicionales de enseñanza. Se presenta un estudio a largo plazo que compara cursos con la metodología de enseñanza invertida y cursos que siguen métodos tradicionales. Durante un periodo de seis años, se analizaron un total de 11 cursos con una muestra de 157 participantes. Los resultados indican que la enseñanza invertida en ingeniería de software tiene un notable potencial y recibe una valoración más positiva por parte de los estudiantes en diversas áreas en comparación con la enseñanza tradicional.

Alidoosti et al., [7] presentan un juego basado en tarjetas llamado “Ethics-Aware DecidArch”, diseñado para ayudar a los arquitectos de software a reflexionar sobre las consideraciones éticas. Los primeros resultados, derivados de cuatro sesiones de juego, indican que el juego facilitó que los arquitectos de software (i) reflexionaran sobre diversas soluciones para abordar problemas éticos, (ii) tomaran decisiones éticas proporcionando razones fundamentadas, y (iii) reflexionaran sobre la operacionalización de los valores éticos y las posibles compensaciones asociadas.

En resumen, los trabajos más recientes reportados por la literatura reportan estrategias (como katas, aula invertida y herramientas) y temas como la ética, la belleza que ayudan a la formación en temas de AS. Sin duda estos temas aportan a la solución que se planteará

en capítulos posteriores.

Tabla 3-9.: Desafíos de la formación de arquitectos de software alineados con la industria del software.

No	Desafío	Referencias
1	Experiencia de los instructores en proyectos reales	[104]
2	Naturaleza abstracta y difusa	[66] [128] [67] [115] [118] [68] [65] [60]
3	Dificultad para implicar a clientes y expertos de la industria	[29]
4	ir más allá de la enseñanza tradicional	[97] [96] [99] [118] [85]
5	Los proyectos pequeños y sencillos son insuficientes para proporcionar a los estudiantes experiencia práctica	[97] [20] [110] [39]
6	Falta de uso de proyectos de código abierto	[110] [50]
7	Cada estudiante requiere un seguimiento personal	[116]
8	Las prácticas modernas desarrolladas en la industria tardan en llegar al mundo académico	[116]
9	Enseñar arquitecturas de software sigue siendo una tarea desafiante	[86] [1] [9] [124]
10	Los cursos deben adaptarse a los cambios	[73]
11	Los proyectos finalizados intimidan a los estudiantes	[123]
12	Brechas entre el aula y la industria	[57]
13	Claridad con las metodologías ágiles	[9]
14	Requisitos de los estudiantes	[65]

Tabla 3-10.: Métodos de validación

No	Método de validación	Referencias
1	Encuestas a estudiantes	[104] [66] [97] [20] [70] [68] [75] [69] [67] [86] [92] [125] [73] [57] [9] [118] [65] [22] [85] [44]
2	Métricas para medir el antes y después del curso.	[74]
3	Métricas para medir la calidad de los artefactos producidos.	[111] [98] [57]
4	Métricas para analizar las evaluaciones de los estudiantes.	[56] [65] [22] [55]
5	Comparar las competencias adquiridas en el curso con las que aparecen en la literatura..	[58]

4. Catálogo de patrones de formación en arquitectura de software

Resumen

En este apartado, se presenta un catálogo de patrones de formación concebido como respuesta a los desafíos asociados con la instrucción de arquitectos de software en el ámbito universitario. Considerando un conjunto definido de competencias esenciales para el desarrollo de arquitectos de software y tomando en cuenta tanto las estrategias identificadas en la literatura como las experiencias previas, hemos estructurado este conocimiento en forma de patrones de formación. Estos patrones no solo ofrecen una guía valiosa para los docentes en la planificación y ejecución de sus actividades de enseñanza, sino que también constituyen un recurso flexible. De este modo, los educadores pueden emplear este catálogo como una base de conocimiento desde la cual seleccionar y combinar los patrones que mejor se adapten a sus objetivos, facilitando así el diseño y la implementación de cursos de arquitectura de software a nivel universitario que potencien el desarrollo de competencias en los estudiantes de acuerdo con las expectativas de la industria del software.

4.1. Introducción

En esta sección presentamos un catálogo de patrones de formación en AS articulados en una guía que permiten a los profesores diseñar y ejecutar cursos, a nivel de pregrado. Estos patrones permiten desarrollar competencias de creación, evaluación y documentación de arquitecturas de software acordes con las expectativas de la industria de software. Los patrones de formación fueron extraídos de la revisión de la literatura del Capítulo 3, a partir de las distintas experiencias reportadas por profesores que proponen estrategias para formar sus estudiantes con habilidades en arquitectura de software.

En términos generales, un "patrón" se refiere a una solución general y reusable para un problema recurrente. Puede aplicarse en diversos contextos, desde la arquitectura y el diseño hasta la resolución de problemas en campos específicos, como la informática, la psicología, la biología, entre otros. Christopher Alexander dice: "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe el núcleo de la solución a ese

problema, de tal manera que se puede utilizar esta solución un millón de veces, sin hacerlo nunca de la misma manera dos veces" [6].

Inspirados en el concepto de patrones de software, patrones de diseño y patrones de arquitectura de software, se propone el concepto de patrones de formación. Los patrones de formación son estrategias reutilizables para problemas comunes en el diseño y ejecución de cursos de arquitectura de software con el fin de lograr competencias en AS. Estos patrones representan buenas prácticas y experiencias probadas por la comunidad de profesores y proporcionan soluciones estructuradas y eficientes para situaciones específicas. Cada patrón describe un problema que ocurre con frecuencia en un contexto determinado y presenta una solución general que puede ser adaptada y aplicada a diferentes escenarios. En este capítulo presentamos siete patrones de formación y su respectivo lenguaje.

De esta forma intentamos resolver la pregunta ¿Cómo lograr adecuadamente el desarrollo de competencias relacionadas con la creación, evaluación y documentación de arquitecturas de software en potenciales egresados de programas de informática y afines, mediante una guía de diseño y desarrollo de cursos basada en patrones de formación?

4.2. Metodología para extraer los patrones de formación de AS

La finalidad subyacente de los patrones de formación reside en la organización de estrategias con el objetivo de cultivar las diversas competencias en AS. A pesar de la intrincada naturaleza de este proceso, no existe una fórmula mágica para su extracción; no obstante, es factible discernir estrategias de formación recurrentemente empleadas, alineadas con competencias pertinentes a un contexto específico. Dichas estrategias, al adaptarse a fuerzas inherentes al diseño y ser efectivas únicamente cuando estas fuerzas están en equilibrio, han sido consagradas como patrones. Estos patrones se han catalogado de manera sistemática, proporcionando a los docentes y formadores una referencia accesible para consultar, seleccionar y aplicar en sus cursos de Arquitecturas de Software y disciplinas afines.

Los patrones de formación fueron extraídos de la revisión de la literatura a partir de los reportes de experiencias de cursos de arquitectura de software. En dicha revisión buscamos problemas de formación recurrentes y las soluciones encontradas y experimentadas por los profesores.

Inspirados en los objetivos de los patrones de diseño de la GoF, se definieron los siguientes objetivos para los patrones de formación:

- Proporcionar catálogos de elementos reusables en el diseño de cursos de AS.

- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores de cursos de AS.
- Estandarizar el modo en que se realiza el diseño de cursos.
- Facilitar el aprendizaje de las nuevas propuestas de cursos de AS condensando conocimiento ya existente.

Para llegar a los patrones de diseño se siguieron los siguientes pasos:

1. Se hizo la búsqueda de las competencias en AS que la industria de software demanda de los recién egresados.
2. Se hizo la búsqueda de las experiencias de formación en arquitecturas de software a partir de un mapeo sistemático previo.
3. Se definió el formulario para especificar los patrones.
4. Se realizó la extracción de los patrones a partir de unos criterios definidos.

A continuación explicamos los detalles de los pasos anteriores.

4.2.1. Búsqueda de las competencias en AS que la industria de software demanda de los recién egresados

Para identificar y documentar las competencias de AS, se realizó un ciclo de investigación-acción, en el cual se diseñó un estudio basado en encuestas y talleres en el que participaron ingenieros de software de la industria y profesores universitarios que imparten cursos relacionados con el diseño y evaluación de la AS. Este ciclo de investigación fue publicado en el artículo "Visión de las competencias en arquitectura de software integrando las perspectivas de la industria y la academia" [90]. La Tabla **4-1** muestra los instrumentos específicos que utilizados para recolectar los datos. Cada instrumento permitió ir encontrando y refinando la lista de competencias.

Al final, cada una de las competencias encontradas se les asignó un identificador de la forma: C01, C02, C03, etc. Además se las clasificó en tres grupos: obligatorias, opcionales y fuera del alcance para un curso de pregrado tal como se aprecia en la Figura **4-1**. El listado completo de competencias está en el Apéndice C.

Tabla 4-1.: Instrumentos de recolección de datos en la búsqueda de competencias de AS.

No	Instrumento	Objetivo
1	Revisión de la literatura	Recolectar la lista de competencias de un arquitecto de software utilizando la revisión de la literatura.
2	Encuesta de valoración a docentes	Valorar la lista de competencias según el criterio de los docentes.
3	Encuesta de valoración a ingenieros	Valorar la lista de competencias según el criterio de la industria.
4	Workshop con ingenieros de la industria	Clasificar las competencias mejor valoradas por la industria en tres grupos: obligatorias, opcionales y fuera de alcance de un curso de pregrado.
5	Encuesta de competencias obligatorias a docentes	Valorar con la academia las competencias clasificadas como obligatorias.
6	Focus group con docentes	Debatir la validez la lista de competencias obligatorias encontradas.

4.2.2. Búsqueda de las experiencias de formación en arquitecturas de software

A partir del mapeo sistemático realizado entre los años 2011 y 2023 que buscaba los desafíos en la enseñanza de la AS y las estrategias de formación utilizadas por los docentes, se filtró aquellos artículos que describen únicamente experiencias de cursos de arquitectura de software. El mapeo sistemático arrojó un total de 56 estudios primarios, de los cuales se hizo un filtró obteniendo 34 artículos que hablan específicamente de experiencias en formación de AS. Estos 34 trabajos fueron la fuente a partir de la cual se extrajeron los patrones de formación de AS y se pueden apreciar en el Apéndice B.

Se derivaron las estrategias recurrentes y las relaciones de equilibrio, basándonos en las experiencias acumuladas de las experiencias reportadas. Este proceso se llevó a cabo al considerar las fuerzas presentes en los contextos de uso. De esta manera, se logró la identificación de los siete patrones de formación, cada uno de los cuales se construyó a partir de la comprensión de las dinámicas y las interacciones específicas observadas en el desarrollo de los cursos de AS.

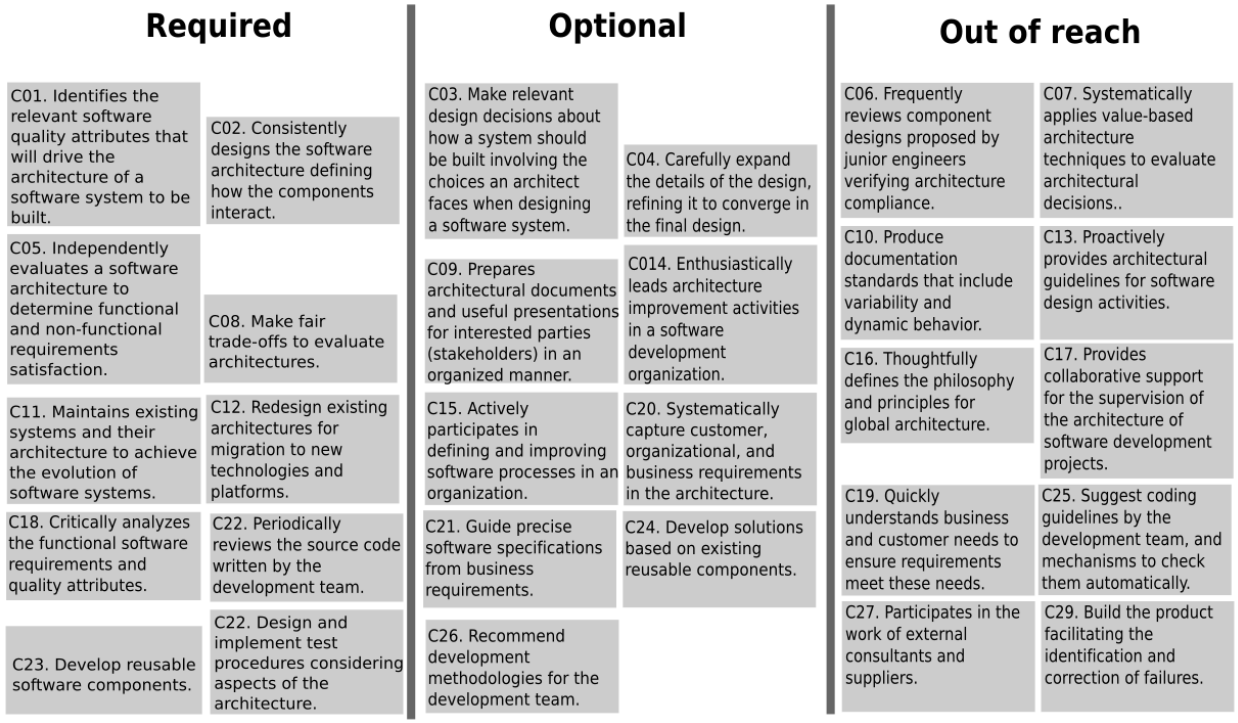

Figura 4-1.: Clasificación final de las competencias en AS

4.2.3. Definición de la estructura interna para especificar los patrones

Una vez identificadas las experiencias e identificados los patrones recurrentes, la tarea fue encontrar una manera de estructurar el conocimiento derivado. Se buscó y comparó distintas maneras de especificar patrones en áreas similares a nuestro objeto de estudio, específicamente se tuvo como base patrones de software. Los patrones de software generalmente se documentan en varias formas. Estos formularios se conocen como esquemas de patrones, formularios de patrones o plantillas de patrones [83]. Varios ejemplos de estas plantillas se pueden encontrar en la literatura, dos de las más representativas son la propuesta por "Gang of Four" (GoF) [40] y la forma de "Pattern-Oriented Software Architecture" (POSA) [15]. La Tabla **4-2** muestra las plantillas utilizadas por distintos autores.

Tabla 4-2.: Plantillas de patrones

Título del artículo/Libro	Plantilla
Design Patterns: Elements of Reusable Object-Oriented Software 1st Edition [40]	Name, Intent, Also known as, Motivation, Applicability, Structure, Participants, Collaborations, Consecuences, Implementation, Sample code, Know uses, Related patterns
Pattern-Oriented Software Architecture [15]	Name, Brief, Example, Context, Problem, Solution, Structure, Dynamics, Implementation, Example resolved, Known uses, Consequences, See also
Design patterns in communications software [101]	Problem, Forces, Solution, Rationale, Resulting Context, Example
Patterns for Computer-Mediated Interaction [107]	Intent, Context, Problem, Scenario, Symptoms, Solution, Dynamics, Rationales, Check, Danger spots, Know uses, Related patterns

Coplien & Schmidt [27] y muchos otros investigadores afirman que una forma de presentación de patrones consta de al menos las tres secciones Problema, Contexto y Solución. La sección *Problema* describe de manera concisa el problema en cuestión, la sección *Contexto* describe las situaciones en las que ocurre el problema, así como las fuerzas y restricciones que surgen para una posible solución, y la sección *Solución* describe cómo resolver las fuerzas dentro de ese contexto.

Teniendo en cuenta las plantillas que proponen otros autores, y los campos obligatorios sugeridos por Coplien & Schmit, se propuso una estructura para describir los patrones de formación. Se tomaron los campos que se adaptan a nuestros patrones de formación; algunos campos son nuevos y no provienen del mundo de los patrones de software. La estructura se muestra a continuación:

- **Nombre**: Permite identificar el patrón y usarlo en un potencial lenguaje de patrones.
- **Contexto problemático**: Expone el contexto del problema con el propósito de reafirmar la selección y comprender la raíz de las causas que originan dicho problema.
- **Fuerzas**: Las fuerzas se refiere a los factores o consideraciones que influyen en la aplicación de un determinado patrón. Representan un escenario concreto que motiva al uso del patrón. Un sinónimo sería “cuando usar el patrón”.
- **Solución**: Describe la solución al problema aplicando el patrón.

- **Ejemplo de uso del patrón**: Permite ilustrar cómo el patrón es utilizado para un caso particular.
- **Competencias que se abordan**: Lista de competencias que se desarrollan aplicando el patrón.
- **Variantes para llevarlo a la práctica**: Describe los casos alternativos o situaciones sobre cómo aplicar el patrón.
- **Requisitos previos**: Permite establecer si se cuenta o no con las condiciones para la aplicación del patrón o previamente se deben abordar otras competencias antes de aplicar el patrón.
- **Ventajas**: Consecuencias positivas que trae aplicar el patrón.
- **Desventajas**: Consecuencias negativas que trae aplicar el patrón.
- **Patrones relacionados**: Permite al momento de usar un patrón llevar de manera natural hacia otros patrones que pueden complementar el proceso formativo. Indica las conexiones y relaciones entre los distintos patrones de formación. Estas relaciones ayudan a los diseñadores y desarrolladores a comprender cómo pueden combinarse y aplicarse juntos distintos patrones.
- **Experiencias reportadas del uso del patrón**: Referencias a experiencias reportadas por la literatura en las cuales se evidencia que el patrón ha sido utilizado.

4.2.4. Extracción de los patrones

En esta etapa se hizo la lectura detallada de los 34 artículos seleccionados, tratando de identificar los patrones de formación. Para ello se creó la tabla con los campos: Número, titulo del artículo, problema que pretenden resolver, resumen de las estrategias principales, frase identificadora (una frase fácil de recordar que resuma e identifique el patrón). El propósito de esta tabla es tener un resumen que permita comparar los problemas y las soluciones encontradas. La Tabla **4-3** muestra las cinco primeras filas de este primer paso (El documento completo se puede acceder en este enlace).

Tabla 4-3.: Extracción de los patrones de formación con cinco primeras filas del proceso

No	Título	Problema	Resumen	Frase identificadora
1	Teaching software architecture to undergraduate students: an experience report	Cómo proponer una arquitectura de software que favorezca un atributo de calidad	2009-2012: Los estudiantes hicieron 6 proyectos de 1 a 2 semanas de duración, cada proyecto atacando un atributo de calidad. Por ejemplo: hacer que la interfaz de usuario se ejecute dos veces más rápido; hacer una mejora significativa de la confiabilidad y demostrar que mejoró la disponibilidad del sistema.	Proyectos cortos que ataquen un atributo de calidad
2	Extensive Evaluation of Using a Game Project in a Software Architecture Course	Como motivar a los estudiantes a aprender arquitectura de software	Se enseña AS utilizando un proyecto de desarrollo de juegos. Un proyecto de desarrollo de software que se centra en la arquitectura de software. La principal ventaja es que los estudiantes están aprendiendo arquitectura de software a través de la realización de un proyecto completo donde pueden ver los resultados de su diseño arquitectónico como producto.	Proyecto grande que motive a los estudiantes
3	Using public and free platform as a service (PaaS) based lightweight projects for software architecture education	Cómo ayudar a los estudiantes a acumular experiencia en arquitectura de software	El artículo plantea una estrategia educativa de utilizar proyectos ligeros con recursos PaaS públicos y gratuitos (1) para ayudar a los estudiantes a acumular experiencia arquitectónica desde la etapa inicial y (2) para facilitar el fortalecimiento los conocimientos fundamentales de arquitectura de los estudiantes.	Proyectos cortos con repositorio de proyectos
4	Exploration on Theoretical and Practical Projects of Software Architecture Course	Cómo enfrentar a los estudiantes con proyectos similares al mundo de la industria?	Los estudiantes practican con sistemas de software industrial open-source, que son complejos y revelan diferentes puntos de vista de las partes interesadas. Cada grupo analizó no solo el código del software, sino también los documentos del software.	Proyectos de análisis de la arquitectura de sistemas open-source
5	Did our Course Design on Software Architecture meet our Student's Learning Expectations?	Cómo plantear talleres relacionados con la industria que puedan guiar y hacer que los participantes pasen por los procesos de pensamiento de un arquitecto junior	Los estudios de caso son escenarios de la vida real, que permiten a los estudiantes analizar, aplicar los conceptos enseñados y mitigar las posibles compensaciones dentro de un contexto realista.	aprendizaje basado en casos

El siguiente paso fue agrupar las experiencias que fueran similares teniendo en cuenta el campo *Frase Identificadora* de la tabla Tabla **4-3**. Cada grupo detectado lo marcamos con un color distintivo dentro de la tabla Tabla **4-3** y obtuvimos los siguientes grupos:

- Aprendizaje basado en proyectos largos
- Aprendizaje basado en proyectos cortos
- Aprendizaje basado en proyectos open-source
- Aprendizaje basado en casos

- Aprendizaje basado en juegos

Una vez clasificadas las experiencias por colores (o grupos), se intuía que cada color potencialmente representaba uno o varios patrones de formación. Se aseguró que cada grupo cumpliera con los siguientes criterios:

1. Que el grupo representara una experiencia replicable en otro contexto universitario.
2. Que las experiencias reportadas en cada grupo tuvieran una explicación en un grado de detalle considerable.
3. Que hubiera al menos dos trabajos por cada grupo.

En seguida, se empezó a trabajar en detalle cada grupo, leyendo y tomando elementos de las experiencias. Se analizó nuevamente cada grupo de experiencias especialmente sus semejanzas, diferencias; se leyó nuevamente el texto de cada artículo y se empezó a llenar la plantilla de los patrones. Tras realizar este paso, surgieron nuevos conjuntos de patrones. Del estudio del aprendizaje basado en casos, se manifestó un nuevo grupo vinculado a la resolución de problemas. En cuanto al aprendizaje basado en proyectos de código abierto, se desglosó en dos categorías: una asociada a la contribución en un proyecto de código abierto y otra que emplea un proyecto de código abierto dentro del entorno del aula. En primera instancia se llenó los campos: nombre, contexto problemático y solución. De esta forma se obtuvo siete patrones de formación que se pueden ver en la Tabla Tabla **4-4**:

Tabla 4-4.: Patrones de formación encontrados

No	Nombre patrón en inglés	Nombre patrón en español
1	Mini-Projects-based training	Formación basada en mini-proyectos
2	Large Project-based Training	Formación basada en proyectos largos
3	Open-source projects-based training	Formación basada en proyectos open-source
4	In-house project-based training	Formación basada en Proyectos In-house
5	Cases-based training	Formación basada en casos
6	Problem-solving-based training	Formación basada en la resolución de problemas
7	Games-based training	Formación basada en juegos

4.3. Catálogo de patrones

El catálogo de patrones de formación que se presenta a continuación es una recopilación estructurada de soluciones probadas y reutilizables a problemas recurrentes de diseño de cursos de arquitectura de software. Estos patrones presentan las mejores estrategias y experiencias reportadas por la comunidad de docentes según la revisión de la literatura. Cada patrón describe un problema específico que ocurre en el proceso de diseño de cursos de AS y proporciona una solución general y flexible para abordarlo. A continuación explicamos en detalle cada uno de los patrones de formación propuestos en esta investigación.

4.3.1. Patrón 1: Mini-Projects-based training

Nombre: Mini-Projects-based training.

Contexto problemático: Cuando los estudiantes llegan a su primer curso de arquitectura de software ya han visto cursos relacionados con programación y desarrollo de software. En este nivel los estudiantes tienen habilidades en la creación de aplicaciones sencillas, pero trabajando únicamente con requisitos funcionales. Para desarrollar una arquitectura de software el profesional tiene que prestar atención a los atributos de calidad (escalabilidad, desempeño, seguridad, etc.), además de satisfacer los requerimientos funcionales. La industria de software espera entonces que los arquitectos de software diseñen la estructura de las aplicaciones, de tal forma que satisfagan los atributos de calidad y restricciones del sistema [13]. De acuerdo con Sherman y Unkelos-Shpigel [109], el arquitecto es responsable del diseño y las decisiones técnicas en el proceso de desarrollo de software, tiene la función de resolver un problema definiendo las estructuras de un sistema que pueda ser implementado utilizando ciertas tecnologías. Sin embardo, encontrar el balance adecuado entre atributos de calidad del software como seguridad, desempeño, usabilidad, disponibilidad, mantenibilidad, interoperabilidad, entre otros, es una tarea muy compleja. Estos atributos pueden entrar en conflicto unos con otros, por ello el arquitecto de software requiere conocer tácticas, patrones y principios que le ayuden a tomar decisiones correctas [67].

Al mismo tiempo, los proyectos de software son cada vez más exigentes. Se requiere que los sistemas sean fáciles de usar, brindando una buena experiencia de usuario; que funcione en diferentes dispositivos incluso móviles; que sea seguro y mantenga la privacidad; que pueda integrarse a otros sistemas y que facilite la interoperabilidad; que pueda procesar grandes volúmenes de datos; etc. Todo esto hace que los cursos relacionados con la arquitectura de software tienen que brindar más experiencias parecidas a las reales para que los estudiantes se preparen a esas exigencias. Enseñar arquitectura de software para trabajar con proyectos similares al mundo real implica que los estudiantes deben conocer y aplicar nuevos temas

como los atributos de calidad, estilos y tácticas de arquitectura.

Desde esta perspectiva, los estudiantes aún no cuentan con suficientes habilidades en el desarrollo de software para enfrentarse a proyectos de desarrollo exigentes y en dominios complejos [20]. Los estudiantes requieren sumar experiencia para adaptarse las situaciones que se pueden plantear en los proyectos actuales de la industria del software.

Por otro lado, el docente busca desarrollar un curso de arquitectura de software en el que se espera conectar al estudiante con los fundamentos teóricos y prácticos sobre las arquitecturas de software. El objetivo es los estudiantes puedan ganar confianza en el área sin tener que lidiar con toda la complejidad de un sistema grande. Las habilidades más importantes para desarrollar estarán relacionadas con identificar y especificar atributos de calidad del sistema software, elegir los estilos de arquitectura que favorezcan esos atributos de calidad y diagramar la arquitectura. Se busca que los estudiantes desarrollen una experiencia concreta de diseño arquitectónico teniendo en cuenta las restricciones de tiempo y de recursos tanto de los estudiantes como de la infraestructura que cuentan las universidades.

Fuerzas:

- Mediante la aplicación de estilos y tácticas arquitectónicas, el profesor quiere formar a los estudiantes para que logren cumplir los atributos de calidad del software (como escalabilidad, rendimiento y seguridad).
- La industria espera que los estudiantes sepan cómo favorecer atributos de calidad específicos mediante la aplicación de estilos arquitectónicos.
- El docente quiere desarrollar un curso práctico que desarrolle las habilidades de los estudiantes en la construcción de un nuevo sistema de software con una buena arquitectura buscando un equilibrio entre amplitud y profundidad de conocimientos.
- Los estudiantes no tienen experiencia trabajando en proyectos de desarrollo complejos. Por el contrario, a través de pequeños proyectos, pueden aprender a diseñar la arquitectura de un sistema y ganar experiencia para afrontar futuros proyectos de diseño más extensos y complejos.

Solución: El docente trabaja su curso con miniproyectos para que los estudiantes puedan llevar a la práctica los conceptos de patrones de arquitectura y tácticas de arquitectura para favorecer el cumplimiento de los atributos de calidad. [104, 66].

Los estudiantes desarrollan en equipo varios proyectos cortos de una o dos semanas de duración cada uno. Cada miniproyecto favorece principalmente un atributo(s) de calidad. Por ejemplo, hacer un miniproyecto que favorezca la modificabilidad aplicando principios de diseño y una arquitectura en capas; luego desarrollar un miniproyecto que favorezca el desempeño de la aplicación, luego otro miniproyecto que facilite la escalabilidad y otro que priorice la seguridad [104]. Los proyectos no deben ser complejos, pero se parte de la

premisa que favorecen la comprensión de los atributos de calidad y las tácticas de arquitectura involucradas en los estilos arquitectónicos usados [66].

Es recomendable que los proyectos y ejemplos estén desarrollados en tecnologías con los que los estudiantes y profesores estén familiarizados desde sus cursos previos. El docente puede disponer de ejemplos de proyectos para cada atributo de calidad, los cuales pueden ser consultados por los estudiantes en un repositorio de ejemplos.

Al final del curso, los estudiantes tendrán varias versiones del producto software y cada versión favoreciendo un atributo de calidad. La Figura **4-2** muestra los principales elementos de la solución del patrón.

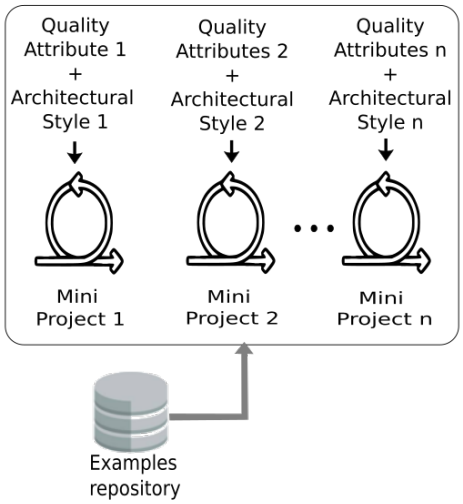

Figura 4-2.: Small Project-based Training

Cada proyecto corto puede describirse en un documento con la siguiente estructura:

- Introducción o contexto del sistema de software a desarrollar.
- Objetivos de aprendizaje a desarrollar en los estudiantes (o competencias).
- Descripción de los requisitos funcionales.
- Descripción de los requisitos no funcionales.
- Definición de los entregables.
- Rúbrica de evaluación del proyecto.

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos

Deseables:

- Sistemas operativos
- Sistemas Distribuidos
- Ingeniería de Software

Ejemplo de uso del patrón: El docente puede elegir la cantidad de miniproyectos que quiere realizar en su curso, puede ser entre dos y seis (dependiendo de la duración del curso en semanas). El primer miniproyecto se puede abordar con una estructura monolítica con una arquitectura en **capas** que favorezca el atributo de calidad de **modificabilidad**. Los estudiantes tendrán que aplicar un diseño en capas usando principios y algunos patrones de diseño de software. El docente puede además, suministrarles un proyecto de ejemplo para que los estudiantes tengan un referente en qué basarse.

El segundo miniproyecto, el profesor puede trabajar con una arquitectura **microkernel** para favorecer los atributos de calidad de **extensibilidad** y **modularidad**. El patrón arquitectónico de micronúcleo también se conoce como patrón arquitectónico de plugins. Por lo general, se usa cuando los equipos de software crean sistemas con componentes intercambiables. Se aplica a los sistemas de software que deben poder adaptarse a los requisitos cambiantes del sistema. Separa un núcleo funcional mínimo de la funcionalidad ampliada y las piezas específicas del cliente. La arquitectura también sirve como enchufe para conectar estas extensiones y coordinar su colaboración.

El tercer miniproyecto, el profesor puede trabajar con una arquitectura **cliente/servidor**, para facilitar el proyecto se puede plantear un sólo servidor atendiendo peticiones simultáneas de varios clientes. Se puede medir en este caso el atributo de **desempeño**.

El cuarto miniproyecto, el profesor puede trabajar la arquitectura **dirigida por eventos** con una topología de intermediario (broker) que favorecen los atributos de calidad de **desempeño**, **escalabilidad** y **tolerancia a fallos**. Esta topología es útil cuando tiene un flujo de procesamiento de eventos relativamente simple y no necesita orquestación y coordinación central de eventos. Como tecnología de intermediario de mensajes livianos se puede utilizar tecnologías como RabbitMQ, ActiveMQ, HornetQ, etc.

El quinto miniproyecto, el profesor puede trabajar una arquitectura de **microservicios** que favorecen los atributos de **escalabilidad** y **elasticidad** debido a que la aplicación monolítica se divide en pequeñas aplicaciones que son independientes y se comunican entre si. Recomendamos trabajar con algún framework especializado en microservicios para hacer más fácil la implementación. En el caso de java se pueden utilizar frameworks como Spring

Boot, MicroProfile, WildFly Thorntail, Cricket, etc.

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Apéndice C: Tabla de Competencias):

Competencias obligatorias: C2, C5, C8, C12, C18, C23.

Competencias opcionales: C3, C4, C9, C14, C21.

Variantes para llevarlo a la práctica: Una forma de abordar la formación en AS es plantear con un proyecto grande (con muchos requisitos) y abordar un conjunto pequeño de requisitos de alta prioridad para el cliente en cada miniproyecto e ir cambiando los atributos de calidad. De esta manera cada miniproyecto requerirá rediseño e involucrar nuevos estilos arquitectónicos.

Ventajas:

- Los estudiantes tendrán la oportunidad de experimentar con varios estilos de arquitectura y ver su aplicabilidad por medio de pequeños proyectos de desarrollo.

Desventajas:

- Los estudiantes no participan en un proyecto del todo real con clientes reales.
- Este patrón obliga al docente tenga una base de código fuente y documentación de los proyectos para evitar que los estudiantes gasten demasiado tiempo resolviendo aspectos de implementación desde cero.

Patrones relacionados:

El patrón Architecture-style-driven Mini-Project pattern puede combinarse en un curso de SA con otros patrones de formación que son livianos de aplicar para docentes y estudiantes y permiten desarrollar diversas competencias para la toma de decisiones. Por ejemplo, Case and Flipped Classroom Training, Problem-solving-based training, y Games-Based Training.

Experiencias reportadas del uso del patrón:

- Teaching Software Architecture to Undergraduate Students: An Experience Report [104] muestra cómo trabajar con proyectos pequeños semi-reales.
- Using Public and Free Platform-as-a-Service (PaaS) based Lightweight Projects for Software Architecture Education [66] muestra una estrategia educativa que utiliza proyectos ligeros con recursos PaaS públicos y gratuitos de tal forma que los estudiantes aprendan de manera colaborativa y experimental.

- Lean learning: Applying lean techniques to improve software engineering education [20] muestra que uno de los enfoques para la formación en AS es a través de ejercicios prácticos.

4.3.2. Patrón 2: Large Project-based Training

Nombre: Large Project-based Training.

Contexto problemático: Los estudiantes tienen competencias suficientes en desarrollo de software para afrontar proyectos de desarrollo medianos y grandes y en dominios complejos [20]. Por tanto, los estudiantes se encuentran en un nivel intermedio de formación en temas de arquitectura de software.

De esta forma, el profesor pretende desarrollar un curso de arquitectura de software para conectar a los estudiantes con los fundamentos teóricos y prácticos de la arquitectura de software. Esta situación les permitirá ganar confianza en el área a través de un gran proyecto de suficiente complejidad que resuelva un problema empresarial. Es importante destacar que el profesor debe tener experiencia y preparación en el abordaje de proyectos de software de mayor complejidad para guiar a sus estudiantes.

Fuerzas:

- El docente quiere formar a los estudiantes en arquitectura de software a través de un proyecto real completo de cierto grado de complejidad, trabajando en un dominio desconocido y donde los estudiantes puedan ver los resultados de su diseño arquitectónico como un producto.
- El docente quiere desarrollar un curso práctico para que los estudiantes comprendan y apliquen la teoría de la AS.
- La industria espera que los estudiantes sepan diseñar la arquitectura de un sistema de software real y complejo, sean capaces de trabajar en equipo tomando decisiones y vean las consecuencias de las mismas.
- Los estudiantes tienen cierta experiencia trabajando en proyectos de desarrollo complejos y necesitan desarrollar habilidades de toma de decisiones en equipo, trabajando en proyectos de complejidad similar a los del mundo real. Esta situación implica que el proyecto tiene varios requisitos funcionales y no funcionales, restricciones y una adecuada gestión.
- Los estudiantes al ser un único proyecto tendrán menos posibilidades de explorar más estilos de arquitectura.

Solución: Los estudiantes aprenden arquitectura de software a través de un proyecto completo real, donde pueden ver los resultados de su diseño arquitectónico como un producto [120]. Se recomienda trabajar en equipos entre 3 y 5 estudiantes. Es esencial aclarar que la evaluación de las contribuciones iguales de los miembros de un equipo no es un objetivo primordial de este modelo. La evaluación de las contribuciones de los equipos es un reto más amplio que abarca varios modelos y enfoques.

Para trabajar con clientes reales proporcionando escenarios y problemas reales significativos y realistas para los proyectos se requiere que la Universidad tenga convenios con empresas [29] y disponer de un banco de proyectos. Muchas universidades disponen de convocatorias semestrales donde las empresas postulan ideas de proyectos y los docentes evalúan y eligen los proyectos adecuados. En este caso, los estudiantes tendrán la oportunidad de hacer ingeniería de requisitos de manera completa, especificar atributos de calidad y aplicar métodos para concertar con los stakeholders los atributos que definirán la arquitectura [104]. Los clientes reales podrían participar de la captura de requisitos, priorización de requisitos en cada entrega y asistir a las reunión de entrega de cada iteración.

Es importante que el proyecto a elegir debe ser lo suficientemente acotado para alcanzar a ser trabajado en un periodo académico. Dependiendo de las habilidades que tengan los estudiantes se pueden elegir proyectos de baja, alta o mediana complejidad arquitectural [9]. El docente debe tener la intuición y la experiencia de qué tipo de proyectos pueden ser factibles acorde al nivel de sus estudiantes. La Figura **4-3** muestra los principales elementos de la solución del patrón.

En los proyectos pueden participar clientes reales o expertos de la industria que proporcionan requisitos y evalúan demostraciones. Además de ayudar con la carga de trabajo de evaluación, los expertos de la industria proporcionan a los estudiantes retroalimentación práctica basada en las mejores prácticas actuales de la industria [123]. Por lo tanto, es necesaria una relación entre la universidad y la empresa. Esta relación puede crearse aprovechando que algunos clientes son egresados de las mismas instituciones educativas [104].

Las universidades pueden firmar acuerdos con empresas en busca de socios. Inicialmente, las empresas entran en un periodo de prueba, tras el cual se convierten en socios que apoyan el proceso educativo [29].

Es habitual que los clientes se muestren reacios a participar en los proyectos de clase porque disponen de poco tiempo debido a sus horarios de trabajo. El proyecto podría garantizar que la participación del cliente sea manejable para su tiempo. Estas alternativas podrían incluir actualizaciones periódicas del estado del proyecto, sesiones de realimentación bien estructuradas o la utilización de representantes de la organización que puedan servir de enlace con los estudiantes.

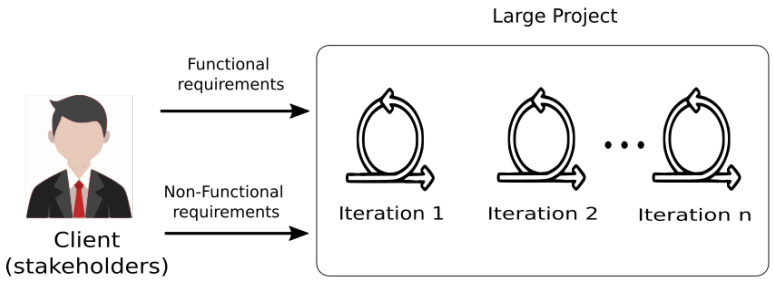

Figura 4-3.: Large Project-based Training

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos
- Desarrollo web a pequeña escala
- Desarrollo movil a pequeña escala

Deseables:

- Sistemas operativos
- Sistemas Distribuidos
- Ingeniería de Software

Ejemplo de uso del patrón: A continuación se describe un ejemplo de un proyecto real. La Cruz Roja de Colombia necesita desarrollar un sistema de software que permita conectar a las personas con los recolectores de sangre en el momento y lugar adecuados [86]. El objetivo fundamental es ayudar a salvar vidas humanas. Se requiere diseñar una aplicación web en la que las personas se registran y brindan información básica, como el tipo de sangre y la zona de residencia. El algoritmo a desarrollar notificará automáticamente a los donantes cuando su sangre sea necesaria en su área y los donantes pueden reservar una fecha para la donación de sangre. Para el centro de salud, la aplicación ofrece un seguimiento de todas las citas y proporciona un canal para notificar sobre la necesidad de sangre. La aplicación debe tener un sistema de autenticación y autorización seguros y debe ser escalable en caso que se empiece a usar en otras ciudades de todo el país.

En el proyecto descrito anteriormente, es un ejemplo de un proyecto real. Los estudiantes tendrán que interactuar con clientes reales, capturar requisitos funcionales y no funcionales, proponer una arquitectura a partir de los atributos de calidad seleccionados como drivers,

tomar decisiones sobre qué tecnología es la más adecuada (y otro tipo de decisiones), evaluar la arquitectura elegida y hacer una implementación por iteraciones.

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Apéndice C: Tabla de Competencias):

Competencias obligatorias: C1, C2, C5, C8, C18, C22, C23.

Competencias opcionales: C3, C4, C9, C20, C24.

Variantes para llevarlo a la práctica: Ninguna.

Ventajas:

- Los estudiantes tendrán la oportunidad de trabajar con un proyecto de la vida real, con clientes reales y desarrollar varias habilidades en AS.
- Trabajar en proyectos reales con clientes reales podría permitir a los estudiantes recibir una compensación económica o estar vinculados a trabajar con la empresa en un futuro cercano.

Desventajas:

- Los estudiantes no podrán experimentar con muchos estilos de arquitectura, pues para el proyecto elegido tendrán que elegir el estilo más adecuado.
- Trabajar con clientes reales involucra tener alianzas entre la Univesidad y las empresas.
- No es fácil involucrar clientes reales en los proyectos de clase, pues los empresarios son personas bastante ocupadas [29].

Patrones relacionados: El patrón Large Project-based Training pattern se puede combinar en un curso de AS con otros patrones de formación que son livianos de aplicar para docentes y estudiantes y permiten desarrollar varias competencias en toma de decisiones. Por ejemplo, Case and Flipped Classroom Training, Problem-solving-based training, y Games-Based Training.

Experiencias reportadas del uso del patrón:

- Exploration on Theoretical and Practical Projects of Software Architecture Course [128] presenta cómo proyectos teóricos y prácticos pueden ayudar a mejorar la capacidad para analizar y diseñar una AS y experiencias relevantes en sistemas a gran escala.

- Extensive Evaluation of Using a Game Project in a Software Architecture Course [120] presenta cómo un proyecto de desarrollo de software en el área de juegos puede ayudar en la enseñanza de la AS.
- Using game development to teach software architecture [104] describe un caso de estudio donde el desarrollo de un juego ayudó a enseñar AS.
- Comparison of learning software architecture by developing social applications versus games on the android platform [127] describe un estudio empírico en el que el objetivo ese descubrir diferencias y similitudes en estudiantes que trabajan en el desarrollo de aplicaciones sociales frente a estudiantes que trabajan en el desarrollo de juegos.
- Scrum as a Method of Teaching Software Architecture [122] muestra un caso donde el uso de scrum ayudó a crear un buen ambiente de trabajo entre los estudiantes cuando desarrollan un proyecto de software complejo.
- Designing and applying an approach to software architecting in agile projects in education [9] presenta un enfoque para la introducción de actividades de arquitectura de software en un curso de proyecto ágil.
- Fontys ICT, Partners in Education Program: Intensifying Collaborations Between Higher Education and Software Industry [29] presenta un modelo educativo impulsado por las necesidades e intereses de los estudiantes, aprendiendo a través de la cooperación con otros estudiantes y proporcionando escenarios y problemas industriales significativos y realistas para los proyectos.
- A Community of Learners Approach to Software Architecture Education [30] muetra un enfoque de enseñanza de AS en el que los estudiantes son tratados como socios en el proceso de desarrollo del conocimiento.
- Agile architecture in action (AGATA) [80] presenta un marco que ayuda a escalar métodos ágiles con equipos más grandes.

4.3.3. Patrón 3: Open-source projects-based training

Nombre: Open-source projects-based training.

Contexto problemático:

Utilizar proyectos de software de código abierto en un curso de ingeniería de software tiene muchas ventajas. Por ejemplo, permite a los estudiantes aprender buenas prácticas de codificación a partir de proyectos del mundo real y les da una visión de un proyecto real. No obstante, es difícil para instructores y estudiantes contribuir a dichos proyectos. Uno de los

primeros retos es identificar y seleccionar el proyecto adecuado con el tamaño y la complejidad adecuados. Otros retos son la inexperiencia de los estudiantes, la duración limitada del curso y las prácticas informales para desarrollar el producto [51].

Cuando los recién graduados se unen a la industria de software, uno de los desafíos iniciales que enfrentan es desarrollar componentes de software en proyectos existentes y generalmente grandes [123]. En el argot de Ingeniería de Software, esto se conoce como un "escenario de brownfield", en contraposición a un "escenario de greenfield" en el cual un equipo de ingeniería de software comienza a desarrollar un proyecto desde cero. Los estudiantes prefieren abordar escenarios con proyectos nuevos en lugar de darle continuidad a anteriores desarrollos. En tales escenarios los estudiantes tienen más confianza pues solo necesitan comprender los requisitos y luego tienen un control completo sobre la arquitectura, el diseño y la estructura del sistema de software. Por el contrario, los escenarios de proyectos existentes tienden a intimidar a los estudiantes, pues requieren habilidades para enfrentarse a sistemas realizados por otros equipos, a leer y entender miles de lineas de código fuente, a entender los modelos y las decisiones de arquitectura que otros tomaron [123].

Fuerzas:

- El docente quiere que los estudiantes desarrollen habilidades para contribuir activamente a proyectos de código abierto.
- El docente quiere formar en arquitectura de software a través de un proyecto software preexistente para desarrollar habilidades para enfrentarse a sistemas desarrollados por otros equipos.
- La industria espera que los estudiantes sepan modificar una arquitectura de software para un sistema real preexistente.
- El docente quiere desarrollar un curso práctico para que los estudiantes aprendan a modificar la arquitectura de un sistema existente.
- Los proyectos con desarrollos previos no son de las preferencias de los estudiantes
- Los estudiantes tienen cierta experiencia trabajando proyectos de desarrollo complejos.
- Las comunidades de desarrollo esperan contribuciones para mejorar y mantener sus proyectos.

Solución: Trabajar con un proyecto open-source para que los estudiantes tengan la oportunidad única de aprender actitudes sólo presentes en escenarios del mundo real, lo que puede aumentar no solo sus habilidades sino también la confianza en sí mismos. Con los proyectos open-source se puede hacer componentes, extensiones, corregir bugs o analizar la arquitectura [97]. Asimismo, los estudiantes tendrán la oportunidad de interactuar con otros

arquitectos y desarrolladores, hacer extracciones de asuntos (issues) en GitHub y recibir realimentación de una comunidad, estudiar documentos de diseño y arquitectura [118]. La Figura **4-4** muestra los principales elementos de la solución del patrón.

Antes de iniciar el proceso, el profesor puede cuidadosamente seleccionar proyectos open-source, considerando tanto la complejidad arquitectural como las habilidades de los estudiantes. Asimismo, el docente tiene la posibilidad de elegir los requisitos y el estilo de arquitectura deseados. Por ejemplo, podría optar por transformar una arquitectura de capas de un módulo a una hexagonal, o realizar una refactorización guiada por un patrón específico, motivada por un atributo particular de calidad. En paralelo, se brinda a los estudiantes la oportunidad de seleccionar el proyecto que más les atraiga, fomentando así un mayor nivel de motivación durante todo el proceso de aprendizaje.

Además, en la elección del proyecto el docente debe tener en cuenta criterios como [112]:

- *El tamaño del proyecto*: Se recomienda tamaños aproximados entre 5.000 y 10.000 lineas de código. Proyectos demasiado grandes pueden ser complicados de entender para los estudiantes.
- *El lenguaje de programación*: A los estudiantes se les facilita trabajar lenguajes como Java, C#, python ya que están familiarizados con su uso.
- *El dominio de aplicación*: Preferiblemente elegir dominios que no requieran demasiado tiempo de aprendizaje para los estudiantes.
- *Diseño modular*: Los proyectos deben ser modulares y estar poco acoplados para demostrar eficazmente las mejores prácticas que ayudan al mantenimiento.
- *Actividad reciente*: son preferibles los proyectos con actualizaciones (commits) recientes.
- *Calidad de la documentación*: Los proyectos deben estar documentados para facilitar su comprensión.
- Facilidad de compilación: Los proyectos se deben poder compilar con poco esfuerzo. En ocasiones la falta de bibliotecas no permiten la compilación.

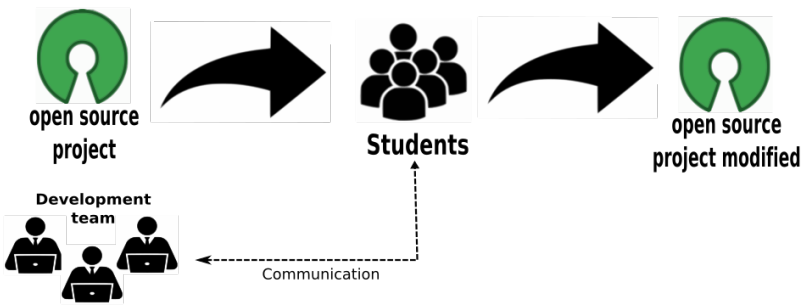

Figura 4-4.: Open-source projects-based training

El docente prepara una lista de proyectos open-source candidatos a ser trabajados por los estudiantes durante el curso tal como lo muestra la Tabla **4-5**.

Luego, los estudiantes de manera individual o en equipos eligen el proyecto donde harán la contribución guiados por el instructor y por sus intereses y preferencias [97].

Después de elegido el proyecto el próximo paso es seleccionar las tareas que el estudiante trabajará en el curso [97]. Esta labor puede ser ágil y democrática, por ejemplo, los estudiantes y el instructor se pueden reunir un día de la semana, abrir el listado de problemas del proyecto y discutir lo que se puede trabajar. Las contribuciones pueden ser de 4 tipos según Hattori and Lanza [48]:

1. *Ingeniería directa*: Agregando nuevas características. Ejemplo, una contribución al repositorio de LibreOffice Impress es permitir al usuario cambiar diapositivas usando un teléfono inteligente. Al agregar una nueva característica los estudiantes están obligados a entender y respectar las restricciones previamente establecidas de la arquitectura del sistema.

2. *Reingeniería*: En el marco de las actividades de refactorización, se pueden realizar la optimización de capas que no cumplen plenamente con las mejores prácticas de Android. Además, se contempla la posibilidad de modernizar o mejorar propiedades del sistema, como la facilidad de modificación o su capacidad de escalabilidad.

3. *Correctivo*: Por ejemplo, corregir errores. Por ejemplo, un proceso de importación a una base de datos, funciona solo con MySQL, pero no funciona cuando se usa Postgres. Las correcciones requieren conocer las restricciones de la arquitectura previamente establecidas.

4. *Gestión*: Por ejemplo, la actualización de la documentación de la arquitectura, reportar bugs, agregar etiquetas en problemas (issues), seguir los procesos de scrum (sprints, historias de usuario, planificación poker, etc.).

No	Proyecto	Lenguaje	Dominio
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

Tabla 4-5.: Ejemplo de listado de proyectos open-source [97]

Finalmente, el docente hace seguimiento de los proyectos [97]. Aunque los instructores no sean expertos en los proyectos open-source, es importante su participación activa, por ejemplo, investigando o haciendo contribuciones al proyecto de open-source.

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos
- Desarrollo web a pequeña escala
- Desarrollo movil a pequeña escala

Deseables:

- Sistemas operativos
- Sistemas Distribuidos
- Ingeniería de Software

Ejemplo de uso del patrón:

En un curso de Arquitecturas de Software, el docente busca que sus estudiantes se enfrenten a la evolución de un proyecto real y descubran su capacidad para superar desafíos reales de la industria. Con este propósito, ha decidido fomentar la participación en proyectos open-source. Tras proporcionar una lista de posibles proyectos, se han seleccionado, de común acuerdo entre el docente y las preferencias de los estudiantes, los siguientes proyectos para la clase::

1. Mejorar la extensibilidad de Jabref aplicando el estilo de micro-kernel (reingenieria).
2. Agregar a Teammates la posibilidad de notificar a los interesados eventos a través de sistemas de manesajería SMS (ingeniería directa).

Estos dos proyectos se abordarán mediante equipos de seis estudiantes. Con el objetivo de evitar conflictos al interactuar con las comunidades de desarrollo, estas iniciativas no se reportarán directamente a dichas comunidades; en cambio, se gestionarán como ramas independientes dentro de los proyectos.

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Apéndice C: Tabla de Competencias):

Competencias obligatorias: C11, C22, C23.

Competencias opcionales: C24, C14.

Variantes para llevarlo a la práctica: Tal como se describió en la solución, hay cuatro posibles variantes de este patron de formación:

1. *Ingeniería directa*: Agregando nuevas características. Al agregar una nueva característica los estudiantes están obligados a entender y respectar las restricciones previamente establecidas de la arquitectura del sistema.
2. *Reingeniería*: Se pueden hacer mejoras arquitectónicas mejorando las propiedades del sistema, por ejemplo, la facilidad de modificación o su capacidad de escalabilidad.
3. *Correctivo*: Las correcciones requieren conocer las restricciones de la arquitectura previamente establecidas.
4. *Gestión*: Por ejemplo, la actualización de la documentación de la arquitectura, reportar bugs, etc.

.

Ventajas:

- Los estudiantes trabajan con proyectos reales [97].
- Se generan habilidades interactuando con sistemas de control de versiones.
- Los estudiantes se vuelven miembros de una comunidad de desarrollo activa [97].
- En sistemas grandes de miles de lineas de código es evidente la necesidad de la arquitectura y modelos simples que ayuden a comprender la complejidad del sistema [22].

Desventajas:

- Los estudiantes e instructores tienen que lidiar con la la complejidad de la arquitectura y de la organización del código fuente de un proyecto grande [97].
- Interactuar con la comunidad puede ser difícil, por ejemplo, la interacción a través de una lista de correo es compleja ya que no se sabe quién es quién y quién responderá [97].
- Al principio los estudiantes tienen que lidiar con las complejidades para comprender y configurar el entorno de desarrollo de software, el sistema de control de cambios, habilidades en el uso de la línea de comando, etc.
- Esta estrategia resulta más compleja para los instructores que requieren orientar adecuadamente a los estudiantes [97].

Patrones relacionados: El patrón Open-source projects-based training se puede combinar en un curso de AS con otros patrones de formación que son livianos de aplicar para docentes y estudiantes y permiten desarrollar varias competencias en toma de decisiones. Por ejemplo, Case and Flipped Classroom Training, Problem-solving-based training, y Games-Based Training.

Experiencias reportadas del uso del patrón:

- Training software engineers using open-source software: the students' perspective [97] presenta las percepciones de los estudiantes sobre la necesidad de contribuir a un proyecto de código abierto como parte de un curso de Ingeniería de Software.
- A Collaborative approach to teaching software architecture [118] muestra cómo los participantes trabajan juntos para estudiar y documentar un gran sistema de software de código abierto de su propia elección.
- Promoting creativity, innovation and engineering excellence [123] muestra un experimento donde los estudiantes desarrollan un componente, extensión o aplicación en o sobre un proyecto de software de código abierto existente.
- Teaching software architectures and aspect-oriented software development using open-source projects [28] presenta un estudio donde los estudiantes deben desarrollar un gran sistema software a partir de proyectos open-source.
- Open-source software in class: students' common mistakes [51] muestra un caso donde se introduce proyectos open-source en un curso de ingeniería de software.
- Leveraging Final Degree Projects for Open Source Software Contributions [93] presenta una experiencia práctica donde los estudiantes contribuyen a proyectos maduros open-source.

4.3.4. Patrón 4: In-house project-based training

Nombre: In-house project-based training.

Contexto problemático:

Trabajar con proyectos de código abierto en el mundo real implica retos para estudiantes e instructores, los cuales son (i) la complejidad del código fuente ya que requiere entender la estructura de todo el proyecto, (ii) la interacción con la comunidad de proyectos de código abierto se realiza a través de listas de correo sin conocer a las personas (iii) entender y configurar el entorno de desarrollo de software también es complejo, implica conocer sistemas

operativos como Linux, sistema de control de versiones, trabajar con línea de comandos, y (iv) la falta de tiempo para contribuir, en ocasiones la duración del curso no es suficiente para conocer el proyecto y luego realizar contribuciones [97].

Trabajar con proyectos open-source, a pesar de las ventajas de enfrentar a los estudiantes a modificar sistemas reales, puede llegar a ser demasiado complejo para los estudiantes y docentes. Por lo tanto, una alternativa es que los docentes dispongan de su propia aplicación open-source para enseñar arquitectura de software con la suficiente complejidad de un sistema industrial [124].

Fuerzas:

- El docente quiere formar en arquitectura de software a través de un proyecto open-source creado por él para resolver algunos de los problemas arquitectónicos comunes de la actualidad.
- El docente quiere desarrollar un curso práctico para que los estudiantes aprendan conceptos de arquitectura a partir de un sistema existente.
- La industria espera que los estudiantes sepan modificar una arquitectura de software para un sistema real preexistente.
- Los estudiantes no tienen experiencia desarrollando proyectos de desarrollo complejos.
- Los estudiantes necesitan una aplicación de código abierto a la medida, diseñada para el aula con el fin de crear una experiencia de aprendizaje más atractiva y personalizada, que puede no lograrse plenamente a través de proyectos externos de código abierto.

Solución: Los docentes pueden disponer de su propia aplicación open-source para enseñar arquitectura de software con la suficiente complejidad de un sistema industrial. En este sentido, los docentes podrían contar con su propio sistema, como por ejemplo, un sistema E-commerce B2C, utilizado como una herramienta pedagógica para abordar aspectos relacionados con la disponibilidad, seguridad y escalabilidad. [124]. Estos proyectos pueden provenir de la industria o pueden haber sido construidos por los docentes. Periódicamente este tipo de proyectos pueden ser revisados por la industria para ver si cumplen con las características de un sistema industrial (ver Figura **4-5**).

De esta forma, a los estudiantes se les presentan las mejores prácticas que se utilizan ampliamente en la industria para resolver algunos de los problemas arquitectónicos comunes de la actualidad [124]. Mediante el uso de un estudio de caso concreto y realista de un área familiar, los estudiantes obtienen un mejor contexto para aplicar los principios arquitectónicos aprendidos en clase. Cada concepto y principio teórico que los estudiantes aprenden en la teoría, está respaldado por un escenario de aplicación concreto en este proyecto open-source. Por ejemplo, el profesor puede enseñar las ventajas de una solución monolítica y luego mostrar una versión distribuida basada en microservicios. Los estudiantes pueden descargar los

proyectos en sus máquinas, estudiar el código fuente, los manuales, ejecutarlos, probarlos y comprobar las ventajas y desventajas estudiados desde lo teórico.

Si se unen esfuerzos de varios docentes de distintas Universidades, se puede llegar a tener un repositorio de proyectos open-source realistas de varios dominios [124], de modo que los instructores puedan hacer referencia a ellos para mejorar la experiencia de aprendizaje de arquitectura de software, así los docentes no sean ingenieros de software en la práctica.

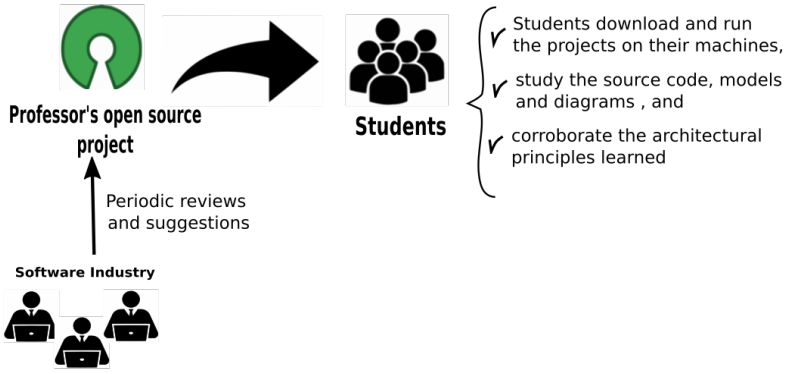

Figura 4-5.: In-house project-based training

Para mantener este tipo de aplicaciones, el profesor puede alojarlas en un repositorio de código en GitHub y buscar ayuda de colaboradores(instructores, amigos del sector, estudiantes de máster) para realizar actualizaciones y mejoras. Sería ideal contar con un repositorio extenso de varios proyectos de código abierto dedicados a la formación. La industria puede desempeñar un papel crucial como aliada estratégica para que las universidades evalúen la vigencia continua de una aplicación como solución empresarial a lo largo del tiempo.

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos
- Desarrollo web a pequeña escala
- Desarrollo móvil a pequeña escala

Deseables:

- Sistemas operativos
- Sistemas distribuidos

- Ingeniería de software

Ejemplo de uso del patrón:

Al comienzo del curso el docente da a conocer a sus estudiantes la aplicación de comercio electrónico open-source (desarrollada por el docente) que servirá para reforzar los conceptos teóricos del curso [124].

El curso comienza con la introducción de la arquitectura tradicional monolítica en capas para construir una aplicación web y luego analiza los inconvenientes y las posibles mejoras. En un sistema monolítico las aplicaciones son desplegadas como una unidad de un servidor web. Para tráfico pequeño, un servidor puede ser suficiente. Pero cuando el servidor de aplicaciones recibe mucho tráfico durante la temporada de alto tráfico de solicitudes, será necesario duplicar los proyectos en más servidores. Estos sistemas son fáciles de desarrollar por los estudiantes. El proyecto de ejemplo del sistema de comercio eletrónico permite a los estudiantes entender los conceptos anteriores.

A continuación, se explica a los estudiantes los inconvenientes de la arquitectura tradicional monolítica. En primer lugar, no todos los módulos del "Sistema de compras" en línea reciben la misma cantidad de concurrencia o presión [124]. Por ejemplo, siempre habrá más personas navegando y buscando que realmente comprando un producto. La presión del módulo llamado Sistema de Administración es generalmente mucho más pequeña que la del "Sistema de Compras en Línea", ya que sus usuarios son solo una docena de administradores y personal de operaciones de TI. Por lo tanto, es un desperdicio implementar todo en varios servidores de aplicaciones al mismo tiempo. Esto conduce a una escalabilidad deficiente, y lo que realmente deberíamos hacer es agregar más servidores para extender los servicios que están bajo mayor presión. El segundo inconveniente es que dicha arquitectura dificulta el desarrollo de un equipo. El trabajo de diferentes equipos debe integrarse en un proyecto para construirlo y ejecutarlo. Por ejemplo, si el equipo de la interfaz de usuario solo necesita modificar la página de un producto (tal vez un error tipográfico), debe volver a empaquetar y volver a implementar toda la aplicación (período durante el cual, todo el sitio web está inactivo).

Una vez explicados las ventajas y desventajas de la arquitectura tradicional, el docente puede pasar a la arquitectura distribuida [124]. Cada módulo de la arquitectura tradicional se debe sacar del sistema monolítico y se desarrolla un sistema independiente para él. Cada módulo se ejecutará en un contenedor Docker separado y se comunicará con otros módulos a través de servicios web RESTful. La separación de estos módulos también puede facilitar una mejor colaboración en equipo y gestión de proyectos.

La implementación y las operaciones también pueden ser parte de este curso. Durante la primera mitad del curso, se utiliza el entorno virtual VMWare para simular el despliegue. Luego se puede cambiar a la implementación en la nube. También se puede presentar el

enfoque DevOps.

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Apéndice C: Tabla de Competencias):

Competencias obligatorias: C01, C05, C08.

Competencias opcionales: C11, C12.

Variantes para llevarlo a la práctica: Ninguna.

Ventajas:

- El docente dispone de una aplicación open-source a las necesidades de su curso.
- Los estudiantes pueden descargar y ejecutar los proyectos en sus máquinas, estudiar el código fuente, modelos, diagramas y manuales.
- Cada concepto aprendido de arquitectura de software los estudiantes lo evidencian en la aplicación.

Desventajas:

- Para el docente puede llegar a ser complejo desarrollar o disponer de ejemplos de aplicaciones open-source a la medida del curso.
- Las aplicaciones de ejemplo se deben estar actualizando en el tiempo según los avances tecnológicos.
- Puede ser que el proyecto no sea motivante para el estudiante.

Patrones relacionados:

El patrón In-house project-based training se puede combinar en un curso de AS con otros patrones de formación que son mas livianos de aplicar para docentes y estudiantes y permita desarrollar varias competencias en toma de decisiones. Por ejemplo, Case and Flipped Classroom Training, Problem-solving-based training, y Games-Based Training.

El patrón Open-source projects-based training es una alternativa a este patrón en caso que el docente no cuente con su propio sistema open-source.

Experiencias reportadas del uso del patrón:

- Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application [124] propone una experiencia de Aprendizaje Basado en Proyectos, que

lleva a las aulas un sistema completo de código abierto para enseñar eficazmente arquitectura de software distribuido.

- A Collaborative Approach to Teaching Software Architecture [118] propone un curso colaborativo de AS donde los estudiantes trabajan juntos para estudiar y documentar un gran sistema open-source.

4.3.5. Patrón 5: Cases-based training

Nombre: Cases-based training.

Contexto problemático:

Tomar decisiones de arquitectura acertadas durante la construcción de un sistema de software, son una de las mayores habilidades que debe desarrollar un futuro arquitecto de software. Del mismo modo, las decisiones proporcionan una elección realizada por el arquitecto de software en un contexto específico junto con su justificación o razón de ser [100].

Las decisiones pueden referirse a elegir la estructura de la aplicación o el sistema, la selección de una tecnología para implementar el diseño o un compromiso entre atributos de calidad. Sea cual sea el contexto, una decisión de arquitectura ejemplar ayuda a los equipos de desarrollo a tomar las decisiones técnicas correctas. Por lo tanto, una decisión de arquitectura debe explicarse en términos de las siguientes características:

- Alternativas de diseño disponibles.
- Justificación de la decisión.
- Documentar la decisión.
- Comunicar eficazmente la decisión a las partes interesadas.

El docente debe proporcionar durante el desarrollo del curso, las condiciones necesarias para que sus estudiantes aprendan a tomar decisiones.

A continuación se presenta un escenario concreto en el que el alumno debe tomar decisiones de arquitectura de software. Al diseñar un nuevo sistema de software, basándose en los atributos de calidad del nuevo sistema (escalabilidad, disponibilidad, seguridad, rendimiento, tolerancia a fallos, elasticidad, entre otros), el arquitecto debe seleccionar primero un pequeño conjunto de atributos que serán los más relevantes a satisfacer por el sistema, y que se convertirán en los drivers de la arquitectura (no es posible diseñar un sistema que satisfaga a todos los atributos). A continuación, hay que decidir qué estilo o estilos arquitectónicos, así como qué tácticas de arquitectura, favorecen estos atributos de calidad. Por ejemplo, un estilo de microservicios favorece a la escalabilidad, la elasticidad y la evolución, pero al

mismo tiempo, la tolerancia a fallos y la fiabilidad sufren cuando se utiliza demasiada comunicación entre servicios; en un estilo de arquitectura pipeline, el coste global, la simplicidad y la modularidad son sus principales puntos fuertes, ya que es monolítica, pero la elasticidad y la escalabilidad son deficientes; en un estilo de arquitectura dirigida por eventos, el rendimiento, la escalabilidad y la tolerancia a fallos son sus principales puntos fuertes, pero la simplicidad y la comprobabilidad son relativamente bajas [100]. En resumen, los estilos arquitectónicos elegidos deben justificarse en función de los requisitos de la aplicación. Además, el arquitecto debe decidir el tipo de aplicación que va a desarrollar: web, web de una sola página, de escritorio, móvil, híbrida (web y móvil). Finalmente, el arquitecto debe elegir las tecnologías adecuadas para el sistema respondiendo a las siguientes preguntas ¿Qué tecnologías ayudan a implementar los estilos arquitectónicos seleccionados? ¿Qué tecnologías permiten implantar el tipo de aplicación seleccionado? ¿Qué tecnologías ayudan a cumplir los requisitos no funcionales especificados?

Fuerzas:

- El docente quiere desarrollar habilidades en sus estudiantes relacionados con la toma de decisiones de arquitectura de un sistema software.
- La industria espera que los estudiantes sepan tomar decisiones de arquitectura de software.
- El docente no quiere acudir al desarrollo de proyectos de software porque implican demasiado tiempo.
- Los estudiantes no tienen habilidades suficientes para lleva a cabo la implementación de un proyecto de software.

Solución:

Los estudios de caso son escenarios de empresas de la vida real, que permiten a los estudiantes analizar, aplicar los conceptos enseñados y compensaciones (trade-off) dentro de un contexto realista [69].

La enseñanza basada en casos se centra el diseño y análisis de proyectos reales de software de las empresas. Los estudiantes experimentan y utilizan la teoría y la tecnología del diseño de arquitectura de software aplicado a proyectos específicos, con el fin de mejorar el efecto de enseñanza [69]. Los principales pasos de este patrón de formación se pueden apreciar en la Figura **4-6**.

Este patrón de formación propone realizar talleres relacionados con la industria en forma de estudios de caso, que puedan guiar y hacer que los participantes pasen por los procesos de pensamiento de un arquitecto junior [69]. Estos talleres podrían incluir ejemplos de proyectos

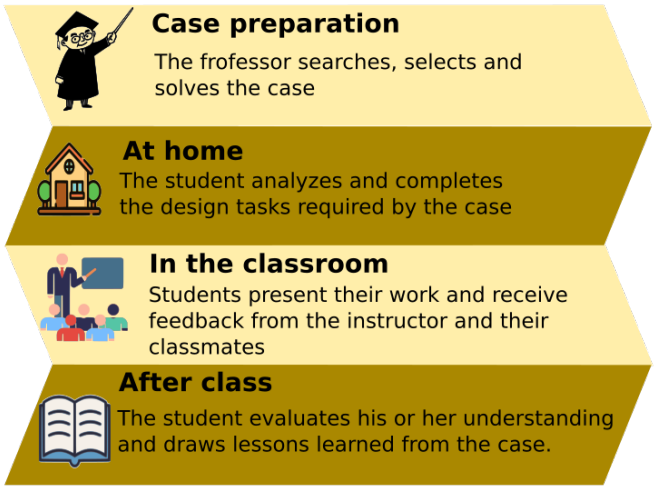

Figura 4-6.: Cases-based training

reales que muestren situaciones por las que pasa un arquitecto de soluciones en un entorno de trabajo real. El docente podría extraer estos talleres de las experiencias de arquitectos del sector, blogs de arquitectura y otros.

Los pasos generales a seguir para conducir un caso son [82]:

1. Primero el docente enseña los fundamentos de arquitecturas.
2. El docente define el tema y objetivos de aprendizaje del caso.
3. El docente busca y selecciona el caso. Lo puede hacer en diversas fuentes, como libros de texto, revistas académicas, sitios web especializados, bases de datos de casos educativos y otros recursos en línea. También pueden plantearse crear sus casos basándose en situaciones del mundo real.
4. Adaptación y personalización. En algunos casos, el profesor puede tener que adaptar el caso a las necesidades de sus estudiantes. Esta adaptación puede consistir en simplificar o ampliar partes del caso o ajustar la información para que sea más pertinente en el contexto educativo.
5. Preparación del material didáctico. Una vez seleccionado y adaptado el caso, el profesor debe elaborar el material que presentará a los estudiantes. Podría incluir descripciones detalladas del caso, datos relevantes, reflexiones, preguntas para el debate y recursos adicionales para ayudar a los estudiantes a comprender el contexto.
6. Trabajo individual o en grupo. Los estudiantes trabajan individualmente o en grupo para analizar el caso, identificar los problemas, proponer soluciones y debatir sus

conclusiones. Esta fase fomenta la participación activa y el pensamiento crítico.

7. Debate y análisis. El docente facilita los debates en clase en los que los estudiantes comparten sus análisis, las soluciones propuestas y sus razonamientos. Esta situación puede dar lugar a debates enriquecedores y a una comprensión más profunda de los conceptos implicados.
8. Síntesis y conclusión. Al final del proceso, el profesor resume las lecciones clave que pueden extraerse del caso y su relevancia en el contexto de aprendizaje más amplio.

El aprendizaje basado en casos es una estrategia educativa eficaz para desarrollar la capacidad de toma de decisiones y fomentar el pensamiento crítico de los estudiantes. Algunas formas en que los instructores pueden garantizar que el estudio de casos aborde eficazmente estas habilidades son:

- Selección de casos pertinentes y que supongan un reto. Los instructores deben elegir casos pertinentes para los objetivos del curso y que supongan un reto para los estudiantes. Los casos deben reflejar situaciones del mundo real en las que los estudiantes se enfrentarán a decisiones complejas.
- Definición clara de los objetivos. Antes de presentar el caso, los instructores deben establecer objetivos claros sobre lo que los estudiantes deben aprender y lograr. Esto proporciona una dirección clara y garantiza que el caso esté alineado con los resultados de aprendizaje deseados.
- Estimular el debate. Los casos deben diseñarse de forma que no haya una única respuesta correcta. Esta discusión fomentará el debate y la discusión de los estudiantes, fomentando el pensamiento crítico al considerar diferentes perspectivas y soluciones.
- Proporcionar información limitada. Los casos deben presentar información limitada, simulando así situaciones reales en las que los responsables de la toma de decisiones a menudo deben trabajar con información incompleta o ambigua. Esta situación ayudará a los estudiantes a desarrollar habilidades para identificar y reunir la información necesaria para tomar decisiones con conocimiento de causa.
- Fomentar la reflexión. Tras analizar el caso, hay que animar a los estudiantes a que reflexionen sobre sus decisiones y sobre cómo han llegado a esas conclusiones. Preguntas como "¿Por qué eligieron esa opción?", "¿Qué otras alternativas consideraron?" y "¿Qué aprendieron de este proceso?" pueden fomentar la autorreflexión.
- Proporcionar comentarios constructivos. Los instructores deben hacer comentarios constructivos sobre las decisiones y los análisis de los estudiantes. Esta retroalimentación no sólo valida su esfuerzo, sino que también les proporciona ideas para mejorar sus habilidades de toma de decisiones en el futuro.

- Vinculación con la teoría y los conceptos. Una vez analizado el caso, es esencial relacionar las conclusiones y decisiones de los estudiantes con las teorías y conceptos pertinentes de la AS. Esta vinculación ayuda a los estudiantes a comprender cómo se aplica la teoría a situaciones prácticas.

Los profesores pueden realizar principalmente la enseñanza basada en casos en forma de **aula invertida**. Antes del encuentro presencial, cada estudiante tiene un tiempo para analizar el caso y completar las tareas de diseño requeridas por el caso. Durante el encuentro presencial, los estudiantes exponen sus trabajos y reciben la realimentación del instructor y de sus compañeros. Después del encuentro presencial el estudiante evalúa su entendimiento y saca las lecciones aprendidas del caso.

Algunas actividades que el estudiantes puede realizar antes del encuentro presencial son:

- Los estudiantes pueden recibir del docente vídeos pregrabados, lecturas o recursos multimedia. Este material puede explicar conceptos clave, demostrar procesos, presentar ejemplos y contextualizar el caso que se abordará en una reunión presencial.
- Leer capítulos de libros de texto, artículos académicos o documentos relacionados con el caso que se tratará en clase.
- Realizar ejercicios, pruebas o tareas relacionadas con el caso. Estas actividades pueden ayudarles a evaluar su comprensión y a prepararse para la reunión presencial.
- Se puede animar a los estudiantes a investigar en Internet o en otras fuentes para profundizar en el tema y descubrir información adicional.
- Se puede invitar a los estudiantes a generar preguntas o inquietudes basadas en los contenidos anteriores. Estas preguntas pueden servir como punto de partida para el debate en la reunión presencial.
- Participar en foros o plataformas en línea donde discuten contenidos previos con sus compañeros, responden a preguntas planteadas por el profesor o generan debates.
- Escribir reflexiones, resúmenes o esquemas sobre contenidos anteriores para organizar sus pensamientos y prepararse para la interacción en clase.

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos

Deseables:

- Sistemas operativos

- Sistemas Distribuidos
- Ingeniería de Software

Ejemplo de uso del patrón:

El docente selecciona el siguiente caso para ser trabajado con sus estudiantes utilizando aula invertida. Un sistema de salud a nivel nacional ha sido diseñado con el objetivo de monitorear la salud de los estudiantes en instituciones de educación primaria, secundaria y terciaria. Se requiere que el alumno diseñe un sistema distribuido que aborde las cualidades de seguridad, rendimiento, mantenibilidad y escalabilidad de este sistema de salud. Las decisiones que se toman sobre el diseño de la arquitectura pueden favorecer una de las cualidades, pero probablemente compensen otra. Por ejemplo, el alumno deberá decidir si desea conservar primero los datos de atención médica en el almacenamiento disponible en cada institución durante el proceso de selección o acceder directamente a un sistema remoto centralizado para conservar los datos. La adopción de la primera puede permitir un mejor desacoplamiento del sistema, pero arriesga la inconsistencia de los datos en todas las instituciones. Por otro lado, la adopción de este último puede lograr una mejor capacidad de mantenimiento de la arquitectura y la consistencia de los datos, pero corre el riesgo de un punto único de falla en cada institución. Para cada compensación, los estudiantes deben poder recomendar acciones de mitigación [84].

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Apéndice C: Tabla de Competencias):

Competencias obligatorias: C01, C02, C05, C08, C12, C18, C22, C23.

Competencias opcionales: C03.

Variantes para llevarlo a la práctica: Una forma de trabajar con casos es a través de conferencias con invitados de al industria de software [26] [128]. A lo largo del curso el docente puede organizar de una a cuatro conferencias o charlas. En cada conferencia el invitado cuenta los detalles de un caso real de arquitectura: el contexto, el problema, las decisiones tomadas y los resultados. De esta forma los estudiantes aprenden de la experiencia vivida por los arquitectos.

Ventajas:

- Los casos se centran en temas de arquitectura y se deja a un lado la implementación en código.

- Los casos permiten en corto tiempo desarrollar habilidades en al toma de decisiones de arquitectura de software.

Desventajas:

- La solución a los casos no es única. La forma en que se define la arquitectura de software depende en última instancia del contexto, las partes interesadas, las preocupaciones y, finalmente, el propósito de la arquitectura [67]. Por lo tanto, el docente tiene un gran reto al momento de resolver el caso.

Patrones relacionados:

El patrón Cases-based training puede integrarse en un curso de Arquitecturas de Software junto con otros patrones de formación que aborden el desarrollo de proyectos, permitiendo a los estudiantes adquirir experiencia tanto a través de casos prácticos como mediante proyectos de desarrollo. Por ejemplo, Mini-Projects-based training, Large Project-Based Training, Open-source projects-based training, y In-house project-based training.

Experiencias reportadas del uso del patrón:

- Applying case-based learning for a postgraduate software architecture course [84] explica cómo aplicar el aprendizaje basado en casos para abordar el reto de impartir un curso de postgrado sobre AS.
- Did our Course Design on Software Architecture meet our Student's Learning Expectations? [69] presenta la utilización de casos en cursos de AS dirigido a estudiantes adultos (de postgrado y trabajadores de la industria) con experiencias laborales que difieren en sus necesidades de aprendizaje y características de los estudiantes universitarios.
- Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course [67] propone un modelo que adapta los conceptos de aprendizaje experimental y gestión de riesgos para diseñar el curso sobre arquitectura de software.
- Improved Teaching Model for Software Architecture Course [55] propone un enfoque de enseñanza basado en casos.
- Flipped Classroom Applied to Software Architecture Teaching [45] propone el aula invertida invertida como un enfoque con potencial para reforzar el aprendizaje de la AS.

4.3.6. Patrón 6: Problem-solving-based training

Nombre: Problem-solving-based training.

Contexto problemático:

Tomar decisiones de arquitectura en equipo durante la construcción de un nuevo sistema de software, son una de las mayores habilidades que debe desarrollar un futuro arquitecto de software [100]. Al inicio de la creación de una aplicación es donde se toman las decisiones más importantes: tecnologías a usar, patrones de arquitectura, atributos de calidad, entre otros. Después de esto, viene el desarrollo y mantenimiento de la aplicación.

El docente debe proporcionar durante el desarrollo del curso, las condiciones necesarias para que los estudiantes aprendan a tomar en equipo este tipo de decisiones.

Cuando los estudiantes trabajan en equipo y se enfrentan a la toma de decisiones complejas, se presentan muchos retos:

- Diferencias en la dinámica del equipo. Diferencias de personalidad, estilo de trabajo y comunicación entre los miembros del equipo. La dinámica de grupo puede afectar a la eficiencia y eficacia del proceso de toma de decisiones.
- Dificultades de comunicación. Una comunicación ineficaz puede dar lugar a malentendidos, falta de claridad y problemas de coordinación. La comunicación eficaz es esencial para compartir ideas, debatir soluciones y llegar a un consenso.
- Gestión del tiempo. Trabajar en equipo puede exigir una coordinación eficaz del tiempo, especialmente cuando se abordan problemas complejos. La planificación y la gestión del tiempo pueden suponer un reto, ya que los estudiantes deben equilibrar múltiples responsabilidades y tareas.
- Conflictos y desacuerdos. Cuando los estudiantes trabajan en equipo y se enfrentan a decisiones complejas, es probable que surjan desacuerdos y conflictos sobre las mejores soluciones. Gestionar estos retos puede ser complicado.
- Tomar decisiones complejas. Los problemas no suelen tener respuestas claras y únicas. Tomar decisiones en un entorno de incertidumbre puede suponer un reto para los estudiantes, ya que deben evaluar distintas opciones y considerar múltiples perspectivas.
- Equidad en la contribución. Garantizar que todos los miembros del equipo participen en con igualdad y contribuyan de forma significativa puede ser todo un reto. Algunos estudiantes pueden ser menos proclives a expresar sus ideas o pueden ser dominantes en el proceso.

- Presión para llegar a un consenso. Llegar a un consenso puede ser difícil cuando hay diferencias de opinión en el equipo. Algunos estudiantes pueden sentirse presionados para ceder en sus puntos de vista, lo que puede afectar a la calidad de la toma de decisiones.

Fuerzas:

- El docente quiere desarrollar habilidades en sus estudiantes relacionados con la toma de decisiones de arquitectura en equipo de un sistema software nuevo.
- La industria espera que los estudiantes sepan tomar decisiones de arquitectura de software.
- El docente no quiere acudir al desarrollo de proyectos de software porque implican demasiado tiempo.
- Los estudiantes no tienen habilidades suficientes para lleva a cabo la implementación de un proyecto de software.
- Cuando los estudiantes trabajan en equipo y se enfrentan a la toma de decisiones complejas, surgen muchos retos, como diferencias de personalidad, dificultades de comunicación, conflictos y desacuerdos, gestión del tiempo y equidad en la contribución, entre otros. Para superar estos retos, los instructores deben exponer a sus estudiantes a ejercicios que les permitan desarrollar habilidades de trabajo en equipo y de toma de decisiones.

Solución:

El enfoque de aprendizaje basado en problemas permite trabajar por equipos de estudiantes, resolviendo problemas arquitectónicos reales o ficticios, y los instructores juegan un papel mínimo y no interfieren en la discusión [68]. A medida que los estudiantes comienzan a explorar las dificultades, los instructores pueden actuar como facilitadores y utilizar preguntas de orientación para devolverlos al objetivo principal de aprendizaje. Por ejemplo, uno de los subgrupos explica su solución de diseño.

Los pasos para aplicar el enfoque basado en problemas son:

1. Identificación y selección del problema. El instructor identifica un problema realista, pertinente, estimulante y estimulante para los estudiantes, relacionado con los objetivos del curso de AS.
2. Presentación del problema. El instructor presenta el problema a los estudiantes de forma clara y concisa. Proporciona la información necesaria para que los estudiantes comprendan el contexto del problema.
3. Creación de equipos. El instructor organiza a los estudiantes en equipos de colabora-

ción. Puede hacerlo al azar o teniendo en cuenta los puntos fuertes y las capacidades individuales. Es importante fomentar la diversidad en los equipos para promover diferentes perspectivas.

4. Análisis y comprensión del problema. El instructor invita a los equipos a analizar y comprender plenamente el problema. Además, el instructor anima a los estudiantes a plantear preguntas, identificar la información que falta y definir los objetivos del problema.
5. Generación de ideas y debate. Los equipos generan ideas y posibles soluciones para abordar el problema. Se fomenta el debate en los equipos para explorar diferentes enfoques.
6. Análisis y desarrollo de soluciones. Los equipos analizan diferentes soluciones propuestas y evalúan sus pros y sus contras. Esta actividad fomenta el pensamiento crítico al considerar los aspectos éticos, sociales y técnicos de las soluciones.
7. Presentación y retroalimentación. Cada equipo socializa las diferentes soluciones delante de los demás. El instructor da retroalimentación constructiva sobre las soluciones y el proceso de toma de decisiones.

Un enfoque particular del patrón de formación basado en problema son las Katas de Arquitectura. La palabra Kata se la toma del Karate y hace referencia a un ejercicio individual de entrenamiento. Una Kata de Arquitectura es una actividad definida por Ted Neward[1] donde se busca diseñar la arquitectura de un sistema cercano a la realidad. La actividad suele realizarse en equipos, donde a cada uno se le asigna una ejercicio que se debe resolver en un tiempo determinado. Existe una persona con el rol de moderador, y que hace las veces de cliente, gerente de proyectos, usuario final, entre otros. El moderador tiene la tarea de aclarar las inquietudes que surjan en la kata. Los principales elementos de este patrón de formación se pueden apreciar en la Figura **4-7**.

[1]Ted Neward es arquitecto de desarrollo de software independiente y mentor en la zona de Sacramento, California. Es autor de varios libros.

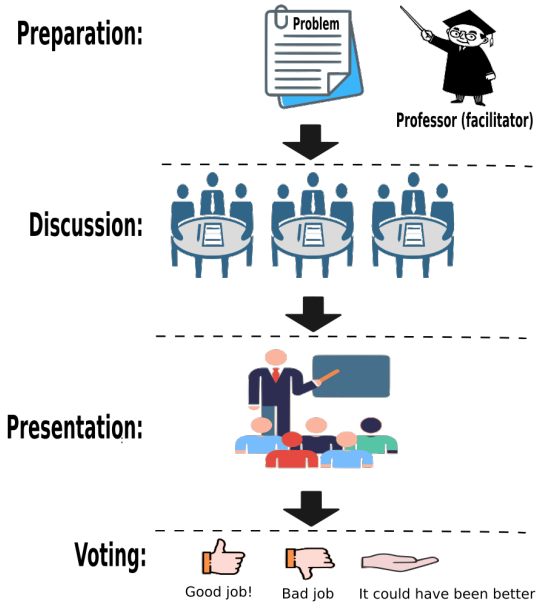

Figura 4-7.: Problem-solving-based training (Por ejemplo, Architecture katas)

Los pasos que se pueden seguir para aplicar las Katas de Arquitectura son:

1. El instructor *conforma los equipos* de trabajo de entre 3 a 5 estudiantes por grupo.
2. El ejercicio a resolver se asigna al azar. Ted Neward definió una lista inicial de ejercicios para las katas arquitectónicas en el sitio web `architecturalkatas.com`. El instructor puede pedir al sitio que seleccione una kata al azar (véase Figura **4-8**). Esta lista de ejercicios es extensa, y el instructor puede elegir cualquiera de ellos. Cada kata consta de requisitos funcionales, requisitos no funcionales y restricciones. Cuando haya dudas sobre los requisitos, se puede consultar al instructor. Los estudiantes pueden hacer suposiciones sobre los requisitos que faltan para tomar sus decisiones de diseño.
3. Viene la *discusión* para lo cual se les puede dar un tiempo de 45 minutos a los equipos para que propongan una solución arquitectónica al problema. Como los problemas son cortos en su redacción, se deben hacer supuestos sobre algunos requisitos faltantes o tecnologías a usar. Se recomienda que los estudiantes usen el modelo C4 para diseñar sus propuestas. Para la elección de los patrones de arquitectura acordes a los requisitos de calidad del problema, recomendamos los capítulos 10 al 18 del Libro *Fundamentals of software architecture* de Mark Richards [100].
4. Luego, viene la *presentación* de las propuestas. Para ello se elige una o dos personas por equipo para que expongan sus propuestas.

Figura 4-8.: Ejemplo de un ejercicio de kata generado al azar por el sitio architecturalkatas.com

5. Finalmente, viene la *votación*. El resto de equipos votan a partir de la presentación:
    - *¡Muy buen trabajo! (pulgar hacia arriba):* Esto indica que el ejercicio se resolvió a cabalidad, la solución es coherente y se seleccionaron tecnologías razonables.
    - *Mal trabajo (pulgar hacia abajo):* Se hicieron supuestos importantes sin ninguna validez.
    - *Pudo haber sido mejor (mano neutral u horizontal):* No se tiene una visión clara del proyecto y se olvidaron aspectos importantes.

Pantoja et al. muestra un ejemplo de la aplicación de un ejemplo de una kata en un curso de AS [91]. En este ejemplo, los estudiantes una vez comprendido los estilos de arquitectura, son sometidos a resolver una kata de arquitectura para desarrollar habilidades de diseño.

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos

Deseables:

- Sistemas operativos
- Sistemas Distribuidos
- Ingeniería de Software

Ejemplo de uso del patrón: Desarrollar en equipos de tres estudiantes la siguiente Kata de Arquitectura. La fase de discusión debe durar 45 minutos. Utilizar el modelo C4 para diseñar las soluciones.

El guerrero del camino: Una importante agencia de viajes quiere construir un panel de administración de viajes de próxima generación, que le permita a los viajeros ver todas sus reservas existentes, organizadas por viaje, ya sea en línea o a través de un dispositivo móvil. El sistema debe soportar más de 10 mil usuarios registrados a nivel mundial.

Los requerimientos de este sistema son:

- Debe conectarse al sistema existente de la agencia para aerolíneas, hoteles y alquiler de vehículos. La conexión debe permitir cargar automáticamente reservas a través de cuentas de viajero frecuente, cuentas de puntos de hoteles y cuentas de recompensa de alquiler de vehículos.
- Los clientes pueden añadir manualmente reservas existentes.
- Los ítems en el panel se pueden agrupar por viaje, y una vez un viaje esté completado, los ítems se remueven automáticamente del panel.
- Los usuarios también pueden compartir su información de viaje a través de redes sociales.
- Interfaz de usuario lo más rica posible a través de todas las plataformas.

Contexto adicional:

- Debe integrarse de manera transparente con sistemas de viaje existentes.
- Se están negociando alianzas para que existan proveedores "favorecidos".
- Debe funcionar a nivel internacional.

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Apéndice C: Tabla de Competencias):

Competencias obligatorias: C01, C02, C05, C08, C12, C18, C22, C23.

Competencias opcionales: C03.

Variantes: Ninguna

Ventajas:

- Los problemas se centran en temas de arquitectura y se deja a un lado la implementación en código.
- Los problemas permiten en corto tiempo desarrollar habilidades en el arquitecto de

software.

Desventajas:

- No hay soluciones únicas para los problemas, el docente requiere experiencia para orientar las propuestas de los estudiantes.

Patrones relacionados:

El patrón Problem-solving-based training se puede combinar en un curso de AS con otros patrones de formación que involucren el desarrollo de un proyecto de software. Por ejemplo, Mini-Projects-based training, Large Project-Based Training, Open-source projects-based training, and In-house project-based training.

Experiencias reportadas del uso del patrón:

- Using Architectural Kata in Software Architecture Course: An Experience Report [81] propone las katas de arquitectura como ejercicio de grupo para enseñar a diseñar, documentar y evaluar la arquitectura de software.
- Teaching adult learners on software architecture design skills [68] [45] propone enseñar AS a alumnos adultos utilizando casos y resolución de problemas.
- Applying case-based learning for a postgraduate software architecture course [84] aplica el aprendizaje basado en casos y problemas para abordar el reto de impartir un curso de postgrado sobre AS.
- Aligning Software Architecture Training with Software Industry [91] muestra la aplicación de un taller de katas de arquitectura en un curso de AS de pregrado.

4.3.7. Patrón 7: Games-based training

Nombre: Games-based training.

Contexto problemático:

Enseñar arquitectura de software es una disciplina difícil porque el rol de arquitecto es multifascético [67]. El arquitecto requiere desarrollar habilidades técnicas, analíticas, de comunicación, etc. La mayoría de los arquitectos talentosos en la industria han adquirido un amplio conocimiento a lo largo de muchos años de experiencia. En cierto modo, si deseamos que el diseño arquitectónico sea sistemático y reproducible, necesitamos mejorar los métodos para enseñar. No es aceptable esperar simplemente a que un aspirante a arquitecto acumule 10 o 20 años de experiencia si consideramos que la ingeniería de software es una disciplina

de ingeniería real [19].

Los docentes necesitan métodos de enseñanza que sean divertidos y permitan motivar y acotar el tiempo de formación relacionados con la toma de decisiones de arquitectura de software. La formación basada en juegos es una estrategia educativa que utiliza elementos de juegos para fomentar el compromiso, la participación y el aprendizaje de los estudiantes. Aunque es eficaz para muchos, también presenta retos para los educadores. He aquí algunos de ellos:

- Diseñar juegos educativos eficaces. Crear juegos que sean educativos y atractivos en temas de AS puede ser complicado. Los juegos deben equilibrar eficazmente la diversión con el contenido educativo, garantizando que se cumplan los objetivos de aprendizaje sin sacrificar el atractivo del juego.
- Alineación con los objetivos de aprendizaje. Garantizar que los juegos aborden objetivos de aprendizaje específicos puede ser un reto. Los instructores deben diseñar juegos que se relacionen directamente con los temas y habilidades que se enseñan.
- Evaluar el aprendizaje. Determinar cómo evaluar el aprendizaje de los estudiantes a través de los juegos puede resultar complejo. Los instructores deben desarrollar métodos de evaluación que sean apropiados y que reflejen los conocimientos y habilidades adquiridos a través de la experiencia de juego.
- Dificultad en el diseño de la progresión. Los juegos educativos deben tener una curva de dificultad adecuada para los estudiantes. Los estudiantes pueden perder interés o frustrarse si el juego es demasiado fácil o difícil.

Fuerzas:

- El docente busca cultivar habilidades en sus estudiantes vinculadas a la toma de decisiones en la arquitectura de sistemas de software de manera estimulante y cautivadora. El docente considera que la introducción de elementos lúdicos y actividades atractivas contribuirá significativamente al interés y compromiso de los estudiantes con la asignatura, generando un aprendizaje más efectivo y satisfactorio.
- El docente no quiere acudir al desarrollo de proyectos de software porque implican demasiado tiempo.
- La industria espera que los estudiantes sepan tomar decisiones de arquitectura de software.
- Los estudiantes no tienen habilidades suficientes para lleva a cabo la implementación de un proyecto de software.

Solución:

El docente puede acudir a los juegos para enseñar a tomar decisiones durante el diseño de la arquitectura de software de manera estimulante y cautivadora. Los juegos pueden proporcionar una ilustración útil del proceso de toma de decisiones de diseño y enseñar a los estudiantes el poder de la interacción en equipo para tomar decisiones acertadas [61]. Los juegos permiten desarrollar habilidades en diseño de software de manera divertida y atractiva para los estudiantes (ver Figura **4-9**).

El juego no es un sustituto de la instrucción "tradicional" sobre diseño, sino más bien como un complemento de dicha instrucción [61]. En consecuencia, no se espera que los jugadores aprendan en detalle cómo diseñar o cómo tomar decisiones de diseño óptimas simplemente jugando. En cambio, el juego puede usarse como punto de partida para discusiones más profundas sobre las complejidades del diseño arquitectónico o para practicar varios aspectos del proceso de diseño. Los participantes del juego pueden comprender, en poco tiempo y de una manera entretenida y convincente, cómo se realiza el diseño y los diferentes conceptos y actividades asociadas con esta actividad crucial.

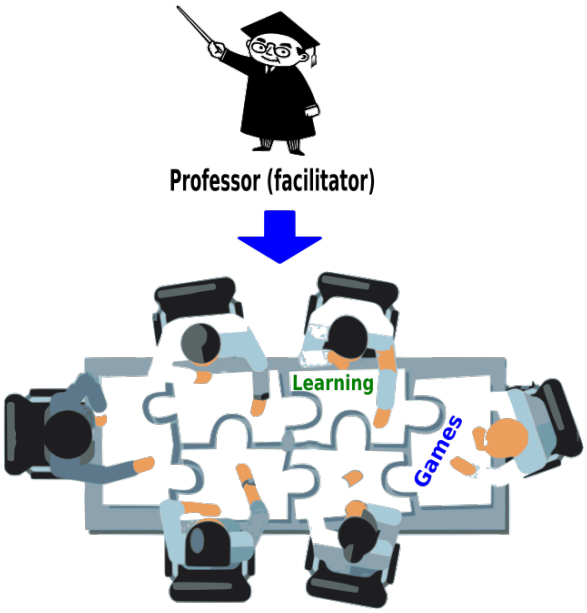

Figura 4-9.: Games-based training

El instructor puede aplicar juegos para enseñar la toma de decisiones de AS, en situaciones como:

1. Diseñar arquitecturas que cumplan los atributos de calidad necesarios para el éxito del sistema y guiar a los diseñadores para que tomen decisiones informadas y coherentes con los requisitos y expectativas del proyecto. Esta situación implica identificar los

atributos de calidad más críticos para el sistema. Estos atributos varían en función del contexto y los requisitos del proyecto. A continuación, se priorizan los atributos en función de su importancia para el sistema. Dependiendo del dominio del software y de las necesidades del usuario, algunos atributos pueden ser más críticos que otros.

2. Evaluar y analizar arquitecturas de software en términos de atributos de calidad como rendimiento, escalabilidad, disponibilidad, seguridad, usabilidad, mantenibilidad y otros factores relevantes para el sistema. Los arquitectos y los equipos de desarrollo deben comprender cómo afectan las decisiones arquitectónicas a los atributos de calidad y cómo las compensaciones (trade-off) pueden influir en la arquitectura final.

A continuación se describen tres juegos relacionados con la formación en arquitectura de software y toma de decisiones.

Smart Decision es un juego de diseño de arquitectura. El núcleo de este juego es aplicar el método Attribute-Driven Design (ADD) [19]. ADD se centra en traducir los requisitos más importantes del sistema de software (también llamados drivers arquitectónicos) en un conjunto de estructuras a partir de las cuales se desarrolla el sistema. La traducción de los drivers arquitectónicos en estructuras es el proceso de diseño arquitectónico. ADD suele ser iterativo: se selecciona un subconjunto de controladores al comienzo de una iteración y luego se toman decisiones de diseño para identificar elementos y crear estructuras a partir de ellos para satisfacer a los drivers seleccionados. Seguidamente, se seleccionan otros drivers y se establecen más estructuras, o se refinan las estructuras existentes hasta que se crea una arquitectura inicial. El proceso de diseño consiste en tomar decisiones y, a menudo, éstas implican elegir entre soluciones comprobadas y documentadas para problemas de diseño recurrentes. Estas soluciones probadas, que llamamos conceptos de diseño, son los componentes básicos del diseño. Los conceptos de diseño pueden ser conceptuales, como patrones de diseño o tácticas, o más concretos, como marcos de aplicación.

DecidArch - Jugando a las cartas como arquitectos de software es un juego desarrollado para lograr tres objetivos de aprendizaje: 1) crear conciencia sobre la lógica involucrada en la toma de decisiones de diseño, 2) permitir la apreciación del razonamiento detrás de las decisiones de diseño candidatas propuestas por otros, y 3) crear conciencia sobre las interdependencias entre decisiones de diseño [61]. DecidArch es un juego de mesa que es económico y fácil de introducir en el aula para enseñar a los estudiantes universitarios sobre el concepto de toma de decisiones de diseño de arquitectura de software: examinar las compensaciones y los compromisos entre las demandas de las partes interesadas y los principales atributos de calidad, frente a una incertidumbre moderada. Los detalles y la mecánica del juego se describen en [61].

RPG Role Playing Game es un juego para soportar la enseñanza de ATAM (Architecture Trade-off Analysis Method) a estudiantes de informática ya sea en el salón de clase o a distancia. En este juego los estudiantes asumen algún de stakeholder (interesado), priorizan y examinan los atributos de calidad, negocian la prioridad y dificultad de los escenarios y acuerdan una arquitectura final. Este juego permite ejercitar habilidades de negociación. Los detalles y la mecánica del juego se describen en [78].

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos

Deseables:

- Sistemas operativos
- Sistemas distribuidos
- Ingeniería de software

Ejemplos de uso del patrón: Un docente necesita enseñar a los estudiantes el proceso de diseño hacia la satisfacción de atributos de calidad específicos. Los atributos de calidad son características o propiedades del sistema que afectan su comportamiento y rendimiento en áreas como la seguridad, la escalabilidad, la disponibilidad, el rendimiento, la modularidad, entre otros. Para este propósito necesita enseñar de manera divertida y amena cómo aplicar el método ADD (Attribute-Driven Design). El ADD se centra en identificar y abordar estos atributos de calidad desde las etapas iniciales del diseño arquitectónico.

ADD es valioso para garantizar que la arquitectura del software se diseñe de manera consciente y deliberada para cumplir con los requisitos de calidad específicos que son cruciales para el éxito del sistema. Este enfoque ayuda a los arquitectos de software a tomar decisiones informadas y a anticipar posibles desafíos relacionados con los atributos de calidad desde las etapas iniciales del desarrollo.

El docente más que enseñar el método ADD de manera explicativa quiere que los estudiantes lo pongan en práctica jugando. Para ello, organizará los estudiantes en equipos y los pondrá a jugar *Smart Decision* el cual tiene como núcleo aplicar el método Attribute-Driven Design (ADD). El docente conseguirá los requisitos de un nuevo sistemas para que a través del juego elegido los estudiantes traduzcan los requisitos más importantes del sistema de software (también llamados drivers arquitectónicos) en un conjunto de estructuras a partir de las cuales se desarrolla el sistema. A través de iteraciones los estudiantes seleccionarán un subconjunto de controladores al comienzo de una iteración y luego tomarán decisiones de diseño

para identificar elementos y crear estructuras a partir de ellos para satisfacer a los drivers seleccionados. Seguidamente, se seleccionarán otros drivers y se establecerán más estructuras, o se refinarán las estructuras existentes hasta que se crea una arquitectura inicial.

La mecánica del juego Smart Decision incluye las reglas del juego, los pasos y la puntuación. Las decisiones inteligentes requieren un facilitador que guíe a los jugadores a comprender la mecánica del juego mediante una presentación (el docente). El juego requiere un mínimo de dos jugadores y un máximo de seis que compiten entre sí. Estos jugadores pueden ser individuales o equipos. El juego se desarrolla en una serie de rondas donde cada ronda representa una iteración en el proceso de diseño de un sistema.

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Apéndice C: Tabla de Competencias):

Competencias obligatorias: C01, C08, C18.

Competencias opcionales: Ninguna.

Variantes para llevarlo a la práctica: Ninguna

Ventajas:

- Los juegos se centran en temas de arquitectura y se deja a un lado la implementación en código.
- Los juegos son una forma divertida de desarrollar habilidades de AS para los estudiantes.

Desventajas:

- No se interactúa con sistemas reales.

Patrones relacionados:

El patrón Games-Based Training se puede combinar en un curso de AS con otros patrones de formación que involucran el desarrollo de proyectos de software. Por ejemplo, Mini-Projects-based training, Large Project-Based Training, Open-source projects-based training, y In-house project-based training.

Experiencias reportadas del uso del patrón:

- Smart Decisions: An Architectural Design Game [19] describe la experiencia en el uso de un juego para enseñar ADD en un curso de AS.
- A role-playing game to teach ATAM (Ar- chitecture Trade-off Analysis Method) a

simulation tool and case study [78] describe la experiencia en el uso de un juego para enseñar ATAM en un curso de AS..

- DecidArch: Playing cards as software architects [61] presenta un juego para alcanzar concienciar sobre los fundamentos que intervienen en la toma de decisiones de diseño.
- LEARN Board Game: A game for teaching Software Architecture created through Design Science Research [113] propone un juego de mesa juego de mesa para la enseñanza de conceptos y estándares de AS de forma interactiva.
- DecidArch V2: An Improved Game to Teach Architecture Design Decision Making [31] muestra la segunda versión y los resultados del juego DecidArch para enseñar AS.

4.4. Lenguaje de patrones

Alexander et al. [6] definen un lenguaje de patrones como "una colección de patrones relacionados que captura todo el proceso de diseño y puede guiar al diseñador a través de directrices de diseño paso a paso". Siguiendo este concepto, se propuso una versión preliminar de un Lenguaje de Patrones de Arquitectura de Software para denotar un conjunto de patrones relacionados que colaboran dentro de los límites del diseño de cursos de SA. Si bien este lenguaje no se aborda en el alcance de la presente investigación, su necesidad se vuelve evidente una vez que se ha definido el catálogo de patrones de formación.

La Figura **4-10** muestra gráficamente el lenguaje propuesto donde los rectángulos muestran cada uno de los siete patrones de formación y el arco muestra la relación entre ellos. La relación *frecuentemente utilizada (often used)* significa que un patrón de formación puede utilizarse con otro en un mismo curso de AS. De esta forma, un profesor puede elegir para su curso varios patrones de formación. Cada patrón permitirá desarrollar un conjunto de habilidades de AS.

Los patrones de formación están divididos en dos grupos bien diferenciados. EL primer grupo corresponde a aquellos que extienden de un patrón de desarrollo dirigido por proyectos (projects-development-driven pattern). El segundo grupo, aquellos patrones que extienden de un patrón dirigido por toma de decisiones (decision-making-driven pattern).

Existen patrones de formación que resultan mutuamente excluyentes, lo que implica que, debido al esfuerzo requerido para su implementación tanto por parte del docente como de los estudiantes, no es posible aplicar más de uno simultáneamente. Este es el caso de los patrones dirigidos por proyectos. Por otro lado, los patrones de formación de la jerarquía de toma de decisiones no demandan un esfuerzo excesivo para su aplicación, permitiendo así su combinación con cualquier otro patrón de formación. Este segundo grupo de patrones puede ser integrado fácilmente en una única sesión de clase. De esta manera, un docente tiene

la opción de aplicar, por ejemplo, el patrón 'large project-based training' junto con otros patrones como 'Problem-solving-based training', 'Cases-based training', y/o 'games-based training'.

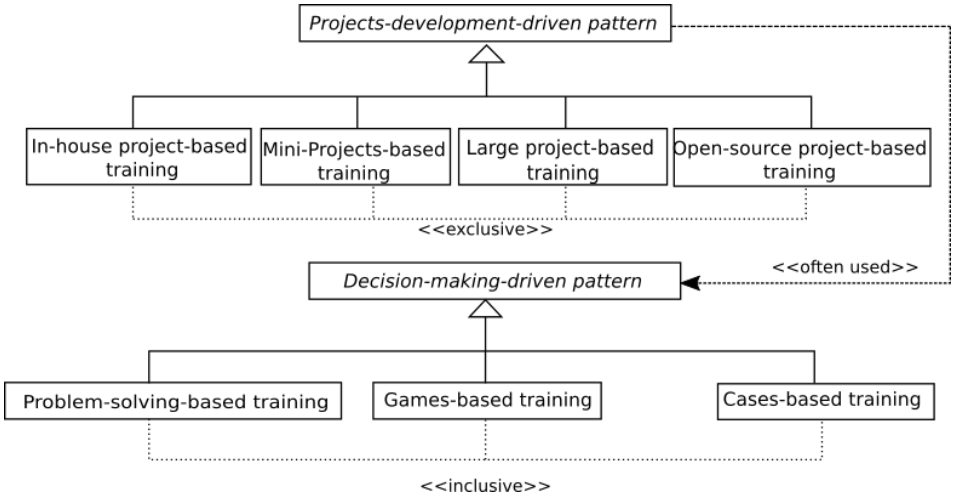

Figura 4-10.: Lenguaje de patrones de formación de AS

4.5. Guía de diseño de cursos de AS

La Guía de Diseño de Cursos de arquitectura de software (SAGITA - Software Architecture: GuIdeline for TrAining) permite al docente en cinco pasos diseñar su curso y seleccionar los patrones de formación adecuados (ver Figura **4-12**). SAGITA, además de ser un acrónimo, simboliza la flecha de un arco circular. De esta manera, evoca la dirección necesaria para alcanzar la meta, que en este caso es el logro de competencias en arquitectura de software **??**. Las tensiones del arco se definen por el contexto de los requisitos del curso y los desafíos inherentes a la formación en arquitectura de software. La dirección de la flecha se determina a través del catálogo de patrones de formación y el lenguaje de patrones.

A continuación explicamos cada uno de los pasos.

4.5.1. Paso 1: Preparar el curso

El primer paso es *preparar el curso* que involucra realizar las actividades previas a la planificación, lo cual involucra:

- *Tener en cuenta los prerrequisitos necesarios de la asignatura de arquitecturas de software y ubicar el semestre adecuado en el plan de estudios de cada Universidad.* Los

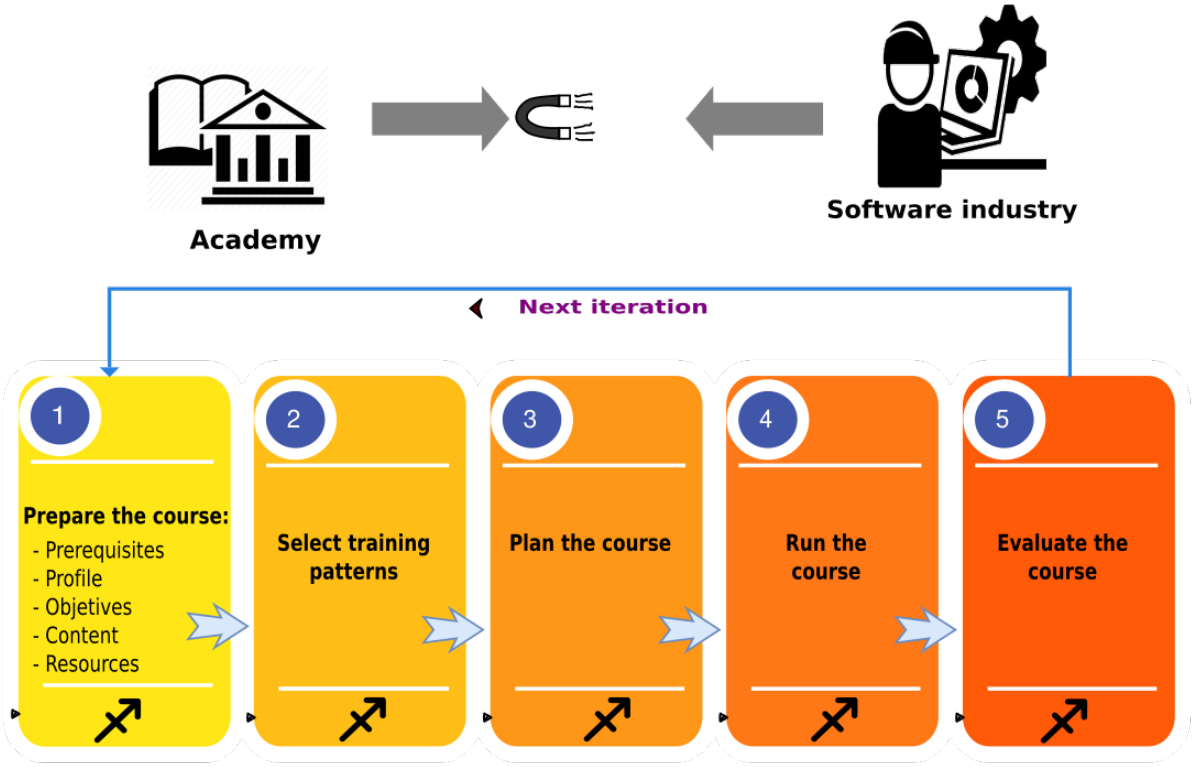

Figura 4-11.: Los cinco pasos de la Guía SAGITA

temas relacionados con AS se orientan en cursos de últimos semestres porque el proceso de diseño de la arquitectura involucra muchos conocimientos aprendidos en cursos previos, como programación orientada a objetos, estructuras de datos, redes, bases de datos, sistemas operativos, sistemas distribuidos e ingeniería de software. La AS considera y aplica en forma integral estos conocimientos previos [65]. Colocar el curso de diseño y arquitectura de software en semestres bajos hace que los estudiantes no tengan los conocimientos necesarios para asumir el curso. Se recomienda ubicar la asignatura al menos en el semestre VI para que el estudiante haya cursado la mayoría de prerrequisitos.

- *Definir el perfil que tendrá el curso*. El perfil depende del tipo de curso, por ejemplo, si es un primer curso de arquitectura, si es un curso electivo, un nuevo curso o una reforma curricular. Se puede definir un *perfil técnico* para que los estudiantes adquieran las habilidades técnicas de diseño de la arquitectura, tales como: la definición de características arquitectónicas (o atributos de calidad) de un sistema, la selección y aplicación de estilos arquitectónicos para un dominio del problema y la creación de los componentes del sistema. El curso puede ser de sólo arquitecturas de software, o puede ser una combinación de arquitectura y diseño detallado. En el segundo caso, se deben lograr habilidades en la aplicación de patrones de diseño los cuales permiten construir

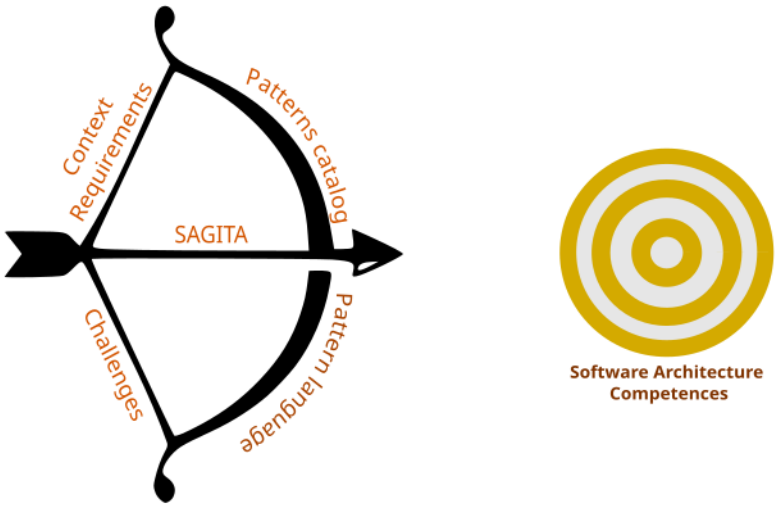

Figura 4-12.: SAGITA representa la dirección necesaria para alcanzar las competencias en arquitectura de software

aplicaciones flexibles ante cambios futuros. También, se puede tener un *perfil administrativo* que fomenta en los estudiantes la adquisición de las habilidades blandas de un arquitecto, tales como: comunicación, liderazgo y negociación y trabajo en equipo.

- *Definir los objetivos de aprendizaje*. Una vez definido el perfil del curso de Arquitecturas de Software se procede a definir los objetivos de aprendizaje. Se recomienda usar algunos de la siguiente lista:

  1. Analizar los requisitos de negocio para extraer y definir las características arquitectónicas.
  2. Lograr el consenso entre las partes interesadas a través de la capacidad de escuchar y resolver problemas.
  3. Aplicar los patrones de arquitectura más relevantes en la construcción de un sistema y conectarlos con los atributos de calidad.
  4. Documentar la arquitectura de un sistema de manera completa y precisa.
  5. Documentar las decisiones más importantes que han afectado la arquitectura proyecto utilizando las plantillas adecuadas para este propósito.
  6. Evaluar en forma sistemática la calidad del diseño de software.

- *Identificar el contenido central o fundamental que se abordará en el curso*. Una vez definidos los objetivos de aprendizaje se deben definir los contenidos fundamentales del curso. La Tabla **4-6** muestra una lista de posibles temas organizados por capítulos. Cabe mencionar que este contenido debe ser tomado como plantilla, el docente tiene

la libertad para tomar y modificar elementos de este listado.

Un libro actualizado que puede servir como referencia para la mayoría de los temas del curso es Fundamentals of software Architecture de Mark Richards [100].

- *Establecer los recursos disponibles*. Se debe tener en cuenta los recursos necesarios para diseñar el curso de arquitectura de software. Estos recursos los clasificamos como: humanos-docentes, humanos-clientes y tecnológicos-contenidos. Para los *recursos humanos-docentes* se debe contar con un equipo de docentes con conocimientos y experiencia en arquitecturas de software. Los antecedentes de los instructores deben incluir una gran cantidad de experiencia diversa del mundo real. Desafortunadamente, muchos profesores siempre han sido académicos y no tienen experiencia en el diseño de grandes proyectos reales [104]. Para enfrentar este reto se sugiere involucrar a estudiantes de maestría y doctorado que estén en contacto con el mundo de la industria de software [120]. Para los *recursos humanos-clientes* es fundamental contar con clientes reales. Sin embargo, hay dificultad para involucrar clientes y expertos de la industria [29]. Este tipo de personas siempre están ocupadas en sus organizaciones. Se recomienda involucrar como clientes de la industria egresados de las misma Universidades pues este tipo de personas siempre estarán dispuestas a colaborar con su alma mater [29]. Para los *recursos tecnológicos-contenidos* se requiere material de arquitectura de software actualizado a la realidad actual, tales como: presentaciones, resúmenes, ejercicios, talleres entre otros. En Internet la mayoría de recursos de este tipo están desactualizados, no se ajustan al concepto y realidad de la industria actual [39]. Para afrontar este desafío, se requiere disponer de un repositorio de material que esté siendo alimentado y actualizado constantemente por los mismos docentes. Una buena fuente de contenidos actualizados son los blogs, lecturas y documentos sobre las últimas tendencias arquitectónicas [55] [128] [128] [55].

4.5.2. Paso 2: Seleccionar e instanciar los patrones de formación del catálogo

El segundo paso es *seleccionar e instanciar los patrones de formación* que se utilizarán en el desarrollo del curso para alinearse con las necesidades de la industria de software. Los patrones de formación son los que están descritos en la Capítulo 4. La Tabla **4-7** sirve para definir los patrones de formación seleccionados por el docente para su curso.

4.5.3. Paso 3: Planificar el curso

El tercer paso es *planificar el curso* que involucra llevar a un documento las decisiones tomadas en los dos pasos anteriores. Como elementos importantes, se debe tener claros los

objetivos de aprendizaje establecidos, los contenidos y los patrones de formación elegidos.

La Tabla **4-8** muestra unas plantilla que le pueden servir al docente de guía para realizar la planificación de su curso de AS. Los campos de esta plantilla son:

1. *Semana No.* El número de la semana 1, 2, 3... según el período académico de cada institución.
2. *Temáticas*: se refiere a los temas a trabajar acorde al contenido de la asignatura.
3. *Actividades*: Actividades o metodologías a utilizar para tratar las temáticas, por ejemplo: Clase presencial, taller, debate, conferencia de invitado, socialización de trabajos, katas de arquitectura, estudio de caso, etc.
4. Observación: Cualquier observación por ejemplo, materiales a utilizar, recursos especiales, etc.

4.5.4. Paso 4: Ejecutar el curso

El cuarto paso es la *ejecución del curso* que involucra en llevar a cabo los aspectos definidos en la planificación, especialmente la ejecución de los patrones de formación. Debido a los imprevistos que se pueden presentar, lo planeado y lo ejecutado pueden tener algunas diferencias. Para la ejecución se sugiere llevar una bitácora de actividades ejecutadas igual o similar a la plantilla de planificación. Únicamente se requiere agregarle una columna con la fecha de ejecución de cada actividad tal como lo muestra la Tabla **4-9**.

4.5.5. Paso 5: Evaluar el curso

El quinto paso es evaluar el curso y los patrones de formación. Evaluar el curso es importante para conocer las prácticas que fueron efectivas y merecen repetirse en próximos cursos, pero también para conocer las prácticas que se deben mejorar. Se puede utilizar una encuesta que permita evaluar la satisfacción del curso, la utilidad y el logro de competencias. Podemos tomar como referencia las competencias de la Apéndice C. A continuación se sugieren algunas preguntas de dicha encuesta.

Estimado(a) estudiante, necesitamos su colaboración para evaluar su satisfacción del curso durante el periodo académico y el logro de las competencias adquiridas.

1. En términos generales que tan satisfecho quedó con el curso de Arquitectura de Software que acaba de cursar. 1->Nada satisfecho ... 5->Muy satisfecho
2. En términos generales que tan útiles le parecieron las estrategias utilizadas por el docente para acercar el curso a las exigencias de la industria de software actual (Charlas

con invitados, katas, aprendizaje basado en proyectos...). 1->Nada útiles ... 5->Muy útiles.

3. Qué tanto logró desarrollar la competencia C01: Identifica claramente los atributos de calidad relevantes del software que conducirán la arquitectura de un sistema de software a construir. 1->Poco desarrollada ... 5->Muy desarrollada.
4. Qué tanto logró desarrollar la competencia C02: Diseña consistentemente la arquitectura de software definiendo cómo los componentes interactúan entre si. 1->Poco desarrollada ... 5->Muy desarrollada.
5. Qué tanto logró desarrollar la competencia C05: Evalúa independientemente una arquitectura de software para determinar la satisfacción de los requisitos funcionales y no funcionales. 1->Poco desarrollada ... 5->Muy desarrollada.
6. Qué tanto logró desarrollar la competencia C08: Realiza imparcialmente un análisis de compensación (trade-off) para evaluar arquitecturas. 1->Poco desarrollada ... 5->Muy desarrollada.
7. Qué tanto logró desarrollar la competencia C11: Mantiene los sistemas existentes y su arquitectura para lograr la evolución de los sistemas software. 1->Poco desarrollada ... 5->Muy desarrollada.
8. Qué tanto logró desarrollar la competencia C12. Rediseña las arquitecturas existentes para la migración a nuevas tecnologías y plataformas. 1->Poco desarrollada ... 5->Muy desarrollada.
9. Qué tanto logró desarrollar la competencia C18. Analiza críticamente los requisitos de software funcionales y de atributos de calidad. 1->Poco desarrollada ... 5->Muy desarrollada.
10. Qué tanto logró desarrollar la competencia C22. Realiza periódicamente revisiones del código fuente escrito por el equipo de desarrollo. 1->Poco desarrollada ... 5->Muy desarrollada.
11. Qué tanto logró desarrollar la competencia C23. Desarrolla componentes de software reutilizables. 1->Poco desarrollada ... 5->Muy desarrollada.
12. Qué tanto logró desarrollar la competencia C28. Diseña e implementa procedimientos de prueba considerando aspectos de la arquitectura (tipos de componentes/servicios, integración). 1->Poco desarrollada ... 5->Muy desarrollada.

Finalizados estos pasos se vuelven a repetir para iniciar otro ciclo completo de formación, es decir, que el curso se irá mejorando en cada ciclo.

No	Capítulo	Temas
1	Introducción	Qué es la arquitectura de software. La importancia de la arquitectura de software. Qué influye en la arquitectura de una aplicación. Arquitectura vs diseño detallado. Qué debes saber para formarte como arquitecto de software. Principios SOLID.
2	Atributos de calidad	Conociendo los atributos de calidad del software. Cómo identificar los atributos de calidad de un sistema a construir. Recomendaciones para seleccionar los atributos de calidad de un sistema. Taller de atributos de calidad - QAW (Quality Attribute Workshop).
3	Patrones de arquitectura de software	Los patrones de arquitectura más relevantes de una aplicación y su relación con los atributos de calidad: Arquitectura en capas, Arquitectura microkernel, Arquitectura monolítica, Monolitos modulares, Arquitectura orientada a eventos, Arquitectura orientada a servicios, Arquitectura orientada a microservicios, Arquitecturas limpias. Cómo seleccionar los patrones de arquitectura de una aplicación. Resolviendo problemas de arquitectura de software (Katas de arquitectura).
4	Diagramas de arquitectura de software	La importancia de diagramar. Acoplamiento y cohesión de componentes. Herramientas y técnicas para modelar la arquitectura de una aplicación. Creando la arquitectura de un aplicación mediante los diagramas del modelo C4: Contexto, contenedores y componentes. Creando la arquitectura de una aplicación mediante vistas 4+1.
5	Documentación de arquitectura	Introducción a la documentación. Guia de Simon Brown. Arc42. Registro de decisiones de arquitectura.
6	Evaluación de la arquitectura	Revisión por pares. Mini ATAM. DCAR.

Tabla 4-6.: Contenido de un curso de Arquitecturas de Software para programas de pregrado

No	Patrón de formación	Motivo o razón
1	Problme-solving traingin	Me parece interesante aplicar las katas de arquitectura.
2	...	...

Tabla 4-7.: Plantilla definir los patrones de formación elegidos por el docente.

Semana No	Temática	Actividades	Observación
1	Presentación del curso. Reglas de juego	Clase presencial	
2	Qué es la AS. La importancia de la AS. Qué influye enla AS	Aula invertida	Lectura y video para el aula invertida
3	...	...	...
4	Charla 1: arquitecto de software	Conferencia remota invitado de la industria	Pendiente concretar invitado
6	Taller de preparación del primer parcial. Primer parcial	Trabajo en grupos. Primer parcial	
7	Arquitectura basada en microservicios. Arquitecturas limpias	Clase magistral	Lecturas adicionales
8	...	...	...
12	Katas de arquitectura	Problem-solving-based training	Grupos de tres personas, se trabajarán dos ejercicios de katas.
13	...	...	...
16	Examen Final	Examen en grupos de dos	Previo se dejará un taller de preparación

Tabla 4-8.: Plantilla de planificación del curso de AS con algunos ejemplos para guiar al docente.

Semana No	Fecha	Temática	Actividades	Observación

Tabla 4-9.: Plantilla de planificación del curso de AS.

5. Validación de SAGITA y el catálogo de patrones

Resumen

En esta sección se presenta la validación de la hipótesis a través de la guía de diseño de cursos SAGITA y los los patrones de formación propuestos. Se utilizó varios mecanismos para ello.

5.1. Introducción

La validación de la hipótesis se realizó utilizando varios mecanismos, cada uno en un ciclo de investigación-acción. En resumen, se involucró validaciones por expertos, la realización de un experimento y tres estudios de caso. Los estudios de caso corresponden a cursos de arquitectura de software en distintas universidades donde se utilizó SAGITA para el diseño del curso. El diseño de los estudios de caso fue realizado bajo los lineamientos propuestos por Runeson y Host [103]. Se tuvo en cuenta que los docentes implicados en los cursos estuvieran dispuestos a probar SAGITA y las patrones de formación motivados por mejorar los cursos y alinearlos con las necesidades de la industria.

La Figura **5-1** muestra una línea de tiempo con los mecanismos de validación empleados.

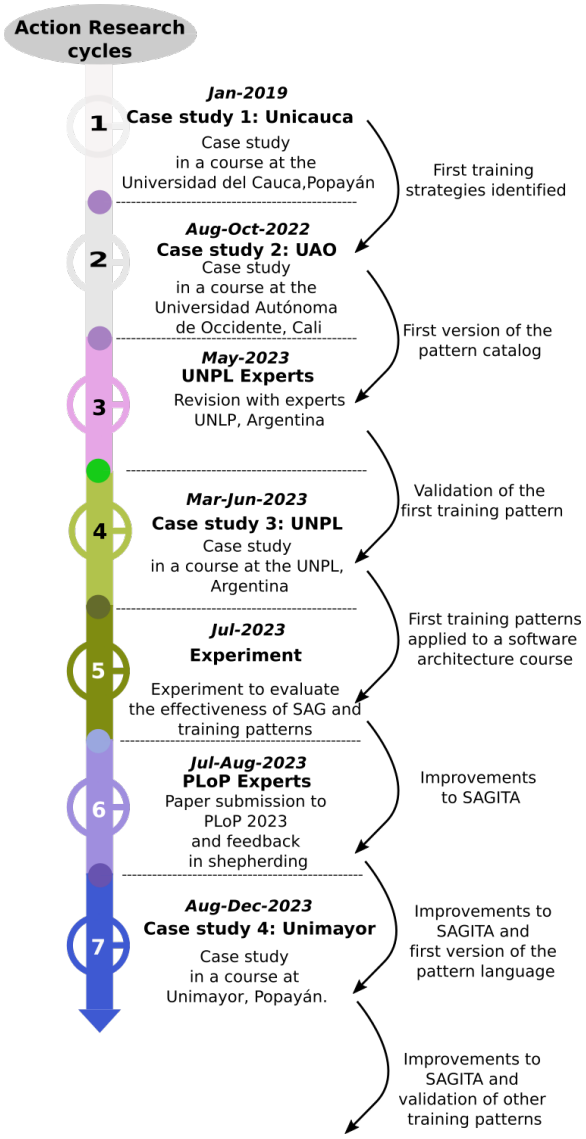

Figura 5-1.: Línea de tiempo de los distintos mecanismos de validación

Durante la ejecución de cada uno de los mecanismos de validación se fueron encontrando problemas, errores y hallazgos que permitieron ir refinando la guía SAGITA y el catálogo de patrones de formación. Las siguientes secciones describen los detalles de cada uno de estos mecanismos.

5.2. Estudio de caso 1: Curso en la Unicauca

La presente investigación sobre formación en AS comenzó a inicios de 2019, donde se reestructuró un curso de Ingeniería de Software II del Programa de Ingeniería de Sistemas la

Universidad del Cauca. Se buscó alinear las competencias que se esperan de los arquitectos de software los profesionales de la industria, con las prácticas docentes que imparten esas competencias. El resultado de este trabajo se plasmó un tiempo después en el artículo "Aligning Software Architecture Training with Software Industry Requirements"' [91]. En esta reestructuración se realizaron los siguientes pasos:

- Se buscó en la literatura los desafíos que tiene la formación en AS. Cada uno de estos desafíos fue debidamente etiquetado.
- Del listado anterior de desafíos, se eligió aquellos que se presentan en el curso de Ingeniería de Software II.
- En seguida, se diseñaron y aplicaron estrategias para tratar cada desafío. Se intentó abarcar la mayoría de desafíos.
- Se analizó y reportó el impacto que en tuvo estas estrategias al final del curso.

De esta manera se logró mejorar el curso buscando una alineación con las necesidades de la industria. Tal alineación es necesaria para mejorar la empleabilidad de los estudiantes de pregrado, hacer que su progresión de los institutos académicos a las industrias sea sin esfuerzo, y para obtener mejores resultados de aprendizaje. En ausencia de tal alineación, los recién egresados tienen que volver a formarse, lo que conlleva costos de formación y retrasos en la contratación de los recién egresados por parte de sus posibles empleadores. A partir de de esta reestructuración del curso se inició un trabajo de investigación más profundo para comprender el problema y buscar una solución más integral.

5.3. Estudio de caso 2: Curso en la UAO

Esta sección describe un estudio de caso exploratorio que permitió examinar el impacto que tuvo la aplicación de la Guía de Diseño de Cursos de AS SAGITA (ver Sección 4.5) en un curso de Ingeniería de Software de la Universidad la Universidad Autónoma de Occidente de la ciudad de Cali (UAO) - Colombia. La selección de este estudio de caso se hizo por ser un caso representativo, es decir, un curso de pregrado en el que el Diseño y la Arquitectura de Software son el eje principal. Por otro lado, se aprovechó la pasantía realizada en la Institución.

5.3.1. Diseño del estudio de caso

Objetivo

El objetivo de este estudio de caso es obtener una evidencia empírica y exploratoria del impacto que tuvo para el docente y para los estudiantes aplicar la Guía propuesta a un curso de AS.

Preguntas de investigación

RQ1: ¿Qué impacto tienen las estrategias de formación elegidas en el desarrollo curso de Ingeniería de Software de la UAO?

RQ2: ¿Cuál fue la utilidad percibida y la facilidad de uso de la Guía por el docente en este estudio de caso?

Contexto del estudio

Este caso fue planeado y ejecutado en el segundo periodo académico 2022 del curso de Ingeniería de Software I del programa de Ingeniería Informática de la Universidad Autónoma de Occidente, sede Cali. Para este momento la Guía estaba en una versión temprana y en lugar de patrones de formación, tenía un listado de estrategias de formación reportados por la revisión de la literatura. Sin embargo, se pudo explorar de manera cualitativa el impacto que tuvo esta Guía para el desarrollo de curso.

El estudio de caso se clasifica como de *mejora*, pues trata de mejorar ciertos aspectos en el fenómeno estudiado, en decir, en el curso de Ingeniería de Software.

La asignatura de Ingeniería de Software I está dividida en tres cortes. El primero, dedicado a la ingeniería de requisitos. El segundo corte abarca los temas de diseño, por ejemplo, arquitectura de software, estilo de arquitectura en capas y MVC y diagramación de la arquitectura. En el curso no se alcanza a trabajar estilos de arquitectura distribuidos.

5.3.2. Preparación de la recolección de datos

Las técnicas usadas para la recolección de datos fueron análisis de documentos y la entrevista semi-estructurada[1]. Las preguntas de la entrevista fueron las siguientes:

[1]En una entrevista semi-estructurada las preguntas son planeadas pero no necesariamente son desarrolladas en el orden que se estableció, sino que durante el desarrollo de la conversación puede decidir el orden [103]

1. ¿Cómo se llama la asignatura, en qué semestre de dicta, de qué programa(s) la matriculan?
2. Describa qué temas de arquitectura de software se tratan en la asignatura.
3. Describa cómo es la estrategia de aprender arquitectura a través de un proyecto real de empresa.
4. ¿Tiene alguna encuesta que permitió medir la satisfacción del cliente con el proyecto desarrollado?
5. ¿Tiene alguna encuesta que permitió medir la satisfacción de los estudiantes con la implementación del proyecto de clase?
6. ¿Qué ventajas tiene su estrategia de trabajar un proyecto real?
7. ¿Qué dificultades tiene su estrategia de usar proyectos reales?
8. Considera que las estrategias propuestas en la guía ¿permiten acercar la formación en arquitectura de software al mundo real de la industria de software?
9. ¿Qué le hace falta a la guía propuesta para que cumpla su propósito de proponer un conjunto de pasos y estrategias de apoyo en el diseño y ejecución de cursos de arquitecturas de software a nivel de pre-grado?
10. Qué otras estrategias ha empleado para enseñar arquitectura de software que puedan ser replicables por otros docentes en otras universidades?

5.3.3. Ejecución del estudio de caso

El estudio de caso se ejecutó aprovechando una estancia de investigación llevada a cabo en Universidad Autónoma de Occidente entre el 16 de agosto hasta el 17 de octubre de 2022. Además durante esta estancia se trabajó en la primera versión del catálogo de patrones de foramación (ver Catálogo de patrones de formación).

La asignatura de Ingeniería de Software I es de carácter obligatoria, tiene 3 créditos y se divide en 2 sesiones de 1 hora 30 minutos a la semana. Tiene como pre-requisitos programación orientada a objetos y diseño conceptual. Esta asignatura es la primera aproximación al desarrollo formal de un sistema software. Proporciona a los estudiantes los fundamentos teóricos y prácticos para desarrollar software que sea confiable, seguro, trabaje con eficiencia y se ajuste a las necesidades de los interesados, siguiendo un modelo de proceso sistemático y disciplinado de la Ingeniería de Software.

Inicialmente al docente del curso se le dio a conocer la Guía SAGITA. El docente leyó detenidamente la Guía y aportó algunos comentarios para mejorarla. En seguida, el docente

aplicó la guía a su curso e hizo las mejoras respectivas. La Guía le permitió incluir algunos temas de AS (principios SOLID y patrones de diseño básicos como Fábrica de Método, Singletón, Observer y Facade) y elegir las estrategias para acercar el curso a las realidades de la industria de software.

Durante el desarrollo del curso, el investigador interactuó con el docente y los estudiantes en varias ocasiones tanto de manera presencial como a través de video-conferencias.

Al final del curso, se aplicó la entrevista semi-estructurada al docente con el fin de tener datos para responder a las preguntas de investigación. Algunas preguntas permiten describir el desarrollo del caso.

5.3.4. Resultados y análisis

Haciendo una triangulación de los datos dimos respuesta a las preguntas de investigación. La transcripción de la entrevista de la Subsección 5.3.2 se encuentra en el Apéndice D.

RQ1: ¿Qué impacto tienen las estrategias de formación elegidas en el desarrollo curso de Ingeniería de Software de la UAO?

El docente aplicó la estrategia de realizar un *proyecto real para una empresa*. Esta estrategia ha venido siendo aplicada hace un par de períodos académicos. El docente hace los contactos previos con la empresa para que genere un proyecto pequeño de tal forma que pueda ser implementado en el curso. El canal de comunicación con la empresa, en esta ocasión, fue el arquitecto de software de la organización quien se encargó de dar los requisitos y evaluar los prototipos presentados por los estudiantes. Las decisiones de qué estilo de arquitectura, el tipo de tecnología es la adecuada y el diseño de la arquitectura estuvieron a cargo de los estudiantes, pero fueron guiados por el docente y el arquitecto de la empresa. El proyecto se desarrolló en equipos de 3 estudiantes.

El docente y la empresa decidieron que el equipo que presente la mejor solución tiene la opción de seguir trabajando en la empresa. Esta incorporación a la industria se hace mediante una prueba que los estudiantes presentan ante la empresa una vez finalizado el proyecto. Generalmente los estudiantes se vinculan medio tiempo y reciben capacitaciones posteriores.

El proyecto de clase se trabajó con la empresa OL Software de la ciudad de Cali. Esta empresa generó un proyecto pequeño de tal forma que pueda ser implementado en el curso. El proyecto trabajado es un Mapa de Conocimiento.

El proyecto realizado tuvo un enfoque iterativo e incremental. En la primera entrega los estudiantes entregaron las historias de usuario y unos prototipos de la interfaz de usuario de baja fidelidad. De esta entrega hubo una realimentación por parte de la empresa. La segunda

entrega fue un prototipo software funcional. Finalmente, la tercera entrega fue el producto software implementado en su totalidad. En cada una de las entregas existieron criterios de evaluación por parte del docente. En la tercera entrega fue la única que tuvo criterios de evaluación de la empresa, por ejemplo, si el producto se adecuó a sus necesidades, originalidad, innovación del producto y calidad de la presentación oral que hacen los estudiantes.

En general la aplicación del proyecto con una empresa real ha tenido un impacto positivo sobre los estudiantes. El proyecto ha permitido que los estudiantes lleven a la práctica los conocimientos teóricos de ingeniería de software, incluida la arquitectura de software. Aunque el proyecto no es grande, los estudiantes ganaron experiencia práctica y enfrentan los desafíos reales que se presentan en el desarrollo de software del mundo real.

Por otro lado, los estudiantes se ven beneficiado de esta estrategia porque trabajan en equipo y desarrollan habilidades blandas como comunicación, liderazgo, solución de conflictos y coordinación de esfuerzos para lograr objetivos comunes.

RQ2: ¿Cuál es la utilidad percibida y la facilidad de uso de la Guía por el docente en este estudio de caso?

Según el docente, la guía es fácil de usar, los pasos son simples y claros. Igualmente, la utilidad es positiva porque ayuda a planificar y mejorar los cursos relacionados con AS.

Ante la pregunta *¿Qué le hace falta a la guía propuesta para que cumpla su propósito de proponer un conjunto de pasos y estrategias de apoyo en el diseño y ejecución de cursos de arquitecturas de software a nivel de pre-grado?* la respuesta fue "Considero que a la guía no le falta algo concreto, es una hoja de ruta completa. Lo que haría falta para que el curso de arquitectura se imparta de la mejor forma posible, es contar con aliados del sector productivo para que los casos y problemas correspondan con los retos de las empresas".

5.3.5. Discusión del estudio de caso

Utilizar un proyecto de software real para enseñar ingeniería de software y arquitectura de software brindó a los estudiantes una oportunidad valiosas para aplicar los conceptos teóricos y generar habilidades de trabajo en equipo, liderazgo y comunicación. A pesar que los estudiantes no son de semestres avanzados, pues es un primer curso de Ingeniería de Software I, el docente ha logrado coordinar adecuadamente los temas del curso para irlos aplicando estratégicamente al proyecto y generar resultados positivos.

El estudio de caso en la UAO de Cali, evidencia resultados positivos en cuanto a la utilidad, facilidad de uso de la Guía. En general, la Guía tuvo una buena aceptación y logró impactar positivamente en el desarrollo del curso. Sin embargo, hasta este punto, la guía tiene única-

mente estrategias en lugar de patrones de formación, concepto que surge más adelante. La estrategia de usar proyectos reales nos dio un indicio que es posible especificar las estrategias como patrones de formación.

5.4. Validación del primer patrón de formación con expertos de la UNLP

Hasta este punto de la investigación las estrategias de formación evolucionaron hacia patrones de formación reutilizables. Con una primera versión de este catálogo de patrones de formación, se eligió uno de los patrones para ser sometido a una validación con profesores de la Universidad Nacional de la Plata, Argentina. En este ciclo se buscó recibir realimentación en cuanto a los siguientes elementos [76]:

- Que el patrón sea fácil de leer, entender y aplicar.
- Que el nombre del patrón sea el adecuado.
- Que la estructura de los patrones esté completa, es decir, indagan si faltan o sobran campos en el formato propuesto.
- Que la relación entre el problema y la solución sea coherente.

Esta actividad de validación pasó por tres etapas: la planificación, la ejecución y el análisis de los resultados.

5.4.1. Planificación del encuentro

El encuentro se planificó teniendo en cuenta el objetivo del mismo, la logística, el perfil de las personas a invitar, el lugar, fecha, hora y las estrategias de motivación a los invitados. El Dr Luis Mariano Bibbó, profesor de la Facultad de Informática de la UNLP, jugó el rol de investigador de esta experiencia y facilitador de la logística.

Esta validación se llevó a cabo aprovechando una estancia de investigación en el Laboratorio de Investigación y Formación en Informática Avanzada (LIFIA) de la Universidad Nacional de la Plata (UNLP), Argentina en los meses de marzo y abril de 2023. El perfil que buscado fue docentes de la Facultad de Informática que trabajan temas de ingeniería de software y/o lenguaje de patrones.

El taller se planificó para ser llevado a cabo en un tiempo máximo de una hora y media. Elegimos como moderador al profesor Federico Balaguer, quien tendría como función conducir los pasos durante el encuentro. En la apertura uno de los autores daría una introducción

corta de 5 minutos indicando el propósito del encuentro y las reglas de juego. En seguida vendría la intervención de cada uno de los cuatro evaluadores. Los autores de los patrones no tendrían derecho a respuesta, solamente recibir las observaciones. EL taller sería grabado del audio para facilitar el posterior análisis.

Luego se preparó un documento con una introducción al tema de investigación y la descripción de uno de los patrones de formación. Elegimos el primer patrón "Mini-Projects-based training" por ser el más genérico para que la realimentación recibida pudiera ser replicada a los demás patrones. El documento fue enviado a los expertos evaluadores mediante email dos semanas antes del encuentro.

En seguida, hicimos el listado de los docentes de la UNLP que cumplieran el perfil para invitarlos al taller mediante e-mail. La Tabla **5-1** muestra el listado de los docentes invitados y sus áreas de interés. Debemos resaltar que la mayoría de ellos tienen una trayectoria reconocida a nivel mundial en temas de objetos, lenguajes y patrones y han tenido experiencias como evaluadores en la Conference on Pattern Languages of Programs (PLOP) [2]. Decidimos hacer el encuentro de manera presencial, en una sala de reuniones de LIFIA, el martes 2 de Mayo de 2023. La motivación la hicimos con una charla corta previa aprovechando un espacio dado en las reuniones mensuales del grupo.

5.4.2. Conducción del encuentro

En esta etapa se ejecutaron los pasos previamente definidos en la planificación. Durante el taller el investigador dio una breve introducción del patrón a analizar y en seguida cada uno de los asistentes debatieron en profundidad sus comentarios, dudas e impresiones.

El taller se realizó en un tiempo de una y media y asistieron 4 docentes (ver Figura **5-2**). En general, se logró el propósito general del taller y se obtuvo observaciones y sugerencias valiosas para mejorar los patrones.

[2]Pattern Languages of Programs (PLoP) conference, es un evento de primer orden en el que autores y entusiastas de los patrones se reúnen, debaten y aprenden más sobre patrones, diseño, desarrollo de software y el mundo de la construcción en general.

Tabla 5-1.: Docentes que participaron en la validación preliminar del primer patrón de formación

No	Nombre	Título	Áreas de interés
1	Federico Balaguer	Doctor en Ciencias de la Computación, University of Illinois Urbana-Champaign	Algoritmos, Lenguajes de programación, Ingeniería de software
2	Julián Grigera	Doctor en Ciencias Informáticas, UNLP	Ingeniería Web, Usabilidad, Metodologías de Desarrollo Dirigidas por Modelos, Metodologías Ágiles de Desarrollo
3	Alejandro Fernández	Doctor en Ciencias Univesität Hagen, Alemania,	CSCW, Gestión del Conocimiento, Ingeniería de Software.
4	Alejandra Garrido	Doctora en Ciencias de la Computación, University of Illinois Urbana-Champaign	refactoring, experiencia de usuario.

Figura 5-2.: Validación del primer patrón de formación con expertos

5.4.3. Análisis de los resultados

Los evaluadores resaltaron la utilidad del patrón para diseñar cursos de AS. Durante este proceso rico en recomendaciones y sugerencias, recibimos comentarios muy acertados :

- Mejorar la redacción del problema del patrón para alinearlo con la solución.
- Aclarar algunos conceptos que generan dudas como el nivel inicial de los estudiantes, proyecto pequeño, problemas de formación, atributos de calidad, tácticas de arquitectura, etc.
- Mejorar la redacción de las fuerzas del patrón teniendo en cuenta las perspectivas del docente, la industria y los estudiantes.
- Proponer una figura más simple que represente los principales elementos del patrón de tal forma que haya una conexión entre el texto y la figura.
- El ejemplo del patrón está muy largo, se debe simplificar su redacción.
- Corregir algunos errores de redacción.

Todas las sugerencias y comentarios de los cuatro evaluadores fueron analizadas y aplicadas, obteniendo una nueva versión del catálogo de patrones de formación. En esta nueva versión se se hizo énfasis en mejorar el problema, la solución y las fuerzas. Las fuerzas son un elemento muy importante de un patrón porque son las que representan un escenario concreto que motiva al uso del patrón. A partir de esta experiencia se aplicó lo aprendido a los demás patrones de formación del catálogo.

5.5. Estudio de caso 3: Universidad Nacional de La Plata

Esta sección describe un estudio de caso que permitió evaluar el impacto que tuvo la aplicación de SAGITA en un curso de AS de la Universidad Nacional de la Plata, Argentina. Además, se destaca que este curso se sincronizó en ciertas actividades con otro curso de la Universidad del Cauca.

La selección de este estudio de caso se hizo por ser un caso representativo, es decir, un curso de pregrado de Arquitectura de Software. Por otro lado, se aprovechó la pasantía internacional realizada en esta Institución.

5.5.1. Diseño del estudio de caso

Objetivo

El objetivo de este estudio de caso es obtener una evidencia empírica del impacto que tuvo para el docente y para los estudiantes aplicar SAGITA a un curso de AS.

Preguntas de investigación

RQ1: ¿Qué impacto tuvo los patrones de formación en el desarrollo de habilidades de los estudiantes al final del curso?

RQ2: ¿Cómo fue evaluada la utilidad percibida y la facilidad de uso de la Guía por el docente en este estudio de caso?

RQ3: ¿Cómo fue la satisfacción del curso por parte de los estudiantes?

Contexto del estudio

Este caso se llevó a cabo en el primer periodo académico 2023 del curso optativo de Patrones de Arquitectura de Software de la Facultad de Informática de la Universidad Nacional de la Plata, Argentina. Este curso fue cursado por 14 estudiantes de las carreras de Licenciatura en Informática y Licenciatura en Sistemas. El curso tiene como asignaturas prerrequisito Proyecto de Software, Orientación a Objetos 2 e Ingeniería de Software 2. El estudio de caso es de *mejora*, pues trata de mejorar ciertos aspectos en el fenómeno estudiado como son las competencias de los estudiantes en temas de AS.

Los objetivos de aprendizaje del curso son:

- Aprender a identificar tempranamente las decisiones que hacen a la arquitectura que por definición son costosas de modificar en el futuro.
- Conocer los desafíos y problemas que dan lugar a la evolución de los estilos de arquitectura.
- Conocer ventajas y desventajas de cada uno de los estilos arquitectónicos vigentes.
- Aprender a poner en contexto cada estilo arquitectónico con el objetivo de elegir el más adecuado.

A diferencia del estudio de caso de la Universidad Autónoma de Occidente Sección 5.3, ya estaban identificados los patrones de formación. Para este estudio de caso SAGITA estaba en una versión mejorada gracias a la la validación con expertos de la UNPL. Las mejoras

se centraron principalmente en la redacción y coherencia de la descripción del problema, la solución propuesta y las fuerzas.

Además, en este curso se decidió llevar a cabo una experiencia de enseñanza colaborativa con el curso de Ingeniería de Software II de la Universidad del Cauca. Durante la planificación del curso, se compararon los objetivos de aprendizaje, currículos, técnicas de enseñanza y estrategias utilizadas por las dos Universidades, para encontrar puntos comunes que permitieron realizar un trabajo conjunto para mejorar la experiencia de formación de los estudiantes.

5.5.2. Preparación de la recolección de datos

Las técnicas usadas para la recolección de datos fueron las encuestas, análisis de documentos y la entrevista semi-estructurada [3]. Las preguntas de la entrevista aplicadas al docente al finalizar el curso fueron:

1. ¿Una vez leída la Guía de diseño de cursos de AS, describa qué cambios hizo en su curso a nivel de contenido, objetivos, metodología?
2. ¿La planificación de su curso de AS mediante la guía fue de utilidad, vale la pena utilizarla, por ejemplo, le ahorró tiempo y esfuerzo, o por el contrario le agregó más trabajo?
3. ¿La Guía a través de las pasos que plantea para diseñar un curso de AS, es fácil de seguirla, tiene pasos complejos de entender?
4. ¿Acorde a la Guía de Diseño de Cursos de AS, qué patrones de formación decidió aplicar en su curso y por qué?
5. ¿Explique cómo implementó (en qué momento del semestre, con qué recursos humanos y técnicos) los patrones de formación?
6. ¿Qué impacto cree que tuvieron estos patrones en el desarrollo de habilidades de sus estudiantes a nivel de creación, documentación y evaluación de la AS?
7. ¿Qué dificultades tuvo aplicando los patrones de formación durante el desarrollo del curso?
8. ¿Considera que los patrones de formación que empleó, permiten acercar la formación en arquitectura de software al mundo real de la industria de software?
9. ¿Qué le hace falta a la Guía propuesta para que cumpla su propósito de proponer un conjunto de pasos y estrategias de apoyo en el diseño y ejecución de cursos de

[3] En una entrevista semi-estructurada las preguntas son planeadas pero no necesariamente son desarrolladas en el orden que se estableció, sino que durante el desarrollo de la conversación puede decidir el orden [103]

arquitecturas de software a nivel de pre-grado?

5.5.3. Ejecución del estudio de caso

El estudio de caso se ejecutó durante el primer período académico de 2023 aprovechando una estancia de investigación llevada a cabo en la Universidad Nacional de la Plata entre el 15 de marzo y el 15 de mayo de 2023. Por lo tanto, el investigador asistió a las clases, las cuales fueron algunas de manera presencial y otras de manera virtual. El curso de Patrones de Arquitectura fue impartido a un total de 14 estudiantes.

Es crucial destacar que, tanto previo como posterior a la realización de este estudio de caso, se llevó a cabo un trabajo adicional. Antes del período de pasantía, se realizaron reuniones con los miembros del grupo de investigación de La Plata para coordinar las actividades planificadas, así como la logística necesaria. Del mismo modo, después de llevar a cabo el estudio de caso, se continuó trabajando en la optimización de los patrones de formación y en la redacción de artículos

Inicialmente al docente del curso se le dio a conocer la Guía SAGITA. La guía sirvió para armar una planificación del curso. En especial sirvió para definir los temas a desarrollar, los recursos a utilizar, elegir y aplicar los patrones de formación y definir el proyecto de clase Open Market. Finalmente se organizaron los temas obligatorios que se darían en los dos países como fueron principios SOLID, estilos arquitectónicos y con mayor énfasis el estilo de microservicios.

En cuanto a recursos humanos, debido a la complejidad del curso se organizó un grupo de varias personas. El primero, el profesor Luis Mariano Bibbó encargado de dictar las clases teóricas. Además, contó con dos colaboradores. El primero, el ingeniero Marcelo Barreto, especialista en Dev-Ops y el segundo, el ingeniero Agustín Ortú especialista en diseño con Objetos y Java.

Respecto a los patrones de formación se aplicaron varios. En la Universidad del Cauca se eligió como estrategia central el patrón de formación Mini-Projects-based training. La Universidad Nacional de la Plata aplicó una estrategia de formación distinta mediante el patrón Large Project-based Training enfocado a migrar una aplicación monolítica a una solución basada en micro-servicios.

Las dos universidades aplicaron el patrón Cases-based training mediante la variante de charlas con arquitectos de software invitados de la industria. A lo largo del curso se organizaron tres conferencias donde cada invitado expuso los detalles de un caso real de arquitectura: el contexto, el problema, las decisiones tomadas y los resultados. De esta forma los estudiantes aprendieron de la experiencia vivida por los arquitectos. Las tres conferencias fueron:

1. Proyecto E-sidif del ministerio de Economía de Argentina, a cargo del ingeniero Leandro

Quiroga.

2. Estudio de caso del módulo de contracargos de Mercado Libre, migrando un monolito a microservicios, a cargo de los ingenieros Joaquín Alem y Leandro Zoi.
3. Caso de arquitectura de gran escala Banco Galicia, a cargo del ingeniero Santiago Urrizola.

Como actividad integradora, en la parte final del curso (tercer corte) se llevó a cabo una experiencia de desarrollo global fusionando los equipos de trabajo de las dos instituciones educativas en una iteración de desarrollo. La planificación permitió sincronizar los cursos de Colombia y Argentina para que llegaran simultáneamente al tercer corte con los conocimientos y habilidades necesarias.

Las dos Universidades trabajaron con un mismo proyecto de clase, una aplicación de E-commerce similar a Mercado Libre+Delivery que se llamó OpenMarket (ver Apéndice I). Sin embargo, la Universidad del Cauca hizo su propia versión del proyecto, una versión más simple para poder adaptarlo al desarrollo de mini-proyectos (ver Apéndice J).

Un detalle de mucha utilidad fue grabar las clases y las charlas en la plataforma `https://unlpeduar.webex.com/unlpeduar`. Esto permitió que los alumnos y docentes de los dos países pudieran acceder a los contenidos teóricos y prácticos en cualquier momento. El listado de clases grabadas se lo puede acceder en el siguiente enlace: listado de clases grabadas.

Al final del curso se aplicó la entrevista al docente Luis Mariano Bibbó, la transcripción de la entrevista está en el Apéndice E.

5.5.4. Resultados y análisis

Haciendo una triangulación de los datos recolectados a través de las entrevistas, encuestas y análisis de documentos, podemos dar respuesta a las preguntas de investigación.

RQ1: ¿Qué impacto tuvo los patrones de formación en el desarrollo de habilidades de los estudiantes al final del curso?

El patrón de formación Large Project-based Training permitió que los estudiantes llevaran a la práctica los conocimientos teóricos de AS mediante el desarrollo de un proyecto grande y complejo. El proyecto Open Market fue desarrollado en equipos de 4 y 5 estudiantes, quienes realizaron ingeniería de requisitos, propusieron la arquitectura del sistema, tomaron decisiones de diseño y se comunicaron efectivamente para realizar la implementación y pruebas. Aunque el software no contó con clientes reales, los requisitos fueron inspirados en aplicaciones modernas de comercio electrónico como Mercado Libre.

En general la aplicación de este proyecto de desarrollo tuvo un impacto positivo entre los estudiantes. El desarrollo del curso giró entorno a este proyecto. A continuación se describen aspectos relevantes:

- *Experiencia práctica.* Los estudiantes ganaron experiencia práctica al trabajar en este proyecto grande, lo que les permitió aplicar los conceptos teóricos aprendidos en un entorno práctico y enfrentar desafíos reales que se presentan en el desarrollo de software a gran escala.
- *Trabajo en equipo.* Este proyecto de software requirió de la colaboración entre los miembros del equipo. Los estudiantes aprendieron a trabajar en equipo, a comunicarse eficazmente y a coordinar sus esfuerzos para lograr objetivos comunes. Esta habilidad es crucial en la industria del software, donde el trabajo en equipo es la norma.
- *Comunicación y colaboración.* Este proyecto implicó que los estudiantes asumieran distintos roles como desarrolladores, diseñadores, arquitectos, testers, bases de datos y más. Los estudiantes tuvieron que comunicarse y colaborar entre si y con sus docentes y asesores, lo que reflejó la realidad laboral en la industria.
- *Resolución de problemas complejos.* El proyecto de software grande presentó desafíos técnicos y de diseño complejos. Los estudiantes abordaron problemas en etapas tempranas del proyecto y tomaron decisiones arquitectónicas importantes. Esto mejoró sus habilidades de resolución de problemas y toma de decisiones.
- *Planificación y gestión.* El proyecto requirió una planificación cuidadosa y una gestión efectiva del tiempo y los recursos. Los estudiantes adquirieron habilidades en la estimación de tiempos, la asignación de tareas y la administración de recursos, lo que es fundamental para la gestión de proyectos en la industria.
- *Comprensión de arquitectura.* Al trabajar en este proyectos grande, los estudiantes tuvieron la oportunidad de experimentar de primera mano cómo se diseñan y construyen arquitecturas de software escalables y robustas. Pudieron ver cómo se organizan los componentes, cómo se gestionan las dependencias y cómo se garantiza la modularidad.
- *Exposición a tecnologías actuales.* El proyecto implicó el uso de una variedad de tecnologías y herramientas actuales, como Java EE, Spring Boot, Postman, Docker, etc. Los estudiantes se familiarizaron con las herramientas de desarrollo y las prácticas modernas utilizadas en la industria.
- *Preparación para la industria.* Al enfrentar desafíos y situaciones similares a las que se encontrarán en sus futuros trabajos, los estudiantes se prepararon para la transición a la industria del software después de la graduación.
- *Autonomía y autogestión.* Trabajar en el proyectos Open Market requirió una cierta cantidad de autonomía y autogestión. Los estudiantes aprendieron a establecer metas,

seguir plazos y buscar soluciones por sí mismos, lo que fomenta su desarrollo profesional y su confianza en sus habilidades.

- *Portafolio Personal.* Este tipo de proyectos grandes y complejos pueden convertirse en valiosas adiciones a los portafolios personales de los estudiantes, lo que puede ayudarles a destacarse cuando busquen empleo después de la graduación.
- *Diversidad cultural y perspectivas.* La experiencia de desarrollo global que se realizó en la parte final del curso, entre estudiantes de la UNLP y la Unicauca, tuvo impactos adicionales. Trabajar con estudiantes de diferentes culturas y trasfondos académicos brindó a los estudiantes la oportunidad de obtener una comprensión más profunda de la diversidad cultural y las diferentes formas de abordar problemas. Los estudiantes aprendieron a colaborar en equipos distribuidos geográficamente y a través de husos horarios diferentes. Esto mejoró sus habilidades de comunicación y colaboración en un entorno global, lo cual es cada vez más relevante en el mundo laboral actual. Sin embargo, el poco tiempo dificultó la colaboración. De cinco equipos, solamente uno logro una colaboración adecuada y producir una solución software integrada y funcional.

Gracias a la aplicación del patrón de formación Cases-based training se logró coordinar tres charlas con los arquitectos de software invitados de la industria. Los alumnos pudieron apreciar casos reales de la industria. Estas charlas tuvo varios impactos positivos. Algunos fueron:

- *Vinculación con la industria.* Las conferencias de invitados de la industria brindaron a los estudiantes la oportunidad de conectarse directamente con profesionales activos y obtener una comprensión realista de cómo se aplican los conceptos teóricos de AS en el mundo laboral.
- *Aplicación práctica.* Escuchar casos de diseño de software de la vida real les permitió a los estudiantes ver cómo los conceptos de AS se traducen en soluciones concretas y aplicables en situaciones del mundo real.
- *Contexto empresarial.* Los conferencistas de la industria compartieron no solo los aspectos técnicos del diseño de software, sino también los desafíos comerciales y las consideraciones estratégicas que influyen en las decisiones de arquitectura. Esto ayudó a los estudiantes a comprender cómo el diseño de software se integra en el contexto empresarial más amplio.
- *Variedad de perspectivas.* Los conferencistas invitados provenían de diferentes empresas y sectores, lo que expuso a los estudiantes a una variedad de enfoques. Esto amplía su perspectiva sobre las diferentes formas en que se pueden abordar problemas de diseño.
- *Historias de éxito y fracaso.* Los invitados de la industria compartieron historias de proyectos exitosos pero también de desafíos y fracasos en el diseño de software. Estas

historias proporcionaron lecciones valiosas sobre lo que funciona y lo que no, lo que puede ayudar a los estudiantes a tomar decisiones más informadas en sus futuros roles.

- *Interacción directa.* En las sesiones de preguntas y respuestas, los estudiantes pudieron hacer preguntas directas a los profesionales de la industria y obtuvieron respuestas basadas en la experiencia práctica y real.

Respecto a las competencias logradas al final del curso, se preguntó a cada uno de los estudiantes: *Evalúe en qué grado este curso le ayudó a conseguir cada una de las siguientes competencias requeridas por la industria de software. En todas las preguntas se utilizará la siguiente escala; 1. No ayuda, 2. Ayuda poco, 3. Neutral, 4. Ayuda, 5. Ayuda mucho* (La descripción de las competencias evaluadas se pueden apreciar en el Apéndice C de las cuales se tomaron las de la categoría obligatorias). Las repuestas se pueden apreciar en la Figura **5-3**. Podemos ver que algunas competencias como C05 (*Evalúa de forma independiente una arquitectura de software para determinar la satisfacción de los requisitos funcionales y no funcionales*), C08 (*Realiza de forma imparcial un análisis de compensación para evaluar las arquitecturas*) y C011 (*Mantiene fácilmente los sistemas existentes y su arquitectura para lograr la evolución de los sistemas de software*) los estudiantes manifiestan que el curso "ayudó mucho" a desarrollarlas en una gran proporción. Por otro lado, hay competencias que no fueron desarrolladas como la C22 (*revisa periódicamente el código fuente escrito por el equipo de desarrollo*) pues un gran porcentaje afirman que el curso "ayudó poco".

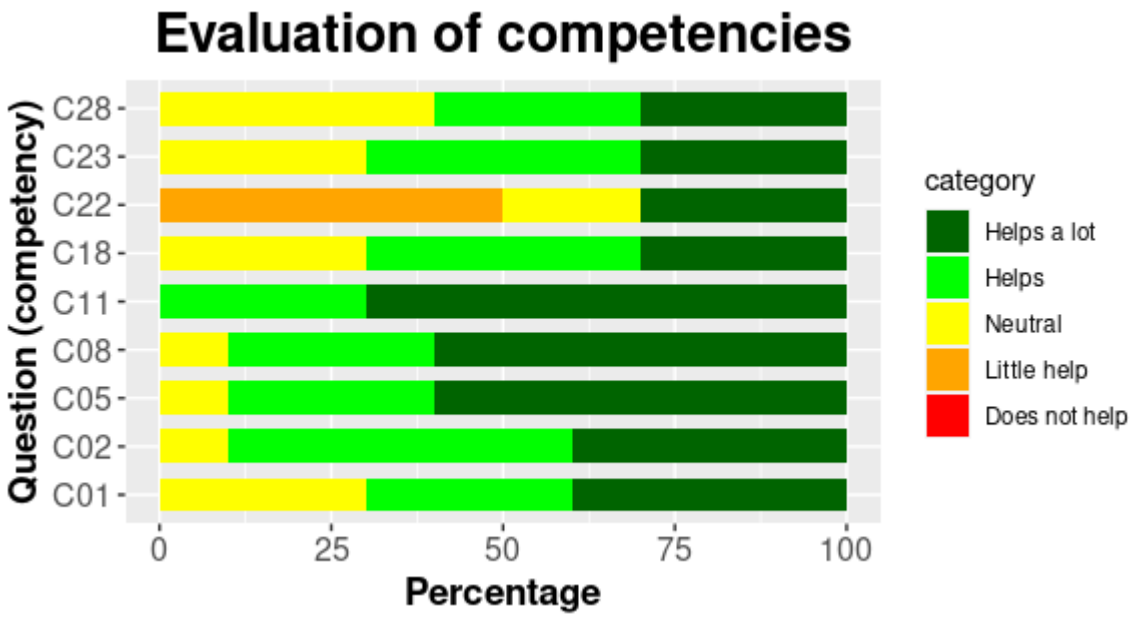

Figura 5-3.: Respuestas a la pregunta: Evalúe en qué grado cree que el curso de Patrones de Arquitectura, ayudó a conseguir cada una de las siguientes competencias.

En cuanto a las dificultades encontradas al aplicar patrones de formación en la UNPL, la

principal reside en el esfuerzo adicional y el tiempo requerido por parte del docente. Cada uno de estos patrones conlleva una fase de planificación y ejecución específica. Por ejemplo, coordinar charlas con invitados de la industria demandó que el docente llevara a cabo tareas como la búsqueda de los invitados, la realización de reuniones previas, y la coordinación de sus disponibilidades, entre otros aspectos. La ejecución de un proyecto de gran envergadura implicó esfuerzos considerables en la redacción, revisión, mejora y ajuste del proyecto. Además, realizar el seguimiento y monitoreo de los proyectos de los estudiantes representó un desafío significativo. Afortunadamente, en este caso particular, el docente pudo contar con el apoyo de dos colaboradores, quienes le brindaron asistencia en cuestiones relacionadas con las tecnologías Java, despliegue de aplicaciones y seguimiento a los estudiantes.

RQ2: ¿Cómo fue evaluada la utilidad percibida y la facilidad de uso de la Guía por el docente en este estudio de caso?

Respecto a la utilidad percibida por el docente su respuesta fue: "*la guía fue de mucha utilidad porque permitió hacer mejoras al curso. Se ajustaron algunos temas de AS, se eligió los recursos necesarios, y finalmente, se eligieron los patrones de formación para acercar el curso a la industria. La planificación realizada se cumplió durante la ejecución. La guía proporciona información precisa y relevante sobre el diseño de cursos de Arquitectura de Software. Los conceptos y las pautas presentados son aplicables directamente a mi área de estudio*".

Respecto a la facilidad de uso, el docente expresó: "*La facilidad de uso de la guía es notable y ha sido una experiencia positiva en general. Esto se debe a las siguientes razones. La guía presenta una navegación clara y una estructura bien organizada. Puedo moverme fácilmente entre secciones y encontrar la información que necesito sin dificultad. El lenguaje utilizado en la guía es claro y accesible. La terminología técnica se explica de manera comprensible, lo que facilita la comprensión, incluso para aquellos que pueden no estar familiarizados con todos los conceptos. La guía utiliza ejemplos ilustrativos que ayudan a aclarar los conceptos, sobretodo de cada patrón de formación. Esto facilita la comprensión y la aplicación práctica de las pautas y recomendaciones. La posibilidad de usar plantillas en cada paso, agrega valor a la guía y simplifica su utilización. Uno de los resultados obtenidos de la guía fue la planificación del curso que resultó muy útil a la hora de hacer el seguimiento del curso. En resumen, la facilidad de uso de la guía es un punto fuerte, ya que su diseño y estructura hacen que sea fácil de navegar y comprender, lo que mejora significativamente mi experiencia al utilizarla*".

RQ3: ¿Cómo fue la satisfacción del curso por parte de los estudiantes?

A través de encuestas aplicadas se capturaron comentarios valiosos sobre el curso. A continuación se describen algunos:

- "Una gran cursada, una linda experiencia y agradecido por mis compañeros, como también con los profesores y con los que vinieron a dar charlas. Pocas veces se dio que vengan gente de otros ámbitos a compartir sus experiencias".
- "En general me gustó la metodología del curso, formar grupos pequeños con un proyecto a desarrollar. Me gustó la implementación de los pipelines y que tengamos que usar docker, siento que me fueron muy útil para desarollos reales".
- "La primera vez que una materia de la facultad trae profesionales de la industria a dar una charla".
- "Es interesante conocer como los problemas que ocurren en la industria son atacados desde el punto de vista del diseño de la arquitectura. Estas charlas permiten conocer estás experiencias, que de otra manera la única forma es enfrentando la problemática en una situación real "
- "Me pareció muy útil y práctica la incorporación de ejemplos reales, paso a paso, acerca de cómo se lleva a cabo un proceso de desarrollo arquitectónico. Permite ver la importancia y el rol de la arquitectura en el software"
- "Fue muy bueno tener la oportunidad de escuchar a alguien de la industria que se encuentra desarrollando un proyecto, ya que aunque nuestros profesores son personas capacitadas en su materia y con experiencia, pocas veces hablan a cerca de esas experiencias y como se enfrentaron a esos problemas utilizando lo visto en clase o al menos no de forma explícita, por lo que en ocasiones algunos temas se sienten como relleno y algo poco aplicable en el mundo real cuando es todo lo contrario."
- "Considero que las charlas son una valiosa oportunidad para intercambiar ideas, compartir conocimientos y fortalecer la conexión entre las personas. No obstante, es importante reconocer que la falta de claridad en los conceptos puede dificultar la comprensión y el aprovechamiento pleno de las charlas. Por ello, considero que es fundamental tener una base sólida de conocimientos y conceptos claros antes de participar en una charla."

Respecto a la satisfacción en general del curso, se hicieron varias preguntas a través de una encuesta a los estudiantes. Se usó una escala Likert, siendo 1 Nada satisfecho y 5 Muy satisfecho. Respecto a la pregunta, *¿En general, qué tan satisfecho quedó con lo aprendido en el curso?* Las respuestas en un diagrama de cajas, se pueden apreciar en la Figura **5-4**. La gráfica de cajas muestra representa la distribución de la satisfacción de los estudiantes que participaron en el curso de Patrones de Arquitectura de Software de la UNPL. La línea horizontal en la mitad de la caja representa la mediana de los niveles de satisfacción. En este caso, la mediana es 4. Podemos decir que la mayoría de las respuestas se encuentran en un rango de puntuaciones entre 3 y 5, con una mediana en 4. Esto sugiere que la satisfacción general tiende a ser alta, ya que la mayoría de las respuestas están en el extremo superior de la escala de 1 a 5.

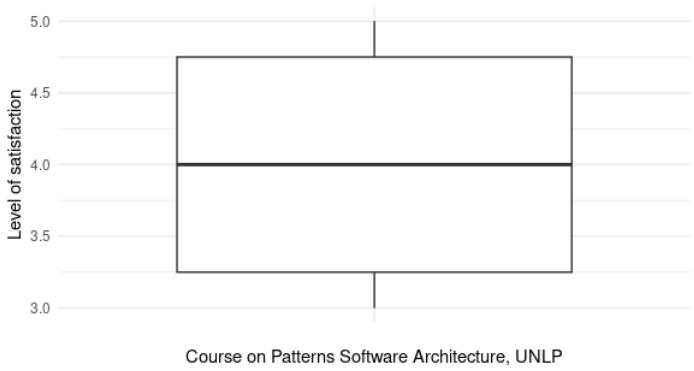

Figura 5-4.: Gráfico de cajas de la pregunta ¿En general, qué tan satisfecho quedó con lo aprendido en el curso?

También se realizaron preguntas para conocer la satisfacción de ciertas partes específicas del curso. La Figura **5-5** muestra los resultados de la pregunta *¿Qué tan útiles le parecen las habilidades, conocimientos que desarrolló a lo largo del curso en las áreas de Arquitectura de Software para afrontar proyectos del mundo real?* Podemos decir que la mayoría de las respuestas se encuentran en un rango de puntuaciones entre 4 y 5, con una mediana en 5. Esto sugiere que la satisfacción general tiende a ser muy alta. Hay un dato atípico que es 2.

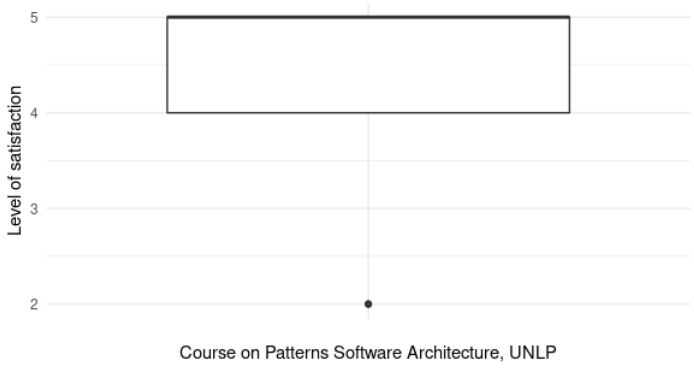

Figura 5-5.: Respuestas a ¿Qué tan útiles le parecen las habilidades, conocimientos que desarrolló a lo largo del curso en las áreas de Arquitectura de Software para afrontar proyectos del mundo real?

La Figura **5-6** muestra los resultados de la pregunta *¿Qué tan satisfecho quedó con las charlas de invitados provenientes de la industria de software?* Podemos decir que la mayoría de las respuestas se encuentran en un rango de puntuaciones entre 4 y 5, con una mediana en 4.5. Esto sugiere que la satisfacción general tiende a ser muy alta.

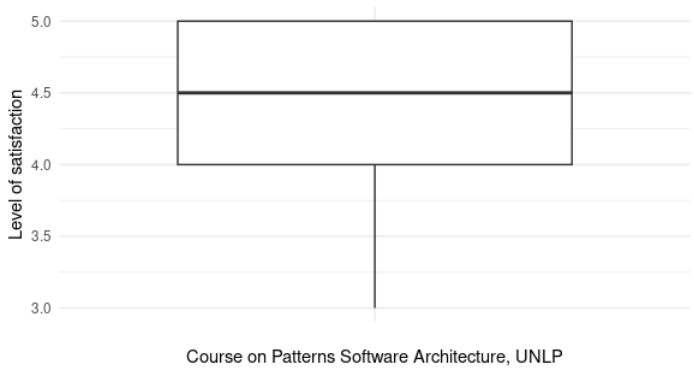

Figura 5-6.: Respuestas a ¿Qué tan satisfecho quedó con las charlas de invitados provenientes de la industria de software?

La Figura **5-7** muestra los resultados de la pregunta *¿Qué tan satisfecho quedó con la experiencia de desarrollo global que se hizo al final del curso mezclando estudiantes de Unicauca y de la UNLP?* Podemos decir que la mayoría de las respuestas se encuentran en un rango de puntuaciones entre 1 y 5, con una mediana en 3. Esto sugiere que la satisfacción general está dispersa, no hay una tendencia positiva. Por lo tanto, no es posible determinar si generó satisfacción o no a los estudiantes.

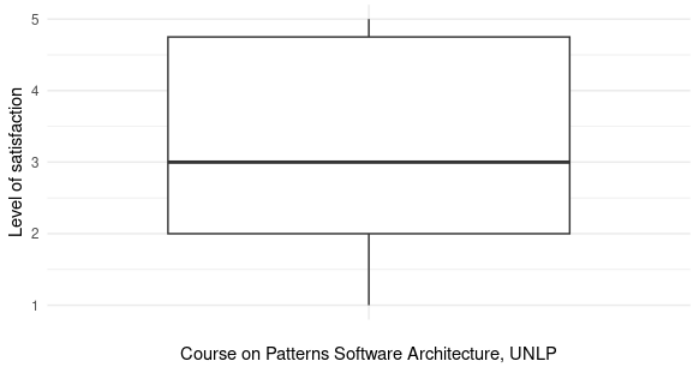

Figura 5-7.: Respuestas a ¿Qué tan satisfecho quedó con la experiencia de desarrollo global que se hizo al final del curso mezclando estudiantes de Unicauca y de la UNLP?

Finalmente, la Figura **5-8** muestra los resultados de la pregunta *¿Qué tan satisfecho quedó con el proyecto de clase Open Market (o mercado libre) que se trabajó a lo largo del curso para aplicar los conceptos aprendidos de Arquitectura de Software?* Podemos decir que la mayoría de las respuestas se encuentran en un rango de puntuaciones entre 3 y 5, con una mediana en 4. Hay un dato atípico que es 2. Esto sugiere que la satisfacción general tiende a ser alta.

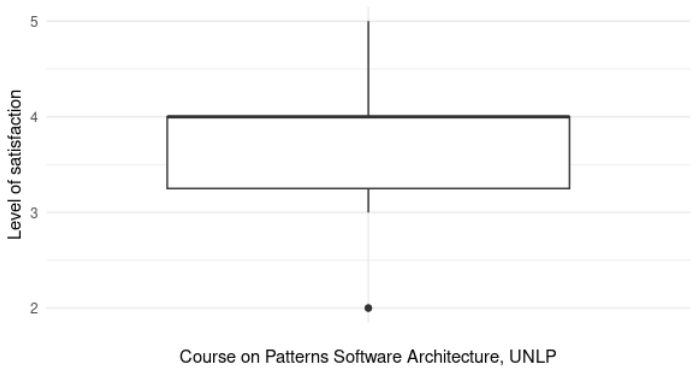

Figura 5-8.: Respuestas a ¿Qué tan satisfecho quedó con el proyecto de clase Open Market?

5.5.5. Discusión del estudio de caso

Utilizar un proyecto de software grande en equipos para enseñar arquitectura de software brindó a los estudiantes de la UNPL una oportunidad única para desarrollar habilidades prácticas, trabajar en equipo, enfrentar desafíos reales y prepararse para carreras en la industria del software.

La estrategia de traer a las clases conferencistas de la industria para compartir casos de diseño de software de la vida real, enriqueció la experiencia educativa de los estudiantes al proporcionarles conocimientos directos, perspectivas prácticas y conexiones valiosas con el mundo laboral.

La experiencia de desarrollo de proyectos de software global agregó una dimensión adicional de diversidad cultural, comunicación global y colaboración interdisciplinaria, preparando a los estudiantes para un mundo laboral cada vez más conectado y globalizado. Sin embargo, esta actividad para futuros cursos debe ser mejorada en varios aspectos, tanto en el manejo de los tiempos como en realizar actividades previas para permitir que los estudiantes se conozcan. Además, en el momento no se cuenta con un patrón de formación en desarrollo de software global.

El curso permitió evaluar las competencias de carácter obligatorio que desarrollaron los estudiantes. Las respuestas son variadas, algunas competencias fueron desarrolladas ampliamente, otras poco y otras de forma neutral.

En general la satisfacción del docente con el uso y aplicación de SAGITA arrojó resultados positivos. SAGITA ayudó al docente a planificar su curso y aplicar estrategias que acercaron el curso con la realidad de la industria de software. Además examinamos la satisfacción de los estudiantes evaluando a través de encuestas el impacto de las estrategias propuestas en su formación. Los resultados en general fueron satisfactorios. A partir de este estudio de

caso se pudo afinar la redacción de los patrones de formación involucrados, especialmente los campos de la solución, el ejemplo y las fuerzas. De otro lado, se corroboró la lista de competencias que se alcanzan con los dos patrones de formación.

5.6. Experimento para evaluar la efectividad de la Guía SAGITA y el catálogo de patrones de formación

El propósito de este experimento es analizar la efectividad de la Guía de Diseño de Cursos de arquitectura de software SAGITA y sus patrones de formación. El proceso de experimentación se ha divido en las siguientes actividades principales (ver Figura **5-9**) según los lineamientos de Wohlin et al., [126]. El primer paso es la *definición del del alcance del experimento (scoping)* en términos de problema, objetivo y metas. Después viene la *planificación*, en la que se determina el diseño del experimento, se considera la instrumentación y se evalúan las amenazas para el experimento. La *operación* del experimento se deriva del diseño. En la actividad *operativa* se recogen mediciones que luego se analizan y evalúan en *análisis e interpretación*. Por último, se presentan y empaquetan los resultados en *presentación y empaquetado*. A continuación se explican cada una de estas actividades. A continuación se explican cada una de estas actividades.

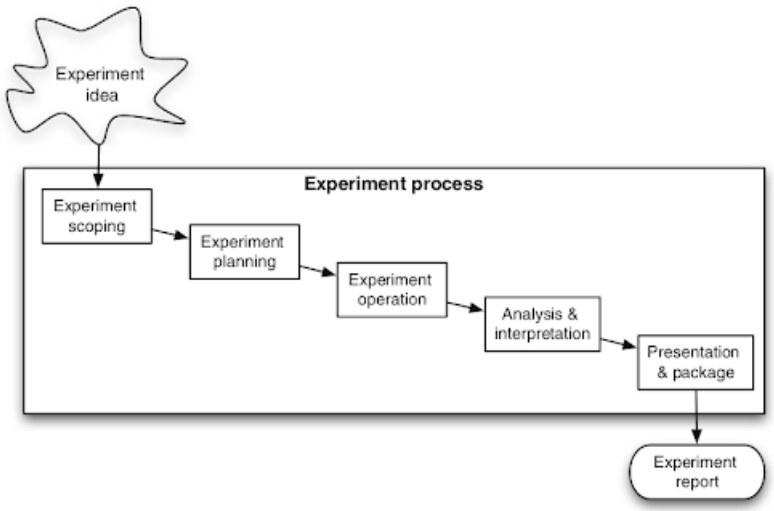

Figura 5-9.: Proceso del experimento que evalúa la efectividad de la Guía

5.6.1. Definición del alcance del experimento

La definición del alcance del experimento lo hicimos siguiendo el siguiente framework:

1. Objeto de estudio (¿qué se estudia?)
2. Propósito (¿cuál es la intención?)
3. Enfoque de calidad (¿qué efecto se estudia?)
4. Perspectiva (¿la opinión de quién?)
5. Contexto (¿dónde se realiza el estudio?)

De esta forma, la definición del alcance queda así. (*objeto*:) Analizar la Guía de diseño de cursos de AS y sus patrones de formación (*propósito*:) con el propósito de evaluar (enfoque calidad:) la efectividad de la Guía para que los estudiantes logren desarrollar las competencias mínimas que la industria requiere (*perspectiva*:) desde el punto de vista del investigador (*contexto*:) en el contexto del diseño de cursos de AS a nivel de pregrado.

5.6.2. Planificación

La planificación prepara sobre cómo será conducido el experimento. La planificación requiere siete pasos:

1. Selección del contexto.
2. Formulación de hipótesis.
3. Selección de variables.
4. Selección de sujetos.
5. Diseño del experimento
6. Instrumentación.
7. Evaluación de la validez.

Paso1: Selección del contexto

La caracterización del contexto lo hicimos acorde a cuatro dimensiones:

- El experimento debe llevarse a cabo de manera presencial.
- Participaron docentes universitarios que trabajan temáticas de ingeniería de software, específicamente de arquitectura de software.
- Durante el experimento se abordó el diseño de un curso de arquitectura ficticio.
- Se trató de un experimento específico donde los sujetos fueron divididos en dos grupos.

El primer grupo fue el experimental y trabajó con la Guía llamada SAGITA [4]. El segundo grupo es el de control y trabajó con una Guía basada en la revisión de la literatura. Las dos guías contienen en general los mismos pasos, pero SAGITA contiene patrones de formación listos para ser usados por el docente, en cambios la segunda guía contiene estrategias de formación y sus referencias bibliográficas. La idea de la segunda guía es simular el esfuerzo que hace un docente preparando un curso de AS a partir de la revisión bibliográfica.

Paso2: Formulación de la hipótesis

1. Hipótesis nula, H_0, la *eficiencia* en el diseño del curso con SAGITA y el catálogo de patrones es *igual o menor* que la eficiencia en el diseño del curso con la información y estrategias más recomendadas por la revisión de la literatura (RL). Esto lo expresamos como $H_0 : \mu_{SAGITA+patterns} = \mu_{SAGITA+RL}$.
2. Hipótesis alternativa, H_1, la *eficiencia* en el diseño del curso con la SAGITA y el catálogo de patrones es *mayor* que la eficiencia en el diseño del curso con la información y estrategias más recomendadas por la revisión de la literatura. Esto lo expresamos como $H_1 : \mu_{SAGITA+patterns} > \mu_{SAGITA+RL}$.

Paso3: Selección de variables

La variable independiente son las estrategias para el diseño del curso con la guía SAGITA. La variable dependiente es *la eficiencia* de la guía SAGITA.

Para calcular la eficiencia usamos el promedio de los *tiempos empleados en aplicar las guías* como uno de los indicadores principales. Es decir, comparamos los tiempos promedio de los docentes al diseñar el curso de AS utilizando SAGITA en contraste con los que utilizan la guía con la revisión de la literatura. Sin embargo, los tiempos por sí solos pueden no proporcionan una imagen completa de la eficiencia de la guía. Para evaluar la eficiencia de manera más precisa, se necesitaría considerar otros factores además de los "tiempos empleados en aplicar la guía", como los recursos utilizados, la calidad del resultado, los costos asociados y la satisfacción de los docentes. En este caso vamos a medir la satisfacción de los docentes utilizando las dos guías y la calidad del resultado mediante las competencias de AS que se lograron abarcar.

Para conocer la satisfacción de los docentes usando las dos guías, utilizamos la siguiente encuesta con preguntas en la escala Likert de 1 a 5 (1: Nada, 5: Mucho):

1. ¿En términos generales, qué tan satisfecho(a) está con la guía de creación de cursos de

[4]Las dos guías están disponibles en formato PDF en la página Artefactos Tesis

arquitectura de software?

2. ¿Cuál es su nivel de satisfacción con la claridad y facilidad de uso de la guía?
3. ¿Cuál es el nivel de utilidad de la guía en tu trabajo como docente de arquitectura de software.
4. ¿Cómo calificaría la calidad de la información proporcionada en la guía?
5. ¿En qué medida considera que la guía cumple con sus expectativas como recurso de apoyo para la creación de cursos de arquitectura de software?
6. ¿Cuánto te ha ayudado la guía a organizar y estructurar los contenidos de tus cursos de arquitectura de software?
7. ¿La guía le ha ayudado a diseñar estrategias más efectivas, por medio de los patrones de formación, para desarrollar competencias en los estudiantes alineadas a las necesidades de la industria?
8. ¿Recomendaría la guía de creación de cursos de arquitectura de software a otros docentes?

De igual manera, para conocer las competencias que se lograron abordar diseñando el curso, utilizamos una segunda encuesta con la pregunta: Evalúe en qué grado cree que el curso de arquitectura de software que acaba de proponer, ayuda a conseguir cada una de las siguientes competencias. En todas las preguntas se utilizará la siguiente escala; 1. No ayuda, 2. Ayuda poco, 3. Neutral, 4. Ayuda, 5. Ayuda mucho. Las competencias a evaluar fueron tomadas del Apéndice C y elegimos las pertenecientes a la categoría obligatorias.

Paso 4: Selección de sujetos

La selección de los participantes, también conocida como muestra de una población, se centró en docentes universitarios que orienten cursos relacionados con arquitectura de software para llevar a cabo este experimento. La elección de la población se vio condicionada por la necesidad de presencialidad, lo que estableció un contexto geográfico específico conformado por profesores de Popayán y Cali.

En este experimento usamos la técnica no probabilística denominada *muestreo de conveniencia* que selecciona como sujetos a las personas más cercanas y convenientes. El tamaño de la muestra fue de 10 profesores.

Paso 5: Diseño del experimento

Para sacar conclusiones significativas de un experimento, aplicamos el método de análisis estadístico T-student [5] a los datos recogidos.

El tipo de diseño elegido fue *un factor con dos tratamientos*. El factor es la guía SAGITA para diseñar cursos y tenemos dos tratamientos del factor: la guía SAGITA y la guía con el estado del arte que recomienda estrategias. La variable dependiente es la eficiencia. La asignación de sujetos a los tratamientos se hizo de forma aleatoria tal como lo muestra la Tabla **5-2**.

Sujetos	Tratamiento 1 (Guía SAGITA)	Tratamiento 2 (Guía con el estado del arte)
1	X	
2		X
3		X
4	X	
5		X
6	X	
7	X	
8		X

Tabla 5-2.: Asignación de sujetos a los tratamientos para un diseño aleatorio.

Paso 6: Instrumentación

Los instrumentos de un experimento son de tres tipos: objetos, directrices e instrumentos de medida.

Los *objetos* de este experimento fueron los documentos con el diseño de los cursos de AS que elaboraron los docentes a partir de las dos guías. Se necesitan *directrices* para guiar a los participantes en el experimento. Esta directrices fueron dadas a los docentes en un documento a través de sus correos electrónicos. Las *mediciones* en un experimento se realizan mediante la recogida de datos. En este experimento los datos fueron recogidos mediante formularios, plantillas a diligenciar y observación.

Procuramos que los resultados del experimento sean los mismos independientemente de cómo se instrumente el experimento. Si la instrumentación afecta al resultado del experimento, los

[5] La prueba t-student se aplica cuando la población estudiada sigue una distribución normal y el tamaño muestral es demasiado pequeño

resultados no son válidos.

Paso 7: Evaluación de la validez

Para sacar conclusiones hay cuatro pasos, en cada uno de los cuales hay un tipo de amenaza para la validez de los resultados: validez de la conclusión, validez interna, validez del constructo y validez externa. El Apéndice G muestra las amenazas que tuvimos en cuenta para la validez del experimento: validez de la conclusión, validez interna, validez del constructo y validez externa.

5.6.3. Operación

En la fase operativa del experimento, los tratamientos fueron aplicados a los sujetos. La fase operativa del experimento contó de tres pasos: la *preparación*, en la que elegimos a los sujetos y preparamos los formularios; la *ejecución*, en la que los sujetos realizaron sus tareas según los distintos tratamientos y se recogieron los datos; y la *validación*, en la que se validaron los datos recogidos.

En la preparación seleccionamos e informamos a los participantes mediante correo electrónico, y preparamos el material como formularios, guías y anexos.

Durante la ejecución el experimento se llevó en una sala en la que los participantes estuvieron reunidos para diseñar un curso de AS ficticio siguiendo los cinco pasos que plantea SAGITA (ver Figura **5-10**). La Tabla **5-3** muestra los docentes que participaron del experimento y los años de experiencia formando en temas de AS. La recolección de datos ser hizo mediante una plantilla diligenciada y un formulario. La plantilla tenía el resultado del diseño del curso de AS y el formulario permitió evaluar la satisfacción del uso de la guía y recolectar datos de las competencias que se lograron abarcar.

Figura 5-10.: Ejecución del experimento con 10 docentes invitados, divididos aleatoriamente en dos grupos: Experimental y Control

Tabla 5-3.: Docentes que participaron en el experimento diseñando cursos de AS.

No.	Universidad	Experiencia (años)
1	Universidad del Cauca	4
2	Universidad del Cauca	10
3	Institución Universitaria Colegio Mayor del Cauca	4
4	Unicomfacauca	4
5	Universidad del Cauca	13
6	Universidad Autónoma de Occidente sede Cali	3
7	Institución Universitaria Colegio Mayor del Cauca	2
8	Universidad Nacional Abierta y a Distancia	1
9	Universidad del Cauca	8
10	Fundación Universitaria de Popayán	2

En la validación de los datos comprobamos que los datos fueran razonables y que se hayan recogido correctamente. Se trató de aspectos tales como si los participantes comprendido los formularios, plantillas y, por tanto, los diligenciaron correctamente.

5.6.4. Análisis e interpretación

Los datos experimentales de la fase de operación sirvieron para el análisis y la interpretación. La interpretación cuantitativa la realizamos tres pasos: (1) estadísticas descriptivas, (2) reducción el conjunto de datos, (3) comprobación de hipótesis. A continuación se muestran cada uno de estos tres pasos.

La Tabla **5-4** muestra los tiempos empleados por los docentes diseñando los cursos de AS. La columna grupo indica si el docente pertenece al grupo experimental o al de control. El grupo experimental utilizó los pasos de la Guía SAGITA y el grupo de control la Guía con el estado del arte. La columna Tiempo es el tiempo total en minutos que cada docente empleó en ejecutar la guía.

Tabla 5-4.: Tiempos empleados por los 10 docentes diseñando un curso de AS

No	Grupo	Tiempo (minutos)
1	Experimental	122
2	Experimental	111
3	Experimental	114
4	Experimental	122
5	Experimental	123
6	Control	155
7	Control	132
8	Control	148
9	Control	146
10	Control	67

Aplicando estadística descriptiva a los datos de la Tabla **5-4** se pudo apreciar cómo se distribuye el conjunto de datos. Utilizamos lenguaje R para este análisis, el Apéndice H muestra el Script utilizado. Primero se realizó un gráfico de cajas (ver Figura **5-11**) y un gráfico QQ (ver Figura **5-12**) para verificar si hay datos atípicos en la muestra. En teoría, en un gráfico de cajas no deben aparecer datos arriba de los limites superiores ni inferiores. En un gráfico QQ se dibuja una linea y se puede verifica que los datos estén cerca a la recta. Se pudo apreciar que aparece un dato atípico que corresponde a la muestra número 10 con un tiempo de 67 minutos. Revisando la plantilla de este docente pudimos verificar que el docente utilizó durante la prueba la herramienta IA Open Chat GPT. Esto lo llevo a utilizar respuestas automáticas y a disminuir los tiempos. Por lo tanto, este dato fue descartado. El grupo de control finalmente quedó con 4 muestras.

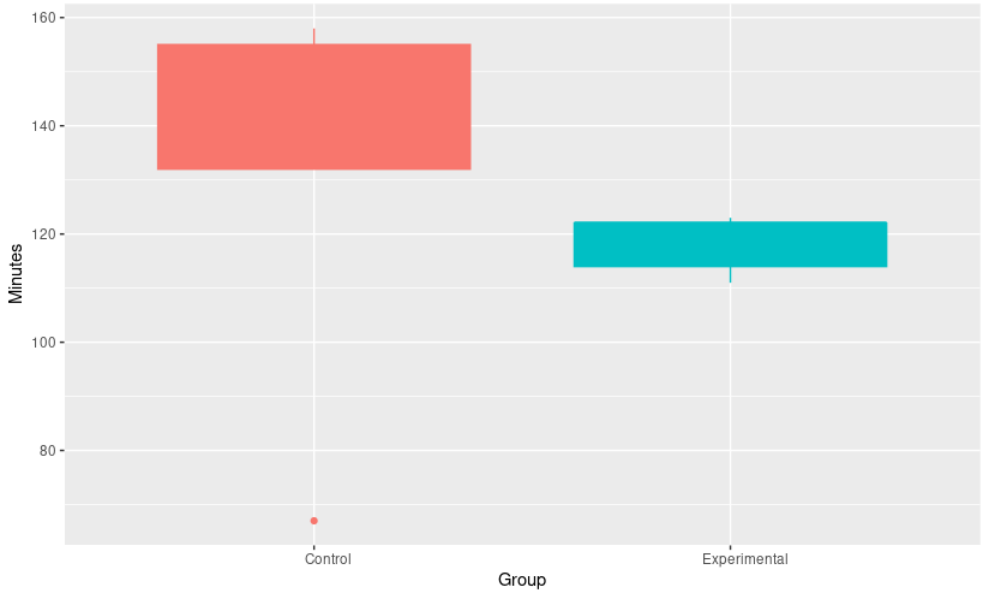

Figura 5-11.: Gráfico de cajas para verificar datos atípicos

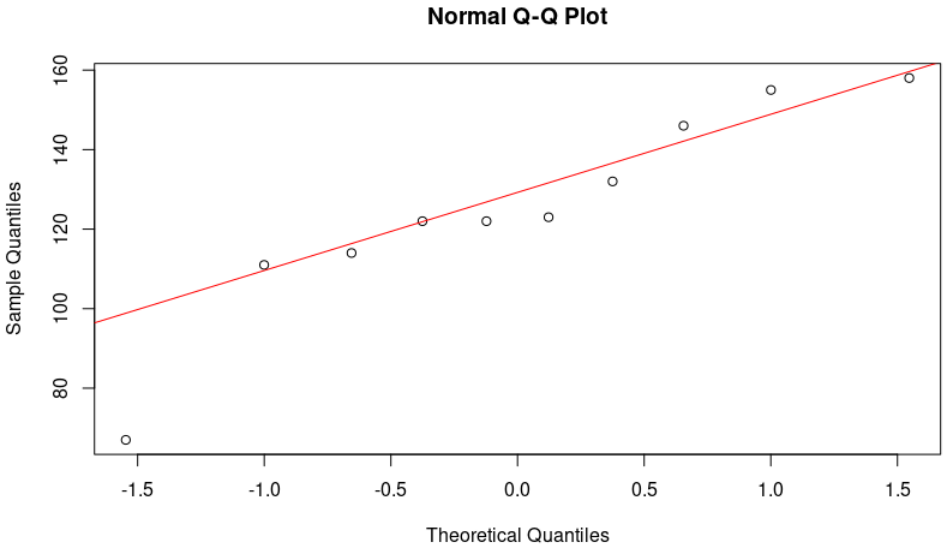

Figura 5-12.: Gráfico QQ para verificar datos atípicos

En seguida, mediante una gráfica de densidad (ver Figura **5-13**), se verificó que los datos siguen una distribución normal [6] y no tuvieran otros datos atípicos.

[6]En una distribución normal, los datos se distribuyen de manera simétrica alrededor de un valor central (media), donde la mayor concentración de datos se encuentra cerca de la media y disminuye gradualmente hacia los extremos. La distribución normal se define completamente por su media y su desviación estándar.

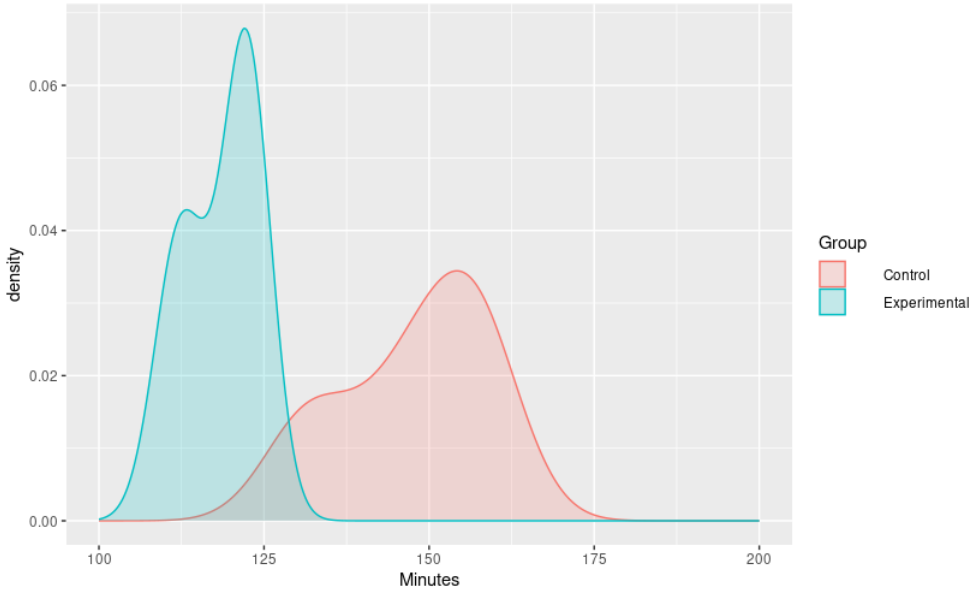

Figura 5-13.: Gráfico de densidad de los datos del experimento

En seguida, aplicamos un test Shapiro-Wilk para tener la certeza si los datos tienen una distribución normal. En este caso, interesa el p-value, si p-value >0.05 no se rechaza la hipótesis nula. La hipótesis nula dice que la muestra sigue una distribución normal. En este caso p-value dio un valor de *0.2048* que es mayor a 0.05, por lo tanto, la muestra sigue una distribución normal.

Además, obtuvimos los promedios de las muestras que se pueden apreciar en la Tabla **5-5**. Vemos que el promedio del grupo experimental es 118 minutos y del grupo de control es de 148 minutos, es decir, hay una diferencia significativa.

Tabla 5-5.: Promedios de la muestras en el experimento

Grupo	Total muestras	Promedio (minutos)
Experimental	5	118
Control	4	148

También se ejecutó una prueba F para comparar las varianzas de las dos muestras de poblaciones normales. Este test arrojó un p-valor de *0.1805* que es mayor a 0.05, por lo tanto, no se rechaza la hipótesis nula, eso implica que las varianzas entre los dos grupos es la misma. Esta información sirve para ejecutar la prueba t-student.

Finalmente, se realizó la *prueba de hipótesis t-student* para comprobar si es posible rechazar la hipótesis nula, H_0 , a partir de la muestra de una distribución estadística. Aplicamos la prueba t-student analizando la variable minutos diferenciándola por el grupo (control y experimental) con varianzas iguales, en lenguaje R lo expresamos de la siguiente manera:

t.test(Minutes Group, data=historial, var.equal = T)

El resultado arrojado de t-student fue:

```
Two Sample t-test
data:  Minutes by Group
t = 5.0284, df = 7, p-value = 0.001516
Alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
15.54817 43.15183
sample estimates:
mean in group Control      mean in group Experimental
147.75                     118.40
```

La prueba t-student arrojó un *p-value=.001516* asociado a un t=5.0284, el cual es menor a 0.05, por lo tanto rechazamos al hipótesis nula. La hipótesis nula dice que la diferencia entre los promedios de los tiempos de las muestras es cero. Por lo tanto, llegamos a la conclusión que las muestras son distintas (aceptamos la hipótesis alternativa).

En conclusión, dado que el tiempo promedio del grupo experimental es menor que el tiempo promedio del grupo de control, por lo tanto, se pudo mostrar que la *eficiencia* en el diseño del curso con la guía SAGITA y el catálogo de patrones es *mayor* que la eficiencia en el diseño del curso con la información y estrategias más recomendadas por la literatura, $H_1 : \mu_{SAGITA} > \mu_{RL}$.

Para medir la satisfacción de los docentes utilizando las dos Guías, aplicamos una encuesta a los diez docentes. La encuesta contenía preguntas relacionadas con la satisfacción, la facilidad de uso y la utilidad percibida de las guías. La Figura **5-14**, Figura **5-15**, Figura **5-16** presentan los resultados donde se pueden comparar los datos de los dos grupos.

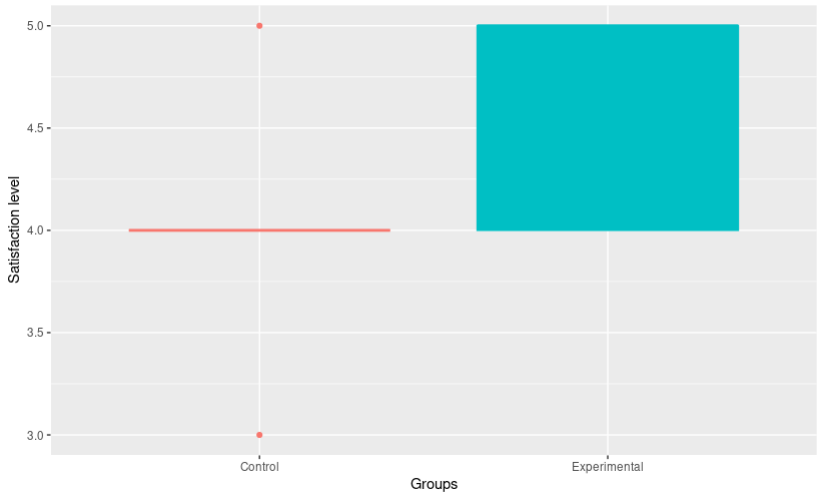

Figura 5-14.: Satisfacción de uso de las Guías para los grupos de control y experimental

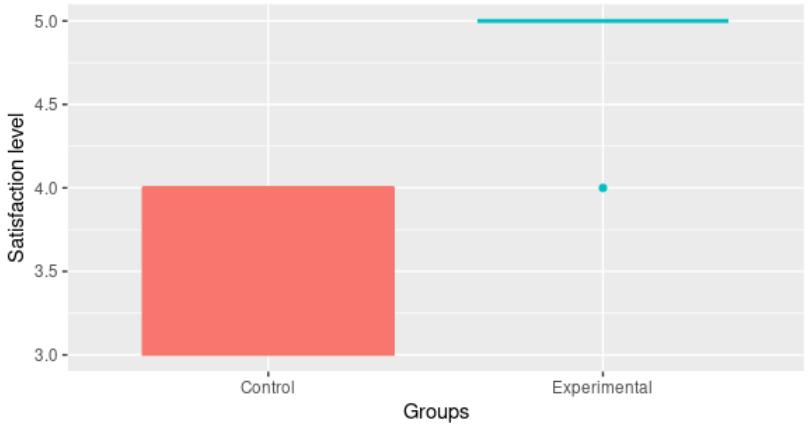

Figura 5-15.: Utilidad percibida de las Guías para los grupos de control y experimental

En general, se puede apreciar que las tres métricas son mayores en los docentes que usaron SAGITA. De esta forma hay indicios que se cumple la hipótesis alternativa.

Respecto a las competencias logradas con el diseño del curso, le preguntamos a cada uno de los docentes: *Evalúe en qué grado cree que el curso de arquitectura de software que acaba de proponer, ayuda a conseguir cada una de las siguientes competencias. En todas las preguntas se utilizará la siguiente escala; 1. No ayuda, 2. Ayuda poco, 3. Neutral, 4. Ayuda, 5. Ayuda mucho*. Las respuestas de cada una de las competencias evaluadas, en cada uno de los grupos de docentes (experimental y control) se puede apreciar en la Figura **5-17**.

En términos generales, se aprecia en la Figura **5-17** que en el grupo experimental hay más porcentajes de respuestas con "Ayuda mucho" como es el caso de las competencias C01, C02,

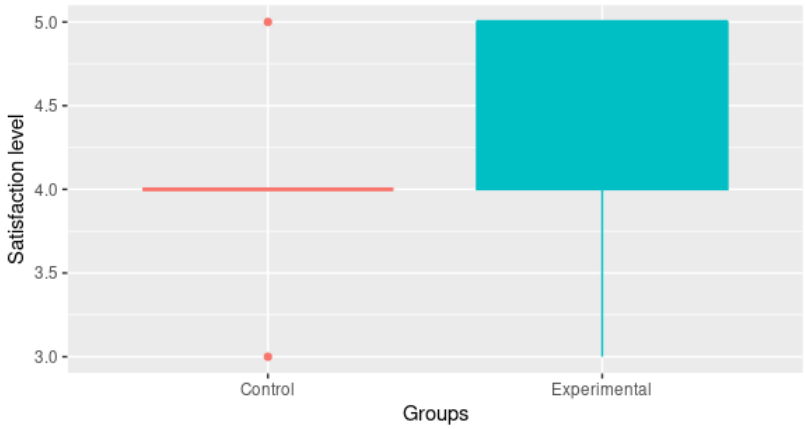

Figura 5-16.: Facilidad de uso de las Guías para los grupos de control y experimental

C05, C012, C018, C22 y C23. Sin embargo no hay una evidencia contundente que permita demostrar que con SAGITA se logró abarcar más competencias.

5.7. Validación con expertos del PLOP 2023

La conferencia "Pattern Languages of Programs (PLoP)" representa un espacio de encuentro y diálogo para autores y apasionados de los patrones, propiciando la reflexión y el aprendizaje en torno a conceptos que abarcan sobre patrones, diseño, desarrollo de software y el mundo de la construcción en general. Inicialmente centrada en patrones y lenguajes de programación vinculados al software, PLoP ha extendido su alcance para abarcar todo tipo de patrones relacionados con software. Además de las temáticas relacionadas con patrones y lenguajes de patrones, PLoP abre sus puertas a presentaciones y ensayos que versen sobre las ideas y legado de Christopher Alexander.

En este sentido, se envió un artículo a la conferencia PLoP 2023 titulado "Towards a Software Architecture Training Pattern Language" [89] con el fin de tener una retroalimentación y revisión de expertos en un proceso llamado pastoreo (shepherding). Gracias a este pastoreo, se tuvo la oportunidad de recibir retroalimentación valiosa de expertos en el campo de patrones. Los participantes y revisores son profesionales con experiencia en la materia, lo que te permitió mejorar el catálogo de patrones de formación y validar las ideas.

Durante un periodo de dos meses, entre julio y agosto, se entró en el proceso de pastoreo con el experto Waheedullah Sulaiman profesor de la Slovak University of Technology. Se envió cuatro rondas de pastoreo. En cada ronda se obtuvo mejoras significativas tanto de los patrones de formación como del lenguaje de patrones que se propuso. Este pastoreo permitió identificar que además de los patrones de formación se necesitaba un lenguaje de patrones.

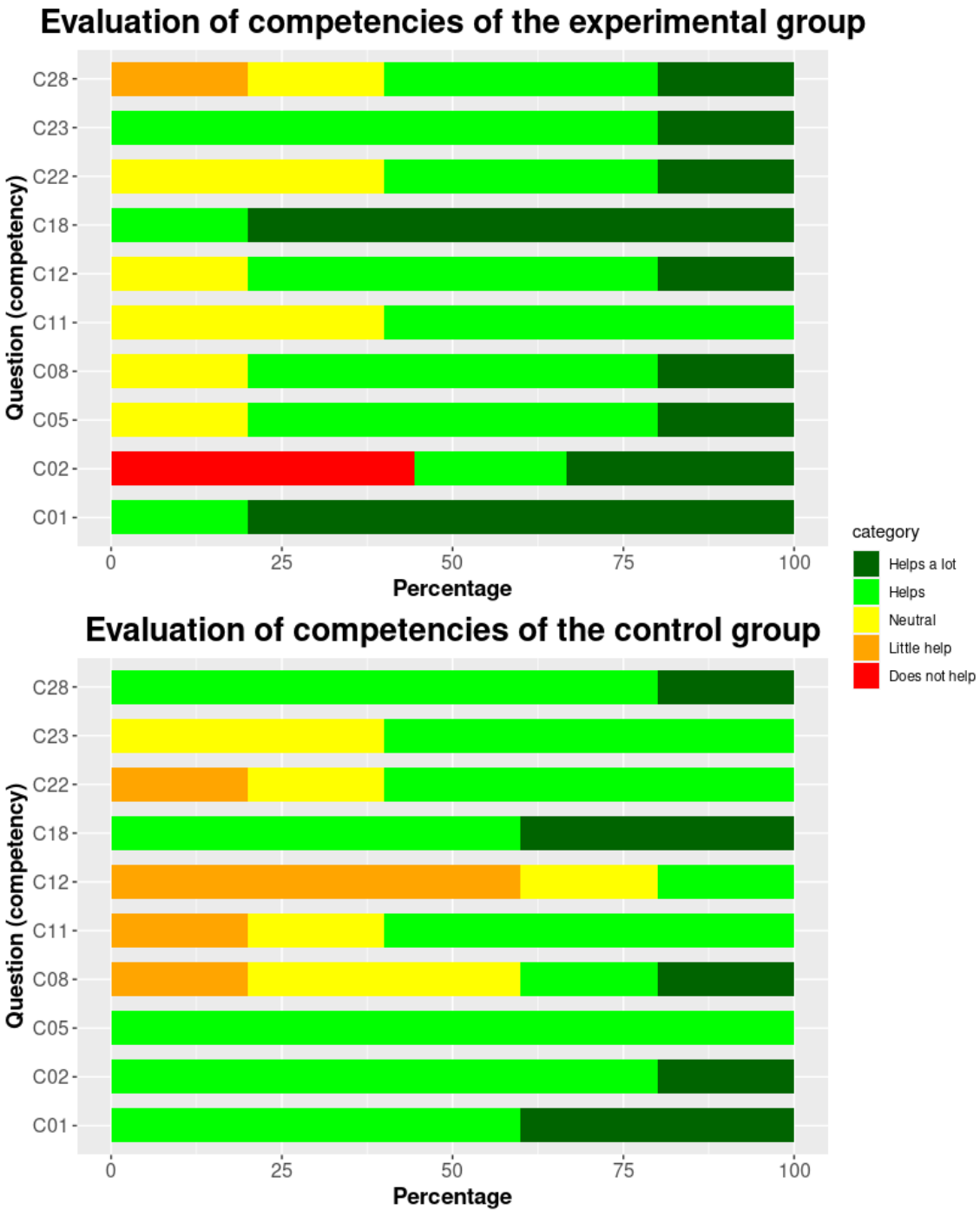

Figura 5-17.: Respuestas a la pregunta: Evalúe en qué grado cree que el curso de AS que acaba de proponer, ayuda a conseguir cada una de las siguientes competencias.

A continuación se muestra algunos de los comentarios enviados por parte del pastor.

- "If I think of a paper with 7 software architecture patterns then the second section might be useful. Maybe the problem is with the current name of the paper. It is a paper with one pattern but most of the paper is introducing terms and preparing for something big which is yet to come".
- "Coming to the pattern itself. The problem statement and the corresponding forces could benefit from some improvements in terms of their structure and coherence.The problem statement appears to be fragmented and lacking a clear flow. It would be beneficial to reorganize the sentences and provide a more cohesive narrative that directly addresses the challenges or issues faced in the context of small project-based training".
- "The forces themselves could be more explicitly stated and linked to the problem statement. Currently, some of the forces mentioned are more general statements rather than specific conflicts or tensions that need to be resolved. (See POSA5 specifically section: Forces, the heart of every pattern language). It would be helpful to provide more context and background information to enhance the understanding of the problem. While the problem statement and forces touch upon relevant aspects of small project-based training, they lack clarity, and alignment with the purpose of the pattern and may require restructuring and further refinement to improve the overall coherence".
- "While reading through this pattern, I often think if the pattern is more about quality attributes rather than small projects. Maybe there is something missing with the name of the pattern, or maybe this topic is too large to fit in one pattern. It is either a higher level pattern (it can be divided into more patterns), or we should look at it from a different angle (neither small project based training nor quality attribute they are both details of how to)".

Durante la conferencia, los artículos aceptados fueron organizados en los llamados Writers' Workshops Groups. El artículo enviado fue discutido en el grupo 6 junto a cinco artículos más. La sesión de discusión del artículo se realizó el jueves 24 de octubre. Al comienzo se expuso brevemente de qué se trata el artículo. En seguida, cada uno de los participantes al Focus Group dieron sus opiniones positivas y negativas del artículo con un buen grado de profundidad y debate. A continuación se presentan los comentarios más relevantes:

- "Who is this pattern language for? Because this is, by the way, the first complete pattern language that we've seen. I think it's really important who is the audience for the pattern language? People who teach software architecture. Okay, now let's dig into that for a second who teaches software architecture. Professor just in academia and maybe in industry as well. Put your finger on it, folks. There's a whole world outside of academia. In fact, most of the world is outside of academia in industry.".
- "What you're suggesting is basically that we go pattern by pattern and talk to univer-

sity professors who have taught software architecture. Maybe interview 25 professors at different universities who are teaching software architecture and find out what they're using and then find out if they're using any of the patterns in the catalog. And that's different from doing a literature survey, because the literature survey is only going to capture what people wrote. Do something like Alexander does, placing stars next to each of the patterns to indicate the confidence level of the pattern's applicability.".

- "I think it would also be useful to interview the authors of the articles to obtain more information".
- "'What did you think of the first picture on page three? Well, can I say two things for my part? Well, actually, I like this kind of chart. It's good, it's really cool. But it could be improved by making the pair of problems and the corresponding solutions much better, not just summary, but the summary of the problem and the corresponding solution. So, in other words, the table didn't convey enough information to really help you understand the structure. Exactly. Yeah. And another thing is that you use some pretty specific technical terms here as well. Well, architecture patterns might be okay, but also architecture tactics and some attributes. So maybe it's better to explain the technical terms in this area, maybe at the beginning, maybe introduction or summary before describing each pattern in detail. It is my opinion".
- "What about figures one and two on page four, were they helpful or not? I found them useful, especially figure one with the relationships. But I think it's probably possible to combine both figures. I think you already tried that with the gray color and the white color. Ring, but there's no legend that says what the gray and white color notes are on this chart. But I think it's the same distinction as in figure two. So they can probably be combined in one figure".
- "We have different types of patterns. One is the gray ones and the others are the white ones. The first ones focus more on how the topics are conveyed or taught and then there are the ones that focus more on how to design the class. The four white colored patterns have an arrow use towards those of problem solving. It's almost a totally connected graph. So I think this figure can be simplified. Yeah, okay. And I think these were very well intentioned, but I think either a legend or a simplification or both would help show the classification better".
- "When you're doing a big pattern language, the most common and most reasonable way to express that pattern language is with the idea of roots and leaves. So you have a root pattern and then you have leaf patterns. So you start with the root pattern that essentially sets the context for that particular category of things that you're looking at. And then you have leaf patterns that say these are the alternatives and these are the things that you have to do in order to get from this one to the next one to the next one to the next one. And it seemed like this one was missing a root pattern. Maybe

the root pattern is as simple as building a software architecture course".

"What was the basic difference between the two open source patterns? I mean, one was to find the students must choose an existing open source project and the other was that the teacher creates some specialized open source project just for the class. If you just want to deal with the software design issues, maybe contributing to a real open source project is not a great idea, because you're going to run into a lot of difficulties because of the human issues".

- "What other things did you like about the things that were presented in this document? Well, I like those patterns that describe how to incorporate some pedagogical aspects, the connection with competencies, also the advantage and disadvantage. So each pattern is well organized, well structured, well, well, although maybe some patterns could be shortened a little bit or divided into smaller patterns".
- "I think it is advisable to separate the problem from the problematic context. The problem should have one or two sentences and then give the context".
- "I have another comment about the field called level. It says that it describes the complexity of the pattern. However, it is assigned to each pattern separately. So I don't see the link knowing that a pattern can be applied in different architecture courses".

En resumen, enviar el artículo a la En resumen, la participación en la conferencia PLoP 2023 resultó en una experiencia enriquecedora que no solo mejoró de manera significativa el trabajo, sino que también facilitó el establecimiento de contactos profesionales, validó nuestras ideas y contribuyó al progreso de la comunidad de patrones en el ámbito del software. A través de las valiosas recomendaciones y comentarios recibidos, se hizo evidente que aún queda un extenso camino por recorrer; los patrones requieren continuas mejoras y pruebas en diversos escenarios para alcanzar su máximo potencial.

5.8. Estudio de caso 4: Unimayor

Esta sección describe un estudio de caso que permitió examinar el impacto que tuvo la aplicación de SAGITA (Software Architecture Guideline) en el curso de Arquitectura y Diseño de Software de la Institución Universitaria Colegio Mayor del Cauca (Unimayor), Popayán. La selección de este estudio de caso se hizo por ser un caso representativo, es decir, un curso de pregrado de Diseño y Arquitectura de Software. Por otro lado, para la docente es la primera vez que orienta este curso, por lo tanto, la guía se probó en las diferentes etapas de desarrollo del curso.

5.8.1. Diseño del estudio de caso

Objetivo

El objetivo de este estudio de caso fue obtener una evidencia empírica del impacto que tuvo para el docente y para los estudiantes aplicar SAGITA a un curso de AS.

Preguntas de investigación

RQ1: ¿Qué impacto tienen los patrones de formación en el desarrollo de habilidades de los estudiantes al final del curso?

RQ2: ¿Cómo fue la utilidad percibida y la facilidad de uso de la Guía por el docente en este estudio de caso?

RQ3: ¿Cuál fue la satisfacción del curso por parte de los estudiantes?

Contexto del estudio

Este caso se llevó a cabo en el segundo periodo académico 2023 del curso Arquitectura y Diseño de Software del programa de Ingeniería Informática de la Institución Universitaria Colegio Mayor del Cauca, Popayán. Este curso fue cursado por 27 estudiantes del semestre sexto. El estudio de caso es de *mejora*, pues trata de mejorar ciertos aspectos en el fenómeno estudiado.

Los objetivos de aprendizaje del curso son:

- Apropiar los patrones de desarrollo de software para mejorar el desarrollo de software.
- Conocer patrones y estilos arquitectónicos en el desarrollo de software.

Para este estudio de caso SAGITA estaba en su última versión.

5.8.2. Preparación de la recolección de datos

Las técnicas usadas para la recolección de datos fueron las encuestas, análisis de documentos y la entrevista semi-estructurada [7]. Las preguntas de la entrevista aplicadas al docente al finalizar el curso fueron:

[7] En una entrevista semi-estructurada las preguntas son planeadas pero no necesariamente son desarrolladas en el orden que se estableció, sino que durante el desarrollo de la conversación puede decidir el orden [103]

1. ¿Una vez leída la Guía de diseño de cursos de AS, describa qué cambios hizo en su curso a nivel de contenido, objetivos, metodología?
2. ¿La planificación de su curso de AS mediante la guía fue de utilidad, vale la pena utilizarla, por ejemplo, le ahorró tiempo y esfuerzo, o por el contrario le agregó más trabajo?
3. ¿La Guía a través de las pasos que plantea para diseñar un curso de AS, es fácil de seguirla, tiene pasos complejos de entender?
4. ¿Acorde a la Guía de Diseño de Cursos de AS, qué patrones de formación decidió aplicar en su curso y por qué?
5. ¿Explique cómo implementó (en qué momento del semestre, con qué recursos humanos y técnicos) los patrones de formación?
6. Qué impacto cree que tuvieron estos patrones en el desarrollo de habilidades de sus estudiantes a nivel de creación, documentación y evaluación de la AS?
7. Qué dificultades tuvo aplicando los patrones de formación durante el desarrollo del curso?
8. ¿Considera que los patrones de formación que empleó, permiten acercar la formación en arquitectura de software al mundo real de la industria de software?
9. ¿Qué le hace falta a la Guía propuesta para que cumpla su propósito de proponer un conjunto de pasos y estrategias de apoyo en el diseño y ejecución de cursos de arquitecturas de software a nivel de pre-grado?

5.8.3. Ejecución del estudio de caso

El estudio de caso se ejecutó gracias a la colaboración de la docente del curso, la Magíster María Isabel Bastidas, quien tuvo buena disposición para aplicar SAGITA en su curso.

Inicialmente a la docente del curso se le dio a conocer la Guía SAGITA. La docente siguió cada una de las etapas de la guía y generó un documento con la planificación. La Figura **5-18** ilustra parte de la planificación que realizó la docente donde involucró temas, actividades y estrategia a utilizar (El documento completo esta en en este enlace).

Fecha	Horas	Capitulo	Temas
14 agosto	2	Principio de diseño de sw	Presentación del curso Concepto de diseño de software
15 agosto	1		Capítulo 1. Principios de diseño de software
21 agosto festivo	0		N/A
22 agosto	1		Principios SOLID
28 agosto	2	Introducción a la arquitectura de sw	Atributos de calidad
29 agosto	1		**Atributos de calidad - taller**
4 Septiembre	2		Análisis de requerimientos Enfocados en arquitectura SW. Ciclo de vida y atributos de calidad. Documentacion de sw
5 septiembre	1	Patrones de diseño I	**Small Project-based Training.** Patrones Arquitectura de sw - Small Project-based Training. Patrones creacionales: Patrón Fabrica Abstracta y Patrón Builder. Patrón. Patrón decorador y patrón adaptador.
11 septiembre 1parcial	2		Patrones Arquitectura de sw. Patrones creacionales: Patrón Fabrica Abstracta y Patrón Builder. Patrón. Patrón decorador y patrón adaptador
12 septiembre 1 parcial	1	Estilo arquitectónico I - MVC	Arquitectura en capas. Modelo – Vista - Controlador (MVC)
18 septiembre 1 parcial	2		Patrones Arquitectura de sw - Primer corte: creación de documento de arquitectura V1 y Se abordará estructura monolítica - atributo de modificabilidad. Arquitectura en 3 capas VMC.
19 septiembre 1 parcial	1		Patrones Arquitectura de sw - Primer corte: creación de documento de arquitectura V1 y Se abordará estructura monolítica - atributo de modificabilidad. Arquitectura en 3 capas VMC.

Figura 5-18.: Planificación del curso de Arquitectura y Diseño de Software de Unimayor

En cuanto a los patrones de formación se aplicaron varios. El primer patrón fue de formación Mini-Projects-based training. Cada mini-proyecto estuvo dedicado a entender el funcionamiento de un estilo de arquitectura. Se trabajaron los estilos de capas, microkernel, eventos y microservicios. Además de la clase presencial, la docente suministró un ejemplo en código java por cada estilo y se les pidió a los estudiantes realizar una extensión a cada programa.

El segundo patrón de formación aplicado fue Large Project-based Training enfocado a desarrollar una aplicación grande. La docente decidió armar equipos de tres estudiantes. Cada equipo trabajó un proyecto distinto, la Tabla **5-6** muestra los proyectos. Cabe destacar que

la iniciación de estos proyectos se remonta al semestre anterior, durante la asignatura denominada "Construcción de Software". En este curso, se llevó a cabo la ingeniería de requisitos y el modelado del negocio como parte integral de la preparación. Este enfoque proporcionó a los estudiantes una experiencia de aprendizaje práctica. Es relevante señalar que estos proyectos no contaron con un cliente real; en su lugar, la profesora asumió el papel de cliente para guiar el desarrollo de manera más directa.

Tabla 5-6.: Proyectos asignados en el curso de Arquitectura y Diseño de Software de Unimayor

No	Proyecto	Descripción	Estilo arquitectura	Tecnologias
1	Registro de pagos	Aplicación web que permite hacer pagos por internet	Orientada a eventos	php (frontend), python (backend), RabbitMQ
2	Entrenamiento de perros y gatos en casa	Aplicación web que da consejos útiles para entrenar y educar perros y gatos en casa.	C/S	html, javascript (frontend), nodejs (backend)
3	Bienvenido a Silvia	Aplicación móvil que muestra lugares de Silvia (Cauca) como restaurantes, tiendas, hospedajes y droguerías.	C/S	Flutter, Firebase
4	Armador de platos para restaurantes	Aplicación desktop que permite armar tipos de platos: orientales, italianos, etc.	C/S	Java con sockets tcp
5	Comidas y hábitos saludables	Aplicación desktop que sugiere las comidas saludables y cómo evitar ciertas enfermedades según las necesidades del usuario para adelgazar o ganar masa muscular.	C/S	java con sockets tcp
6	Entrenamiento de perros y gatos en casa	Aplicación web que da consejos útiles para entrenar y educar perros y gatos en casa.	C/S y MVC	ASP.NET con Entity Framework
7	Planes saludable	Aplicación web que da planes saludables para adelgazar o ganar peso	C/S	html, javascript
8	Asignador de estudiantes y cursos.	Aplicación web que permite armar cursos de estudiantes y hacer consultas	Microservicios	Spring Boot
9	Citas odontológicas en linea	Aplicación web que permite a un paciente perteneciente a una EPS solicitar y gestionar citas odontológicas en linea	Orientado a eventos	php, RabbitMQ

El tercer patrón aplicado fue Cases-based training mediante la variante con charlas con arquitectos de software invitados de la industria. Aunque se habían planificado tres charlas para el curso, finalmente por cuestiones de tiempo se pudieron ejecutar dos. Este patrón es exigente en términos de dedicación pues requiere contactar los invitados, hablar con ellos previamente, buscar agendas comunes, etc. La primera charla titulada "Cómo las decisiones de negocio impactan en la Arquitectura de Software" fue orientada por el ingeniero Santiago Hyun mediante vídeo conferencia. Esta charla mostró el proceso de diseño de arquitectura de una aplicación de comercio electrónico para una empresa que fabrica y vende productos derivados del gusano de seda. La segunda charla titulada "La arquitectura de software en el proceso de desarrollo" fue dirigida por por el ingeniero Nelson Fabián Chantre de manera

presencial (ver Figura **5-19**). El ingeniero Chantre se desempeña como arquitecto de software de Home Center y mostró el papel que juega la AS en el proceso de desarrollo de los sistemas de información de Home Center.

Figura 5-19.: Charla de ingeniero Nelson Fabián Chantre en Unimayor.

El cuarto patrón de formación aplicado fue Problem-solving-based training donde se aplicó una Kata de arquitectura (ver Figura **5-20**). Este taller fue cuidadosamente planificado con tres semanas de anticipación e involucró buscar un sitio adecuado, invitar docentes e ingenieros con experiencia en AS que jugaron el rol de evaluadores, diseñar una rúbrica de evaluación, incentivar a los estudiantes con un premio y preparar a los estudiantes con ejercicios de katas previos. Esta planificación se consignó en este documento que fue enviado por email a los estudiantes e ingenieros evaluadores. Este taller se realizó en cinco pasos:

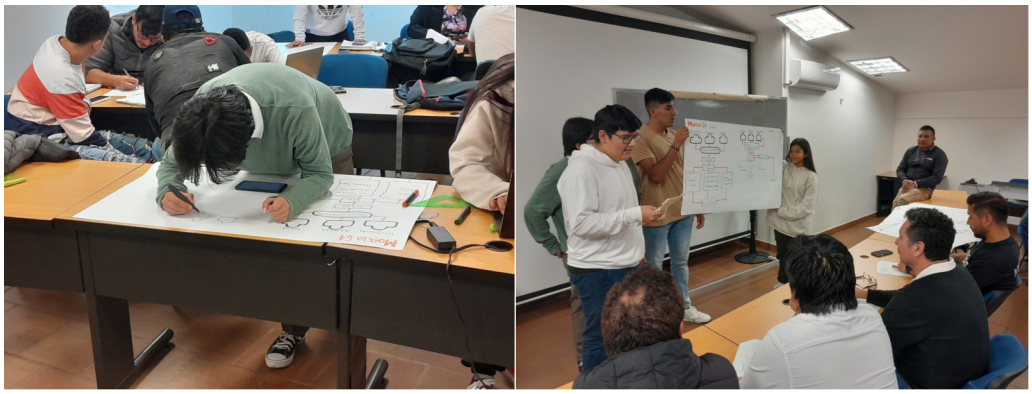

Figura 5-20.: a) Estudiantes de Unimayor resolviendo la kata de arquitectura, b) Estudiantes presentando su solución a los evaluadores

1. *Preparación*. Se realizó la conformación de grupos y la asignación de la kata a trabajar. Se crearon cuatro grupos. Los estudiantes conformaron sus grupos según su afinidad. La Kata a trabajar fue elegida al azar del sitio https://nealford.com/katas/list.html, en este caso salió la kata "Check Your Work", que pide diseñar un sistema software para automatizar la calificación de tareas sencillas de programación de una universidad (5 minutos).

2. *Discusión.* Se hizo el diseño de la solución por equipos. Se hicieron supuestos sobre las tecnologías a usar y se permitió hacer preguntas a los moderadores. Mientras los equipos trabajaron en su solución, los evaluadores también solucionaron la kata (50 minutos).
3. *Presentación.* Se definieron 1 o 2 representantes por equipo para que explicaran tanto el problema como la solución propuesta. Cada equipo tuvo máximo 6 minutos (30 minutos).
4. *Evaluación.* A medida que cada equipo presentó su diseño, los evaluadores anotaron sus evaluaciones y observaciones en la hoja de evaluación. Al final los evaluadores dieron una realimentación de los diseños (20 minutos).
5. *Premiación.* Se computaron las evaluaciones de los equipos. El equipo con el promedio más alto fue declarado como el ganador y recibieron un premio (10 minutos).

Al final del curso se aplicó la entrevista a la docente María Isabel Bastidas, la transcripción de la entrevista está en el Apéndice F.

5.8.4. Resultados y discusión

Haciendo una triangulación de los datos recolectados de entrevistas, encuestas y documentos, se procedió a dar respuesta a las preguntas de investigación.

RQ1: ¿Qué impacto tienen los patrones de formación en el desarrollo de habilidades de los estudiantes al final del curso?

El patrón **Large Project-based Training** permitió que los estudiantes llevaran a la práctica los conocimientos teóricos de AS mediante el desarrollo de un proyecto grande y complejo. Lo novedoso es que cada equipo trabajó con un proyecto distinto, aplicando tecnologías y una arquitectura distinta. Al finalizar el curso, los avances de los proyectos fueron distintos, algunos de ellos fueron exitosamente terminados pero otros no.

En general la aplicación de este proyecto de desarrollo tuvo un impacto aparentemente positivo entre los estudiantes. El desarrollo del curso giró entorno a este proyecto permitiendo que los estudiantes desarrollaran una experiencia práctica resolviendo un problema de cierto grado de complejidad. Sin embargo, no se tuvieron los resultados esperados, pues la calidad y nivel de avance de los proyectos no fue el esperado. Esta situación se originó debido a que los estudiantes carecían de la experiencia y habilidades suficientes en programación y desarrollo de proyectos. En consecuencia, la elección del patrón de formación no se basó únicamente en la conveniencia de las circunstancias (o fuerzas del patrón), sino más bien en la imposición de las normas del curso.

Gracias a la aplicación del patrón **Cases-based training** se logró coordinar dos charlas con arquitectos de software invitados de la industria. Los alumnos pudieron apreciar casos reales de la industria. Estas charlas tuvo varios impactos positivos, algunos fueron:

- *Vinculación con la industria.* La interacción con profesionales invitados ofreció a los estudiantes la posibilidad de establecer conexiones directas con expertos en actividad, proporcionándoles una visión realista de la aplicación práctica de los conceptos teóricos de AS en el entorno laboral.
- *Aplicación práctica.* Apreciar casos reales de diseño de software brindó a los estudiantes la oportunidad de presenciar cómo los conceptos de AS se transforman en soluciones tangibles y aplicables en situaciones concretas del mundo real.

A partir de la primera charla se aplicó una encuesta para medir la satisfacción (ver Figura **5-21**), la utilidad percibida (ver Figura **5-22**) y el logro de competencias en AS. Como se puede apreciar en las gráficas tanto la satisfacción como la utilidad percibas fueron muy bien valoradas por los estudiantes.

Respecto a las competencias con la charla del invitado de la industria, se preguntó a cada uno de los estudiantes: *Evalúe en qué grado la charla con el ingeniero Santiago Hyun, ayudó a conseguir cada una de las siguientes competencias. En todas las preguntas se utilizará la siguiente escala; 1. No ayuda, 2. Ayuda poco, 3. Neutral, 4. Ayuda, 5. Ayuda mucho* (La descripción de las competencias evaluadas se pueden apreciar en el Apéndice C de las cuales se tomaron las de la categoría obligatorias). Las repuestas se pueden apreciar en la Figura **5-23**. Podemos ver de manera general que todas las competencias son evaluadas con un alto porcentaje de "Ayudó mucho" y "Ayuda". La competencia C22 es la única con una valoración de "ayudó poco".

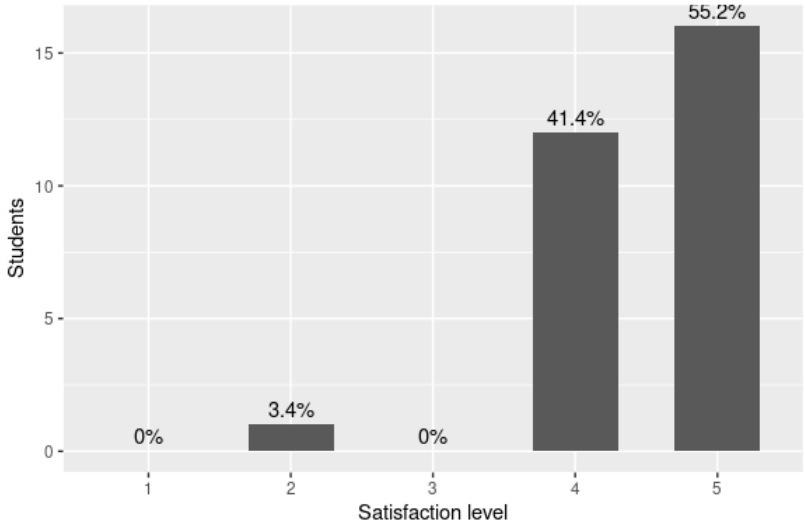

Figura 5-21.: Respuestas a la pregunta: ¿Qué tan satisfecho quedó con la charla del Ingeniero Santiago Hyun, siendo 1:Nada satisfecho y 5:Muy satisfecho?

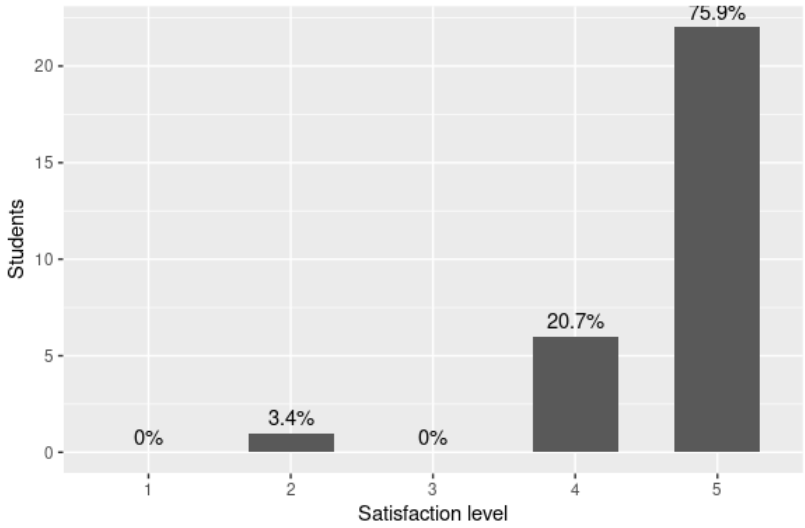

Figura 5-22.: Respuestas a la pregunta: ¿Qué tan útiles le parecen incorporar este tipo de charlas en los cursos de arquitectura de software en el proceso de formación de arquitectos de software, siendo 1:Nada satisfecho y 5:Muy satisfecho?

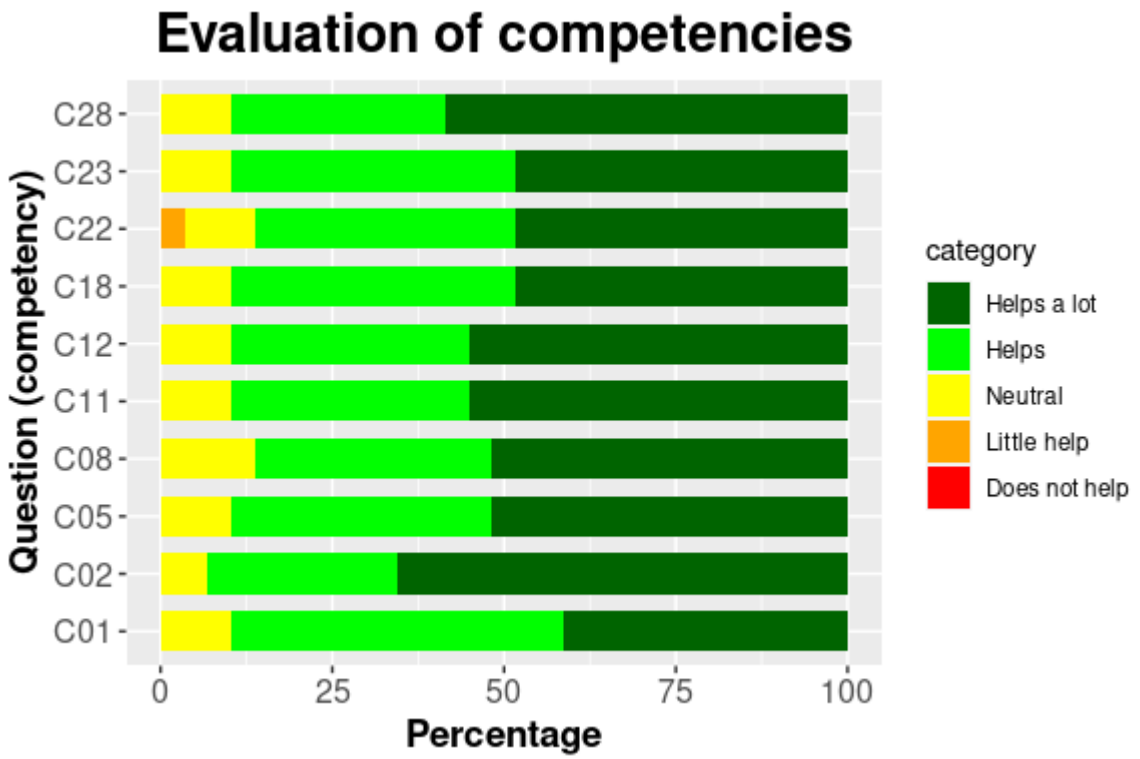

Figura 5-23.: Respuestas a la pregunta: ¿Evalúe en qué grado la charla con el ingeniero Santiago Hyun, ayudó a conseguir cada una de las siguientes competencias?

El cuarto patrón de formación aplicado en este estudio de caso fue **Problem-solving-based training**. En este caso se lo aplicó mediante un taller de **kata de arquitectura**. Al final de este taller se aplicó una encuesta a los estudiantes para saber qué competencias se alcanzan con este patrón. En esta ocasión no se usó la escala de Likert. En su lugar, se le pidió a los asistentes a la kata que escoger las competencias que a su criterio se pueden desarrollar usando katas de arquitectura de software (ver Figura **5-24**). En las respuestas obtenidas se puede evidenciar que las competencias más elegidas son:

1. C18. Analiza críticamente los requisitos de software funcionales y de atributos de calidad.
2. C02. Diseña consistentemente la arquitectura de software definiendo cómo los componentes interactúan entre si.
3. C01. Identifica claramente los atributos de calidad relevantes del software que conducirán la arquitectura de un sistema de software a construir.
4. C05. Evalúa independientemente una arquitectura de software para determinar la satisfacción de los requisitos funcionales y no funcionales.

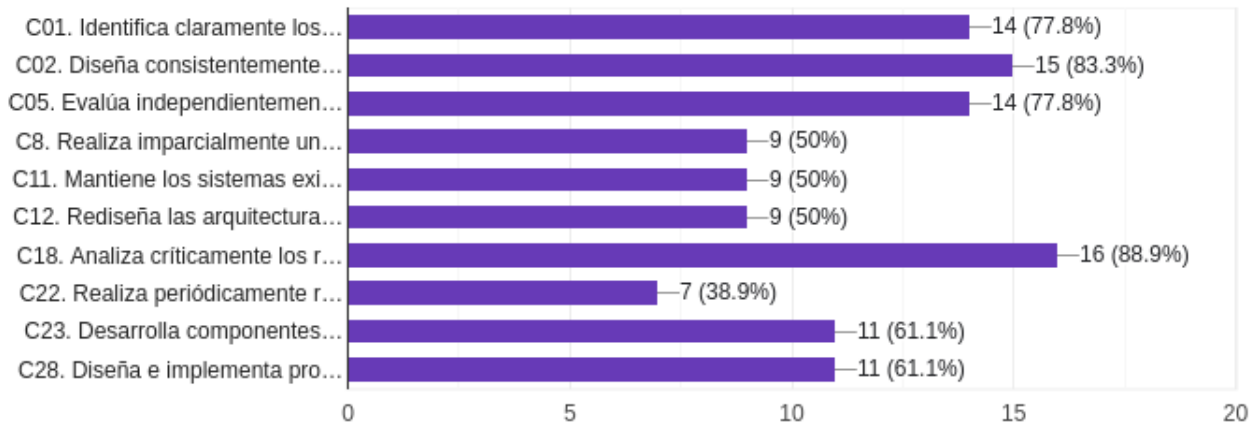

Figura 5-24.: Respuestas a: Elija las competencias que a su criterio se pueden desarrollar usando ejercicios de katas de arquitectura de software

En la encuesta también se solicitó a los estudiantes que compartieran comentarios, tanto positivos como negativos, acerca del taller de katas. A continuación, se presentan algunas de las respuestas proporcionadas al respecto (no hubo comentarios negativos):

- "Que se incluyan ejercicios más prácticos como el de las katas que se hizo anteriormente,porque estos espacios incluyen al estudiante, me gustó mucho".
- "Son muy buenas, porque permite ampliar conocimientos sobre arquitectura de software".
- "Con estos ejercicios podemos referenciar a nivel real como interactuar con distintos tipos de arquitectura y como implementar estrategias para realizar una planeación correcta de un proyecto software, para nosotros como estudiantes es esencial este tipo de ejercicios, conocer además distintos puntos de vista y obtener feedback de lo que realizamos".
- "Es una estrategia bastante atractiva para quienes aún no han entrado al mundo de la arquitectura, me parece que es muy útil y práctica para abarcar varios temas respecto a la problemática presentada".
- "Excelente Actividad".
- "'Es una gran práctica y permite una interiorización de ejercicio y práctica y entender cómo mejorar y también aprender más con práctica".
- "Fue una experiencia increíble, me gustaría que se repita".
- "Me parece un tema muy importante, ya que nos permite desarrollar evolución en nuestro entorno, ya que nos prepara mediante ejercicios para la vida real".

Los resultados de las evaluaciones de la kata se pueden apreciar en la Tabla **5-7**. De acuerdo a los promedios obtenidos, el equipo que se destacó como ganador fue el cuarto, alcanzando una nota promedio de 4.0. Más allá de la calificación lograda, los estudiantes disfrutaron de una actividad lúdica que les brindó la oportunidad de colaborar en equipo, compartir un momento agradable y recibir retroalimentación valiosa por parte de evaluadores expertos. Los evaluadores, conformados por dos profesores de arquitectura de software y dos arquitectos de software provenientes de la industria, aportaron comentarios precisos y acertados gracias a su experiencia. Más que simples apreciaciones sobre los trabajos presentados, proporcionaron consejos prácticos relacionados con el diseño y la arquitectura de software. Esto permitió a los participantes tomar conciencia de la crucial importancia de la arquitectura al desarrollar un sistema de software, especialmente en lo referente al manejo de los requisitos no funcionales.

Tabla 5-7.: Evaluación de la kata de arquitectura.

Equipo No.	Eval.1	Eval.2	Eval.3	Eval.4	Promedio
1	1.0	2.5	4.0	3.5	2.8
2	2.0	3.0	3.5	3.5	3.0
3	2.5	2.5	3.5	3.5	3.0
4	3.0	4.0	4.5	4.5	4.0

A pesar de la buena percepción de los estudiantes por la kata, la planificación y organización de la kata representaron un desafío considerable para la docente, quien se vio inmersa en una serie de tareas exigentes. La labor de conseguir evaluadores invitados, provenientes tanto de la industria como de la academia, implicó un esfuerzo adicional que demandó habilidades de coordinación y comunicación. La necesidad de acordar un horario común entre los distintos participantes añadió una capa de complejidad, requiriendo negociaciones cuidadosas para garantizar la participación de todos de manera efectiva.

Además, la búsqueda del espacio adecuado para llevar a cabo la kata y la adquisición de los materiales necesarios se convirtieron en aspectos cruciales de la preparación. Encontrar un entorno propicio para el evento, que cumpliera con los requisitos logísticos y proporcionara un ambiente propicio para la actividad, demandó una evaluación minuciosa de las opciones disponibles. Asimismo, la docente se vio comprometida en la tarea de adquirir los materiales esenciales, asegurando que cada detalle estuviera cuidadosamente considerado para garantizar el desarrollo exitoso de la kata.

Respecto a las competencias logradas al final del curso, aplicando los cuatro patrones de formación, se preguntó a cada uno de los estudiantes: *Evalúe en qué grado este curso le ayudó a conseguir cada una de las siguientes competencias requeridas por la industria de software. En todas las preguntas se utilizará la siguiente escala; 1. No ayuda, 2. Ayuda poco, 3. Neutral, 4. Ayuda, 5. Ayuda mucho* (La descripción de las competencias evaluadas se

pueden apreciar en el Apéndice C de las cuales se tomaron las de la categoría obligatorias). Las repuestas se pueden apreciar en la Figura **5-25**. Podemos ver que todas las competencias los estudiantes manifiestan que el curso "ayudó mucho' y 'ayudó' a desarrollarlas en una gran proporción.

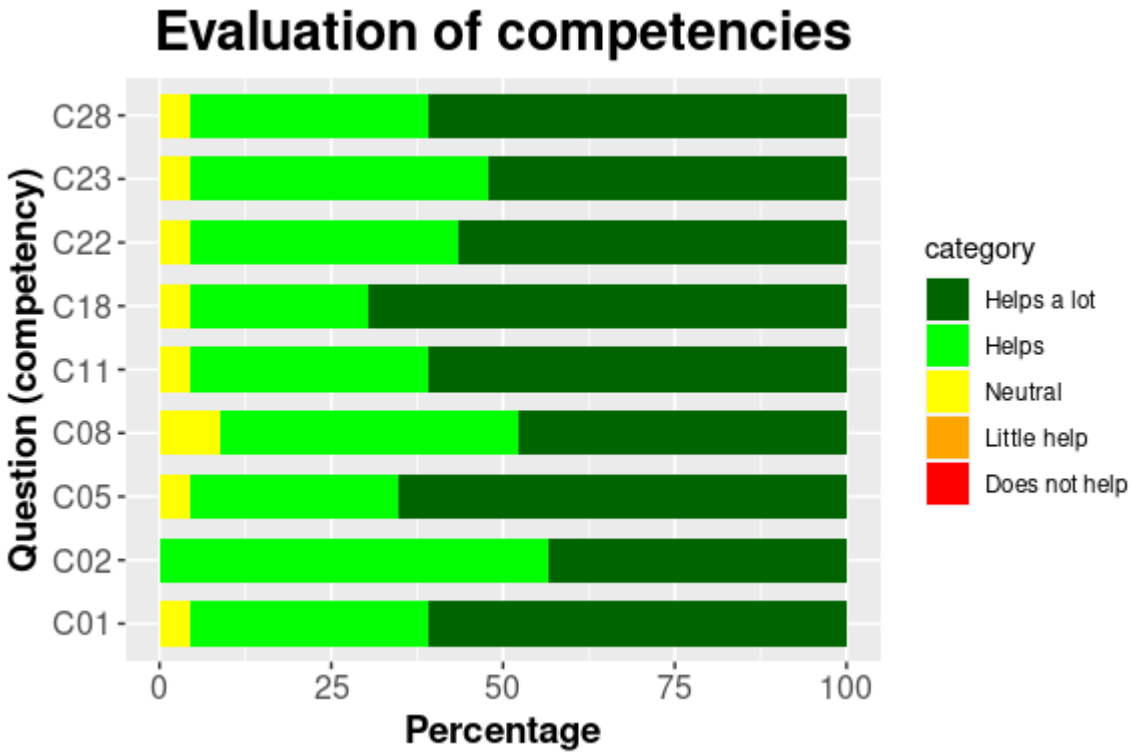

Figura 5-25.: Respuestas a la pregunta: Evalúe en qué grado cree que el curso de Patrones de Arquitectura, ayudó a conseguir cada una de las siguientes competencias.

RQ2: ¿Cómo fue la utilidad percibida y la facilidad de uso de la Guía por el docente en este estudio de caso?

La docente destacó la utilidad significativa de la guía durante la fase de planificación del curso de Diseño y Arquitectura de Software. En particular, resaltó que la guía se convirtió en un recurso invaluable al proporcionar orientación detallada y paso a paso. En su primera experiencia enseñando este curso, la guía no solo le ahorró tiempo precioso, sino que también facilitó la creación de un curso integral y preciso. Esto resultó esencial al considerar las competencias críticas que la industria espera de los futuros graduados en el ámbito de la arquitectura de software.

Además de su utilidad integral, la docente elogió la facilidad de uso de la guía. Según sus comentarios, los cinco pasos delineados en la guía son de fácil comprensión, lo que simplifica

la tarea de seguir el proceso. En cada etapa, la guía ofrece una amplia gama de recursos, consejos prácticos y plantillas, lo que contribuye significativamente a su facilidad de uso y a la eficiencia general del diseño del curso. La combinación de una estructura clara y recursos prácticos la convirtió en una herramienta esencial en la preparación y ejecución del curso.

RQ3: ¿Cuál fue la satisfacción del curso por parte de los estudiantes?

A través de encuestas aplicadas a los estudiantes se capturaron comentarios valiosos sobre las estrategias aplicadas durante el curso. A continuación se describen algunos:

- "Personalmente creo que fueron estrategias muy acertadas, teniendo en cuenta que estas nos ayudan a practicar lo que vamos aprendiendo, a demás de que simulan escenarios de la vida real, también podemos tener una buena retroalimentación de parte de la docente como de los evaluadores, dependiendo del trabajo".
- "Las Katas son muy interesantes, dinámicas y a la vez ponemos en práctica lo aprendido en la asignatura".
- "Fueron excelentes, se instruyeron de forma perfecta y nos dio pies para el manejo de cada enseñanza bajo las prácticas y la implementación".

Respecto a la satisfacción en general del curso, se hicieron varias preguntas a través de una encuesta a 23 estudiantes. Se usó una escala Likert, siendo 1 Nada satisfecho y 5 Muy satisfecho. Respecto a la pregunta, *¿En general, qué tan satisfecho quedó con lo aprendido al final del curso en temas de diseño y arquitectura de software?* Las respuestas en un diagrama de cajas, se pueden apreciar en la Figura **5-26**. La gráfica de cajas muestra representa la distribución de la satisfacción de los estudiantes que participaron en el curso de Dieseño y Arquitectura de Software de la Unimayor. Las mayoría de las respuestas están entre 4 y 5 con una mediana de 5. Esto sugiere que la satisfacción general tiende a ser muy alta, ya que la mayoría de las respuestas están en el extremo superior de la escala de 1 a 5.

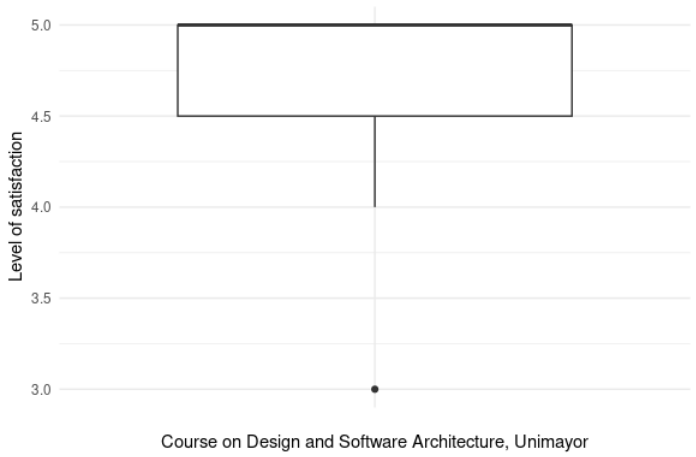

Figura 5-26.: Gráfico de cajas de la pregunta ¿En general, qué tan satisfecho quedó con lo aprendido al final del curso en temas de diseño y arquitectura de software?

También se realizaron preguntas para conocer la utilidad percibida. La Figura **5-27** muestra los resultados de la pregunta *¿Qué tan útiles le parecen las habilidades, conocimientos que desarrolló a lo largo del curso en las áreas de Arquitectura de Software para afrontar proyectos del mundo real?* Podemos decir que la mayoría de las respuestas fueron el máximo valor de 5. Esto sugiere que la satisfacción general tiende a ser muy alta. Hay dos datos atípicos que son 4 y 3.

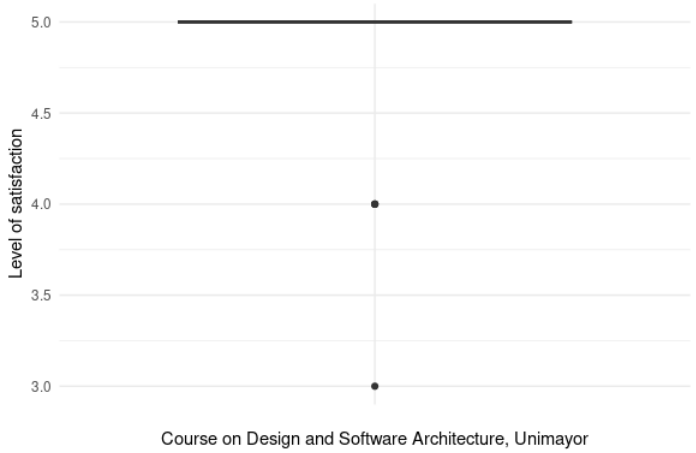

Figura 5-27.: Respuestas a ¿Qué tan útiles le parecen las habilidades, conocimientos que desarrolló a lo largo del curso en las áreas de Arquitectura de Software para afrontar proyectos del mundo real?

5.8.5. Discusión del estudio de caso

Mediante la implementación de la estrategia de mini-proyectos, los estudiantes de Unimayor tuvieron la oportunidad de adquirir conocimientos sobre diversos estilos de arquitectura, con el objetivo de mejorar el cumplimiento de los atributos de calidad. Específicamente, los estudiantes se familiarizaron con los estilos de capas, microkernel, orientado a objetos y microservicios. La habilidad para aplicar estos estilos arquitectónicos de manera efectiva, en función de los atributos de calidad deseados, se posiciona como uno de los aspectos más cruciales en el ámbito de la arquitectura de software.

Aunque la utilización de un proyecto de software extenso representó una experiencia enriquecedora para los estudiantes de Unimayor, lamentablemente, no se alcanzaron los resultados esperados. Los proyectos entregados al término del curso no lograron cumplir con los requisitos planteados, lo que permitió confirmar la importancia de que los estudiantes posean un nivel adecuado de destreza en programación y desarrollo de proyectos de software al abordar este tipo de patrón de formación.

La estrategia de invitar a un conferencista de la industria a la clase, con el propósito de compartir experiencias reales de diseño de software, enriqueció significativamente la vivencia educativa de los estudiantes. Esta iniciativa les brindó conocimientos concretos, perspectivas prácticas y conexiones valiosas con el ámbito laboral. Sin embargo, debido a la limitación temporal y a la abundancia de días festivos durante este semestre, únicamente se pudo llevar a cabo dos de las tres charlas programadas.

La implementación del taller kata de arquitectura se reveló como una estrategia sumamente fascinante, una que no había sido empleada en los estudios de caso previos. Estas katas han brindado a los estudiantes la capacidad de identificar tanto requisitos funcionales como no funcionales, realizar suposiciones de diseño, proponer una arquitectura de software, documentarla utilizando el modelo C4, evaluar su diseño a través de la retroalimentación de expertos y, en última instancia, colaborar en equipo para tomar decisiones y alcanzar consensos.

La utilización y aplicación de SAGITA generó resultados positivos, evidenciando la satisfacción del docente. Este sistema facilitó la planificación del curso y la implementación de estrategias que vincularon de manera efectiva el contenido con la realidad de la industria del software. Asimismo, para evaluar el impacto de estas estrategias en la formación de los estudiantes, se llevaron a cabo encuestas que reflejaron, en términos generales, niveles altos de satisfacción.

5.9. Síntesis de la validación

El propósito de la presente investigación se centra en cómo lograr el desarrollo de competencias relacionadas con la creación, evaluación y documentación de arquitecturas de software en potenciales egresados de programas de ingeniería de sistemas y afines, acorde a la necesidades de la industria. Para ello, se propuso unos patrones de formación que ayuden a diseñar cursos de AS a los docentes. Los patrones de formación están contenidos en una guía llamada SAGITA que orienta a los docentes en cinco pasos en el diseño de sus cursos.

La validación de SAGITA y los patrones de formación se hizo a través de varios ciclos de investigación-acción utilizado varios mecanismos: validaciones con expertos, la realización de un experimento y cuatro estudios de caso. Cada ciclo permitió evolucionar y mejorar la guía y el catálogo de patrones de formación.

La guía y los patrones fueron validados a través de un experimento buscando medir la eficiencia. Para calcular la eficiencia de SAGITA se usó *el tiempo empleado por los docentes en aplicar la guía*. Sin embargo, los tiempos por sí solos pueden no proporcionan una imagen completa de la eficiencia. Por ello, se consideró otros factores como los recursos utilizados, la calidad del resultado y la satisfacción de los docentes.

Una vez obtenido el catálogo de patrones de formación, se eligió uno de los patrones para ser sometido a una validación con profesores de la Universidad Nacional de la Plata, Argentina mediante un taller de escritura. Esta validación buscó recibir realimentación del patrón en aspectos como el nombre, la estructura y sobretodo en la relación entre el problema y la solución. Una vez obtenida la realimentación del grupo de expertos, las recomendaciones se aplicaron tanto al patrón evaluado y a los demás patrones.

Otra validación con expertos se hizo enviando los patrones de formación a la La conferencia "Pattern Languages of Programs (PLoP)". Un grupo de expertos revisó intensamente cada patrón y emitieron comentarios valiosos durante varias rondas que permitieron mejorar todos los patrones propuestos de forma significativa.

Por otro lado, se realizaron tres estudios de caso buscando medir el impacto de SAGITA y el catálogo de patrones en tres cursos relacionados con AS. El primer estudio de caso se hizo con un curso de Ingeniería de Software de la Universidad la Universidad Autónoma de Occidente de la ciudad de Cali - Colombia. Este curso permitió evaluar algunas estrategias interesantes como el uso de un proyecto de software con clientes reales. El segundo estudio de caso se realizó en un curso de Arquitectura de Software de la Universidad Nacional de la Plata, Argentina. Este curso permitió evaluar varios patrones de formación y el impacto que tuvo en los estudiantes y en el docente. El tercer estudio de caso se llevó a cabo en un curso de Diseño y Arquitectura de Software en la Institución Universitaria Colegio Mayor del Cauca. Este curso fue interesante porque la docente era la primera vez que orienta el curso, lo

cual permitió evaluar la totalidad de la guía. Los tres estudios de caso brindaron resultados positivos en cuanto a la utilidad percibida y la facilidad de uso de SAGITA y en el desarrollo de habilidades de los estudiantes al final de los cursos. Se puede decir que SAGITA brindó beneficios a los docentes porque les permitió diseñar eficientemente sus cursos y aplicar los patrones de formación para mejorar la formación de los estudiantes en el desarrollo de las competencias de AS.

En otro ciclo se llevó a cabo un experimento que buscó medir la eficiencia de SAGITA. Se trató de un experimento específico donde los sujetos fueron divididos en dos grupos. El primer grupo fue el experimental y trabajó con la guía llamada SAGITA. El segundo grupo es el de control y trabajó con una guía basada en la revisión de la literatura. Las dos guías contenían en general los mismos pasos, pero SAGITA contenía los patrones de formación listos para ser usados por el docente, en cambio la segunda guía contenía estrategias de formación y sus referencias bibliográficas. La idea de la segunda guía fue simular el esfuerzo que hace un docente preparando un curso de AS a partir de la revisión bibliográfica. Se obtuvo como conclusión que el tiempo promedio del grupo experimental fue menor que el tiempo promedio del grupo de control, por lo tanto, la *eficiencia* en el diseño del curso con la SAGITA y el catálogo de patrones es *mayor* que la eficiencia en el diseño del curso con la información y estrategias más recomendadas por la literatura.

Finalmente, gracias a todos los mecanismos de validación utilizados, SAGITA y los patrones de formación alcanzaron un buen nivel de calidad y posiblemente al aplicarlos a otras Universidades se obtengan resultados positivos en el diseño de cursos de AS. El catálogo de patrones de formación propuesto orienta al docente sobre el qué y el cómo enseñar, y ha permitido lograr significativamente el desarrollo de las competencias más relevantes para la industria de software actual y futura, relacionadas con la creación, evaluación y documentación de una arquitectura software,en potenciales egresados de programas de ingeniería de sistemas y afines.

6. Conclusiones y recomendaciones

6.1. Conclusiones

Un curso de Arquitectura de Software acorde a las necesidades de la industria es algo esencial en los planes de estudio de programas de ingeniería de sistemas y afines. Sin embargo, formar estudiantes de pregrado con las competencias de AS que demanda la industria tiene muchos desafíos. Algunos de estos desafíos tienen como causas: la naturaleza abstracta de las arquitecturas, las bases que requieren los estudiantes, la dificultad para recrear proyectos y ambientes en las aulas con características similares a los de la industria, las dificultades de trabajar en equipo, la falta de recursos y contenidos actualizados, la falta de experiencia de los docentes para cubrir el gran espectro de proyectos reales en términos de estilos arquitectónicos y las variedad de tecnologías disponibles, entre otros.

Por medio de esta investigación se busca organizar el conocimiento que ayuden a los docentes universitarios a diseñar y conducir sus cursos de AS de tal forma que logren enfrentar las dificultades de enseñar AS y por otro lado, a alinear las competencias de los estudiantes con las necesidades de la industria. Para ello, como resultado de una revisión de la literatura y de experiencias preliminares se identifican, estructuran y documentan un catálogo de siete patrones de formación articulados dentro de una guía que ayuda a los profesores diseñar y ejecutar cursos, a nivel de pregrado, que desarrollen competencias de creación, evaluación y documentación de arquitecturas de software acordes con las expectativas de la industria de software. Estos patrones de formación resuelven los problemas que recurrentemente deben enfrentar los profesores, recrean la forma de trabajo en el aula de manera similar a los ambientes utilizados por la industria, ayudando a tomar decisiones al docente y por otro lado, organizan el conocimiento arquitectónico para que las actividades de clase se puedan estructurar y facilitar el aprendizaje incremental. La capacidad de tomar decisiones de arquitectura es algo fundamental en el proceso de formación, pero requiere tiempo para desarrollarse y se acumula a través de múltiples experiencias de proyectos.

Este catálogo de patrones de formación orienta a los docentes sobre qué y cómo enseñar, logrando el desarrollo de las competencias más relevantes para la industria actual y futura relacionadas con la creación, evaluación y documentación de SA en los potenciales egresados de los programas de informática.

El primer paso para encontrar soluciones efectivas sobre cómo formar en AS, es tener el listado de competencias mínimas a desarrollar desde el pregrado. Para obtener ese listado se siguió una serie de pasos de manera secuencial y sistemática involucrando en workshops a los docentes e ingenieros de la industria para establecer desde ambos mundos las necesidades de formación en términos de competencias. Estas competencias han servido para establecer el alcance de los patrones de formación. Las competencias encontradas están clasificadas en las categorías obligatorias, opcionales y fuera de alcance para un curso de pregrado. A la vez están claramente diferenciadas en dos grupos: la primera las relacionadas con el desarrollo de un sistema nuevo, y la segunda, con el mantenimiento de un sistema software junto con su arquitectura. Claramente las competencias de la segunda clasificación son mucho más complejas de desarrollar desde la academia. A los estudiantes les dificulta más entender y modificar la arquitectura de un sistema existente que proponer uno nuevo desde cero.Cabe aclarar que esta segunda es la más demandada por la industria.

Varios patrones de formación trabajan desarrollando proyectos de software en clase. Es importante trabajar con proyectos que expongan gradualmente a los estudiantes a una mayor complejidad del sistema de software de manera incremental. Se busca que los estudiantes sean capaces de adaptar y evolucionar la arquitectura, simulando lo que sucede en proyectos reales.

Los patrones de formación fueron validados con varios mecanismos, por ejemplo, realimentación con expertos, un experimento y cuatro estudios de caso. La revisión exhaustiva de cada patrón se llevó a cabo a través de las validaciones con expertos mediante dos workshops, quienes aportaron valiosos comentarios en múltiples rondas. Estos aportes contribuyeron de manera significativa a la mejora de todos los patrones propuestos. Estas actividades permitieron socializar y mejorar el catálogo para facilitar su lectura, comprensión y aplicación.

Los cuatro estudios de caso arrojaron resultados favorables respecto a la percepción de utilidad y la facilidad de uso de SAGITA, así como en el desarrollo de las capacidades de los estudiantes al concluir los cursos. Esto sugiere que SAGITA aportó ventajas a los docentes, al permitirles una planificación eficaz de sus cursos y la aplicación de patrones de formación para elevar la calidad de la educación de los estudiantes en el desarrollo de competencias en AS.

En otro ciclo de investigación, se llevó a cabo un experimento destinado a evaluar la eficacia de SAGITA. Este experimento se diseñó específicamente con la división de los participantes en dos grupos. El primer grupo, conocido como el grupo experimental, empleó la guía denominada SAGITA, mientras que el segundo grupo, designado como el grupo de control, utilizó una guía basada en la revisión de literatura. Ambas guías, en líneas generales, seguían los mismos pasos; no obstante, SAGITA contenía patrones de formación previamente elaborados y listos para la utilización por parte de los docentes, mientras que la segunda guía incluía estrategias de formación y sus respectivas referencias bibliográficas. La intención detrás de

la segunda guía era emular el proceso que un docente lleva a cabo al preparar un curso de AS basado en una revisión exhaustiva de la literatura disponible. Los resultados de este estudio revelaron que el tiempo promedio requerido por el grupo experimental fue inferior al tiempo promedio del grupo de control. En consecuencia, se puede afirmar que la eficiencia en el diseño del curso, cuando se emplea SAGITA y su catálogo de patrones, es superior a la eficiencia en el diseño del curso basado en la información y estrategias que predominan en la literatura especializada.

Por consiguiente, dado que cada estudio de caso en que se aplicó SAGITA y los patrones de formación, mostraron una valoración positiva por los interesados, y que además, cada hallazgo fue considerado en el camino como una oportunidad de mejoramiento en los ciclos de investigación-acción, se puede afirmar que se cumple la hipótesis planteada en la Sección 1.4: "Un catálogo de patrones de formación (reportados por la literatura y evaluados por la industria) de arquitectura de software para estudiantes de pregrado, que orienten al docente sobre el qué y el cómo enseñar, permitirán lograr significativamente el desarrollo de las competencias más relevantes para la industria de software actual y futura, relacionadas con la creación, evaluación y documentación de una arquitectura software,en potenciales egresados de programas de ingeniería de sistemas y afines".

6.2. Limitaciones

Si bien la investigación es de alcance global ya que la revisión de la literatura fue una fuente rica para la identificación de los patrones de formación y se acudió al juicio de expertos internacionales, los resultados y hallazgos empíricos de tipo primario corresponden a experiencias de Colombia y Argentina. La selección de los estudios de caso se realizó teniendo en cuenta la principalmente disponibilidad de los docentes.

La industria de la arquitectura de software está en constante evolución. Las tendencias y las tecnologías cambian con el tiempo. La presente investigación podría no haber tenido en cuenta todos los cambios futuros, y las necesidades de la industria podrían evolucionar en una dirección no considerada en este estudio como es el caso de la IA, los nuevos retos en la industria 4.0 y la computación cuántica.

Las instituciones educativas pueden variar significativamente en cuanto a sus recursos, enfoques pedagógicos, aspectos culturales o de idioma y requisitos del programa. Las recomendaciones y patrones de formación propuestos pueden no ser igualmente aplicables en todos los contextos educativos, y la investigación podría no abordar todas las diferencias posibles.

Los estudios de caso se llevaron a cabo en un período de tiempo relativamente corto (de un período académico), esto podría limitar la comprensión de cómo se mantienen los beneficios de los patrones de formación a largo plazo. La eficacia de la formación a lo largo de un

período extendido podría ser diferente y requeriría investigaciones futuras.

Una limitación considerable podría ser el tamaño de la muestra en el experimento de validación. El número de participantes en el experimento fue limitado, esto podría afectar la generalización de los resultados a una población más amplia. Aunque los hallazgos pueden ser prometedores, podrían necesitar una validación adicional en una escala más grande.

6.3. Recomendaciones

Los docentes que decidan aplicar SAGITA y el catálogo de patrones deben estar dispuestos a salirse del esquema tradicional de formación de un curso. Deben enfrentar retos para alinear el curso con las necesidades y exigencias de la industria, puesto que cada patrón de formación exige al docente planificar recursos adicionales. Para aplicar los patrones se recomienda leer cuidadosamente las fuerzas para determinar la conveniencia del patrón en el ámbito de cada universidad.

La colaboración estrecha entre las instituciones académicas y la industria es un elemento fundamental. Los docentes y estudiantes pueden beneficiarse al mantenerse actualizados con las tendencias y requisitos actuales de la industria, lo que se puede lograr a través de patrones de formación que involucran charlas con invitados de la industria, clientes y proyectos reales.

Existe la necesidad de proporcionar a los docentes de AS los recursos y apoyo necesario. Esto incluye capacitación, el acceso a materiales actualizados y la libertad académica para ajustar los planes de estudio.

Un curso de AS exige que los docentes se mantengan al día con las tendencias y avances en temas de AS y tecnologías relacionadas. La formación continua y la participación en comunidades profesionales pueden ser cruciales para mantenerse actualizados.

En las universidades se debería realizar evaluaciones periódicas de impacto en los graduados en temas de AS. Esto permitiría rastrear su éxito en la industria y medir si los enfoques de formación empleados están cumpliendo con los objetivos.

Ls investigación continua es importante en la mejora de la formación en AS. Es primordial que otros investigadores amplíen el trabajo planteado en esta investigación y aporten nuevas perspectivas al campo. Se necesita realizar estudios de caso en otras universidades representativas de la región y el país para validar los patrones propuestos.

6.4. Trabajos futuros

El grupo IDIS seguirá investigando alrededor de los patrones de formación, obteniendo más evidencia empírica que permita diseminar los patrones, mejorar y complementar el catálogo y establecer mayor formalidad al lenguaje. Es posible que surjan nuevos patrones. La guía y el catálogo se podrían combinar con otras herramientas para crear un centro de entrenamiento que permita formar arquitectos de software en corto tiempo según las necesidades de la industria.

Sería beneficioso realizar una validación a gran escala para validar aun más la eficacia de SAGITA y el catálogo de patrones de formación propuestos en diferentes instituciones y contextos educativos. Esto podría incluir la colaboración con múltiples universidades con la industria y la recopilación de datos a lo largo de varios años.

Otro trabajo futuro interesante, sería diseñar y desarrollar herramientas educativas específicas, como tutoriales, ejemplos que implementen estilos de arquitectura, estudios de caso, katas de arquitectura resueltos, plataformas en línea o recursos multimedia, que faciliten la implementación de SAGITA y los patrones de formación en AS. Sería útil desarrollar una herramienta inteligente que asista al docente en cada etapa del diseño del curso y que al final recomiende la combinación adecuada de los patrones de formación que debe utilizar.

Se podría realizar una evaluación de empleabilidad de los graduados. Esto implicaría realizar un seguimiento a largo plazo de los graduados a quienes se los sometió a los patrones de formación de AS y medir su empleabilidad y éxito en la industria de software. Esto proporcionaría una evaluación más sólida de la efectividad de la formación.

En el campo del diseño de los materiales educativos, resultaría beneficioso contar con una comunidad dedicada a compartir contenidos y experiencias, organizados en torno a competencias y patrones de formación específicos. Sería especialmente enriquecedor establecer una comunidad que se comunique utilizando el lenguaje de patrones, empleándolo como un vehículo para mejorar las prácticas educativas en la disciplina. La creación de este espacio de intercambio permitiría que el lenguaje de patrones evolucione de manera orgánica, convirtiéndose así en un medio altamente efectivo para la transmisión de ideas relacionadas con la formación en arquitectura de software.

Finalmente, se podría investigar cómo capacitar y apoyar de manera efectiva a los docentes en la implementación de SAGITA y los patrones de formación, y cómo la formación docente continua puede mejorar la calidad de la enseñanza.

6.5. Publicaciones

A lo largo de esta investigación se han publicado varios artículos. Además, están pendientes publicar los resultados de la fase de validación. A continuación los artículos que se han publicado en revistas y eventos.

- W. Libardo Pantoja, Julio Ariel Hurtado. Dificultades y retos de la formación de arquitectos de software en programas de pregrado: Causas y efectos. In *Tercer Simposio Doctoral del Doctorado en Ciencias de la Electrónica*, Popayán, Colombia, 2021. Available: https://sites.google.com/view/simposiodoctoral2021/.
- W. Libardo Pantoja, Julio Ariel Hurtado, and Arvind W Kiwelekar. Aligning Software Architecture Training with Software Industry Requirements. *International Journal of Software Engineering and Knowledge Engineering*. Vol. 33, pag. 435-460, 2023, doi: https://doi.org/10.1142/S0218194023500031.
- W. Libardo Pantoja, Julio Ariel Hurtado, Andrés Solano, and Bandi Ajay. Visión de las competencias en arquitectura de software integrando las perspectivas de la industria y la academia. In *Sexto Congreso Andino de computación, informática educación*, Pasto, Colombia, 2023. Available: http://cacied2023.udenar.edu.co/.
- W. Libardo Pantoja, Julio Ariel Hurtado, Luis Mariano Bibbó, Alejandro Fernández, and Bandi Ajay. Towards a Software Architecture Training Pattern Language. In *Pattern Languages Of Programs Conference PLOP2023*, Illiois, USA, 2023. Available: https://hillside.net/plop/2023/.
- W. Libardo Pantoja, Julio Ariel Hurtado, Bandi Ajay, and Arvind W Kiwelekar. Training Software Architects Suiting Software Industry Needs: A Literature Review. *Education and Information Technologies*, 2023, doi: https://doi.org/10.1007/s10639-023-12149-x.

A. Anexo: Estudios primarios del Mapeo Sistemático

Tabla A1.: Primary studies

No	Authors	Title	Name of the conference or journal	Ref	Category
S1	Chandar R. Rupakheti, Stephen V. Chenoweth	Teaching software architecture to undergraduate students: an experience report	International Conference on Software Engineering 2015	[104]	Teaching of software architecture
S2	Alf Inge Wang	Extensive Evaluation of Using a Game Project in a Software Architecture Course	ACM Trans. Comput. Educ.	[120]	Teaching of software architecture
S3	Zheng Li	Using public and free platform as a service (PaaS) based lightweight projects for software architecture education	Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training	[66]	Teaching of software architecture
S4	Li Zhang, Yanxu Li, Ning Ge	Exploration on Theoretical and Practical Projects of Software Architecture Course	15th International Conference on Computer Science and Education, ICCSE 2020	[128]	Teaching of software architecture
S5	Eng Lieh Ouh, Benjamin Kok Siew Gan, Yunghans Irawan,	Did our Course Design on Software Architecture meet our Student's Learning Expectations?	2020 IEEE Frontiers in Education Conference (FIE)	[69]	Teaching of software architecture
S6	Matthias Backert, Thomas Blum, Rüdiger Kreuter, Frances Paulisch, Peter Zimmerer,	Software Curriculum @ Siemens — The Architecture of a Training Program for Architects	2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE T)	[10]	Teaching of software architecture
S7	Ouh Eng Lieh, Yunghans Irawan,	Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course	2018 IEEE Frontiers in Education Conference (FIE)	[67]	Teaching of software architecture
S8	Shahani Markus Weerawarana, Amal Shehan Perera, Vishaka Nanayakkara,	Promoting creativity, innovation and engineering excellence	Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE) 2012	[123]	Teaching of software architecture
S9	Samuil Angelov, P de Beer	Designing and Applying an Approach to Software Architecting in Agile Projects in Education	Journal of Systems and Software	[9]	Teaching of software architecture
S10	Arie Van Deursen, Maurício Aniche, Joop Aué, et al	A Collaborative approach to teaching software architecture	Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE	[118]	Teaching of software architecture
S11	Ouh Eng autLieh, Yunghansors Irawan,	Teaching adult learners on software architecture design skills	Proceedings - Frontiers in Education Conference, FIE	[68]	Teaching of software architecture
S12	Douglas C. Schmidt, Zach McCormick,	Producing and delivering a MOOC on pattern-oriented software architecture for concurrent and networked software	SPLASH 2013 - Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, and Applications: Software for Humanity	[106]	Teaching of software architecture

Tabla A1.: Primary studies (Continued)

No	Authors	Title	Name of the conference or journal	Ref	Category
S13	Weigang Li	Teaching Reform and Practice of Software Architecture Design Course under the Background of Engineering Education	Proceedings of the 2019 International Conference on Advanced Education, Management and Humanities (AEMH 2019)	[65]	Teaching of software architecture
S14	Paolo Ciancarini, Stefano Russo, Vincenzo Sabbatino	A Course on Software Architecture for Defense Applications	Proceedings of 4th International Conference in Software Engineering for Defence Applications	[22]	Teaching of software architecture
S15	Arvind W. Kiwelekar	A Software Architecture Teacher's Dilemmas	arXiv preprint arXiv:2101.09434	[60]	Teaching of software architecture
S16	Muhammad Aufeef Chauhan, Christian W Probst, Muhammad Ali Babar	Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods	Book: Agile and Lean Concepts for Teaching and Learning: Bringing Methodologies from Industry to the Classroom	[21]	Teaching of software architecture
S17	Fáber D Giraldo, Sergio F Ochoa, MyriamHerrera et al.	Applying a distributed CSCL activity for teaching software architecture	International Conference on Information Society (i-Society 2011)	[44]	Teaching of software architecture
S18	Zhenyan Ji, Jing Song	Improved Teaching Model for Software Architecture Course	Proceedings of the 2015 International Conference on Education, Management, Information and Medicine	[55]	Teaching of software architecture
S19	Bingyang Wei, Yihao Li, Lin Deng, Nicholas Visalli	Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application	Book: Studies in Computational Intelligence	[124]	Teaching of software architecture
S20	Rafael Capilla, Olaf Zimmermann, Carlos Carrillo, Hernán Astudillo	Teaching Students Software Architecture Decision Making	Software Architecture 2020	[16]	Teaching of software architecture
S21	Patrick de Beer, Samuil Angelov	Fontys ICT, Partners in Education Program: Intensifying Collaborations Between Higher Education and Software Industry	Proceedings of the 2015 European Conference on Software Architecture Workshops	[29]	Teaching of software engineering
S22	Gustavo Pinto, Clarice Ferreira, Cleice Souza, et al.	Training software engineers using open-source software: The students' perspective	Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE-SEET 2019	[97]	Teaching of software engineering
S23	Kun May, Bo Yang, Jin Zhou, et al.	Outcome-based school-enterprise cooperative software engineering training	ACM International Conference Proceeding Series	[74]	Teaching of software engineering
S24	Robert Chatley, Tony Field	Lean learning - Applying lean techniques to improve software engineering education	Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering and Education Track, ICSE-SEET 2017	[20]	Teaching of software engineering

Tabla A1.: Primary studies (Continued)

No	Authors	Title	Name of the conference or journal	Ref	Category
S25	Daniela Castelluccia, Bari Aldo, Giuseppe Visaggio, et al.	Teaching evidence-based software engineering: learning by a collaborative mapping study of open source software	ACM SIGSOFT Software Engineering Notes	[18]	Teaching of software engineering
S26	Peter J Clarke, Jairo Pava, Yali Wu, Tariq M King	Collaborative Web-Based Learning of Testing Tools in SE Courses	Proceedings of the 42nd ACM Technical Symposium on Computer Science Educatio	[23]	Teaching of software engineering
S27	Fernanda Gomes Silva, Paolo Dos Santos, et al.	FLOSS in Software Engineering Education: Supporting the Instructor in the Quest for Providing Real Experience for Students	Proceedings of the XX-XIII Brazilian Symposium on Software Engineering	[110]	Teaching of software engineering
S28	Charles Thevathayan, Margaret Hamilton	Imparting Software Engineering Design Skills	Proceedings - Frontiers in Education Proceedings of the Nineteenth Australasian Computing Education Conference, FIE	[116]	Teaching of software engineering
S29	Charles Thevathayan, Margaret Hamilton	Imparting Software Engineering Design Skills	Proceedings of the Nineteenth Australasian Computing Education Conference	[75]	Teaching of software engineering
S30	Matti Luukkainen, Arto Vihavainen, Thomas Vikberg	Three Years of Design-Based Research to Reform a Software Engineering Curriculum	Proceedings of the 13th Annual Conference on Information Technology Education	[72]	Teaching of software engineering
S31	Sriram Mohan, Stephen Chenoweth, Shawn Bohner	Towards a Better Capstone Experience	Proceedings of the 43rd ACM Technical Symposium on Computer Science Education	[77]	Teaching of software engineering
S32	Rune Hjelsvold, Deepti Mishra	Exploring and Expanding GSE Education with Open Source Software Development	ACM Transactions on Computing Education	[50]	Teaching of software engineering
S33	Lynette Johns-Boast, Shayne Flint	Simulating industry: An innovative software engineering capstone design course	2013 IEEE Frontiers in Education Conference (FIE)	[56]	Teaching of software engineering
S34	Maria Palacin-Silva, Jayden Khakurel, Ari Happonen, et al.	Infusing Design Thinking into a Software Engineering Capstone Course	2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)	[86]	Teaching of software engineering
S35	Ravi Shankar Pillutla, Anuradha Alladi	Methodology to bridge the gaps between engineering education and the industry requirements	Eurocon 2013	[96]	Teaching of software engineering
S36	Azim Abdool, Akash Pooransingh	An industry-mentored undergraduate software engineering project	2014 IEEE Frontiers in Education Conference (FIE) Proceedings	[2]	Teaching of software engineering
S37	Jayden Khakurel, Jari Porras	The Effect of Real-World Capstone Project in an Acquisition of Soft Skills among Software Engineering Students	2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE T)	[58]	Teaching of software engineering

Tabla A1.: Primary studies (Continued)

No	Authors	Title	Name of the conference or journal	Ref	Category
S38	L Sun, Ting-qin Lv, Lichun Pan	Reinforcing practice teaching of Software Engineering for fostering the creative talent	2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)	[115]	Teaching of software engineering
S39	Cécile Péraire	Dual-Track Agile in Software Engineering Education	2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)	[92]	Teaching of software engineering
S40	Kevin D Wendt, Ken Reily, Mats P E Heimdahl	First Steps towards Exporting Education: Software Engineering Education Delivered Online to Professionals	2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)	[125]	Teaching of software engineering
S41	Zahra Shakeri Hossein Abad, Muneera Bano, Didar Zowghi	How Much Authenticity Can Be Achieved in Software Engineering Project Based Courses?	Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training	[1]	Teaching of software engineering
S42	Christoph Matthies, Ralf Teusner, Guenter Hesse	Beyond Surveys: Analyzing Software Development Artifacts to Assess Teaching Efforts	2018 IEEE Frontiers in Education Conference (FIE)	[73]	Teaching of software engineering
S43	N Pratheesh, T Devi	Assessment of student's learning style and engagement in traditional based software engineering education	2013 International Conference on Intelligent Interactive Systems and Assistive Technologies	[99]	Teaching of software engineering
S44	Padmashree Desai, G H oshi, M ijayalaskhmi,	A novel approach to carrying out mini project in Computer Science Engineering	2012 IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA)	[34]	Teaching of software engineering
S45	Jeevamol Joy, V. G. Renumol,	Activity oriented teaching strategy for software engineering course: An experience report	Journal of Information Technology Education: Innovations in Practice	[57]	Teaching of software engineering
S46	Carlos Portela, Alexandre Vasconcelos, Sandro Oliveira, Maurício Souza	The Use of Industry Training Strategies in a Software Engineering Course: An Experience Report	2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)	[98]	Teaching of software engineering
S47	Nicolás Martín Paez	A Flipped Classroom Experience Teaching Software Engineering	2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)	[85]	Teaching of software engineering
S48	Camelia Şerban, Virginia Niculescu, Andreea Vescan	Attaining competences in software quality oriented design based on cyclic learning	2020 IEEE Frontiers in Education Conference (FIE)	[130]	Teaching of software engineering
S49	G J Collins	Integrating Industry Seminars within a Software Engineering Module to Enhance Student Motivation	2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)	[26]	Teaching of software engineering

Tabla A1.: Primary studies (Continued)

No	Authors	Title	Name of the conference or journal	Ref	Category
S50	Jaejoon Lee, Gerald Kotonya, Jon Whittle, Christopher Bull	Software Design Studio: A Practical Example	2015 IEEE/ACM 37th IEEE International Conference on Software Engineering	[63]	Teaching of software engineering
S51	Stan Jarzabek	Teaching advanced software design in team-based project course	2013 26th International Conference on Software Engineering Education and Training (CSEE T)	[54]	Teaching of software engineering
S52	Adriana Lopes, Igor Steinmacher, Tayana Conte	UML Acceptance: Analyzing the Students' Perception of UML Diagrams	Proceedings of the XX-XIII Brazilian Symposium on Software Engineering	[70]	Complementary experiences
S53	Williamson Silva, Bruno Gadelha, Igor Steinmacher, Tayana Conte	What Are the Differences between Group and Individual Modeling When Learning UML?	Proceedings of the XX-XII Brazilian Symposium on Software Engineering	[111]	Complementary experiences

Tabla A1.: Primary studies (Continued)

No	Authors	Title	Name of the conference or journal	Ref	Category
S54	Anthony Ward, Adeyosola Gbadebo, Bidyut Baruah	Using job advertisements to inform curricula design for the key global technical challenges	2015 International Conference on Information Technology Based Higher Education and Training (ITHET)	[121]	Complementary experiences
S55	Matthias Galster, Samu Angelov	What makes teaching software architecture difficult?	Proceedings - International Conference on Software Engineering	[39]	Complementary experiences
S56	Zhewei Hu, Yang Song, Edward F Gehringer	Open-Source Software in Class: Students' Common Mistakes	Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training	[51]	Complementary experiences

B. Artículos de los cuales se extrajo los patrones de formación

Tabla A1.: Artículos de los cuales se extrajo los patrones de formación

No	Título del artículo	Ref.
1	Teaching software architecture to undergraduate students: an experience report	[104]
2	Extensive Evaluation of Using a Game Project in a Software Architecture Course	[120]
3	Using public and free platform as a service (PaaS) based lightweight projects for software architecture education	[66]
4	Exploration on Theoretical and Practical Projects of Software Architecture Course	[128]
5	Did our Course Design on Software Architecture meet our Student's Learning Expectations?	[69]
5	Did our Course Design on Software Architecture meet our Student's Learning Expectations?	[69]
6	Software Curriculum @ Siemens — The Architecture of a Training Program for Architects	[10]
7	Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course	[67]
8	Designing and applying an approach to software architecting in agile projects in education	[9]
9	A Collaborative Approach to Teaching Software Architecture	[118]
10	Teaching adult learners on software architecture design skills	[68]
11	Teaching Reform and Practice of Software Architecture Design Course under the Background of Engineering Education	[65]
12	Creating and Teaching a MOOC on Pattern-Oriented Software Architecture (POSA) for Concurrent and Networked Software	[106]
13	A Course on Software Architecture for Defense Applications	[22]
14	Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods	[21]
15	Applying a distributed CSCL activity for teaching software architecture	[44]
16	Improved Teaching Model for Software Architecture Course	[55]
17	Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application	[124]
18	Teaching Students Software Architecture Decision Making	[16]
19	Towards a case-based learning approach to support software architecture education	[82]
20	Using game development to teach software architecture	[120]
21	Comparison of learning software architecture by developing social applications versus games on the android platform	[127]
22	Archinotes: A tool for assisting software architecture courses	[117]
23	An Inverted Classroom Experience: Engaging Students in architectural thinking for agile projects	[24]
24	Requirements and Architecture Modeling in Software Engineering Courses	[46]
25	Smart Decisions: An Architectural Design Game	[19]
26	Supporting software architecture learning using runtime visualization	[43]
27	A role-playing game to teach ATAM (Architecture Trade-off Analysis Method) a simulation tool and case study	[78]
28	Project and team-based strategies for teaching software architecture	[119]
29	Scrum as a method of teaching software architecture	[122]
30	DecidArch: Playing cards as software architects	[61]
31	Applying case-based learning for a postgraduate software architecture course	[84]
32	Flipped Classroom Applied to Software Architecture Teaching	[45]
33	Training software engineers using open-source software: the students' perspective	[97]
34	Promoting creativity, innovation and engineering excellence	[123]

C. Tabla de Competencias

Estas competencias fueron obtenidas de una revisión bibliográfica y a cada una le asignamos un identificador C01, C02, C... Además, en un workshop que invitamos a docentes de arquitectura de software y arquitectos de la industria, hicimos una clasificación de las arquitecturas en tres grupos. Obligatorias, opcionales y fuera del alcance para un curso de pregrado.

Creation of an Architecture:

- C1: Clearly 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 achieve the evolution of 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 compo-

nents/services, integration).

- C29: Builds the product by facilitating the identification and correction of faults.

Evaluation of Future Technologiess

- C30: Explicitly evaluates enterprise software solutions and makes recommendations.
- C31: Carefully manages the introduction of 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.

D. Entrevista del estudio de caso del curso en la UAO

1. *Cómo se llama la asignatura donde se aplicó la Guía, en qué semestre de dicta, de qué programa(s) la matriculan.* La asignatura se llama Ingeniería de Software I y está dedicada al ciclo de desarrollo del software e involucra temas de arquitectura de software. Se dicta en quinto semestre y tiene 3 créditos académicos. Es una asignatura obligatoria para los estudiantes del programa de Ingeniería Informática y es electiva para el programa de Ingeniería Multimedia.

La UAO tiene además al asignatura Ingeniería de Software II, dedicada a temas de calidad de software.

La Universidad también tiene una asignatura electiva de Arquitectura de Software donde se trabajan distintos estilos de arquitectura.

2. *Describa qué temas de arquitectura de software se tratan en la asignatura Ingeniería de Software I.*

La asignatura está dividida en tres cortes. El primero dedicado a la ingeniería de requisitos. El segundo corte abarca los temas de diseño, por ejemplo, arquitectura de software, estilo de arquitectura en capas y MVC y diagramación de la arquitectura. En el curso no se alcanza a trabajar estilos de arquitectura distribuidos.

3. *Describa cómo es la estrategia de aprender arquitectura a través de un proyecto real de empresa.*

La asignatura trabaja con un proyecto de empresa. En este semestre (2022.2) estamos trabajando con OL Software. Esta empresa genera un proyecto pequeño de tal forma que pueda ser implementado en el curso. Este semestre estamos trabajando en el proyecto llamado Mapa de Conocimiento. El canal de comunicación con la empresa es el arquitecto de software quien se encarga de dar los requisitos y dar asesoría en la toma de decisiones de los estudiantes.

Las decisiones de qué tipo de arquitectura, el tipo de tecnología es la adecuada está a cargo de los estudiantes y son guiados por el docente y el arquitecto de la empresa.

El proyecto se desarrolla en equipo de 3 estudiantes. El equipo que presente la mejor solución,

generalmente tiene opción de seguir trabajando en la empresa. Esta incorporación se hace mediante una prueba que los estudiantes presentan ante la empresa finalizado el curso. Generalmente los estudiantes se vinculan medio tiempo y reciben capacitaciones posteriores.

4. *Tiene alguna encuesta que permitió medir la satisfacción del cliente cuando se le entregan los proyectos de desarrollo?* El curso tiene un enfoque iterativo e incremental. La primera entrega es donde se entregan historias de usuario y unos prototipos de la interfaz de usuario de baja fidelidad. De esta entrega hay una realimentación por parte de la empresa. La segunda entrega es un prototipo software funcional. Finalmente, la tercera entrega es el producto software implementado en su totalidad. En cada una de las entregas existen criterios de evaluación por parte del docente. En la tercera entrega es la única que tiene criterios de evaluación de la empresa, por ejemplo, si el producto se adecua a sus necesidades, originalidad, innovación del producto, calidad de la presentación oral que hacen los estudiantes.

Al final del semestre, la empresa ya tiene un conocimiento desempeño de cada equipo, en base a quienes son los que mas han preguntado y han presentado la mejor solución. El producto entregado al cliente está compuesto por codigo fuente, pruebas y documentación de arquitectura.

5. *Tiene alguna encuesta que permitió medir la satisfacción de los estudiantes?*

De forma institucional en la semana 7 los estudiantes aplican una co-evaluación (vistos como oportunidades de mejora) respecto al curso, metodologia, recursos, el profesor. Además, la Institución aplica a los estudiantes al final del curso una encuesta de satisfacción. Sin embargo estos resultados no son accesibles al docente sino un tiempo después. Es decir, el docente no tiene acceso a esos resultados. Además, esas encuestas podrían no se objetivas, pues los estudiantes generalmente las llenan con mucha prisa.

6. *Qué ventajas tiene su estrategia de trabajar un proyecto real?*

La principal ventaja es tener la oportunidad de comunicarse directamente con personas de una empresa: gerentes, arquitectos, desarrolladores. Esto genera mucho valor al estudiante porque reciben comunicación y realimentación, marca una diferencia importante.

7. *Qué dificultades tiene su estrategia de enseñanza con proyectos reales?*

Se han presentado muchas dificultades. Una fue que bases de datos no era prerrequisito de la asignatura, esto impedía desarrollar software real. Una dificultad es el tiempo. Una dificultad que se presenta en general en el curso es que material extra que se les da como guias, ejemplos, artículo, los estudiantes no los leen.

Generalmente la empresa les da un espacio de una hora semanal. En este espacio los estudiantes tienen que saludar, presentarse, contar lo que están haciendo. En ocasiones las preguntas que hacen les faltan fundamentos, o dan respuestas sin fundamento suficiente. Esto refleja poca lectura de los estudiantes en el trabajo de tiempo independiente.

Otra dificultad es sincronizar los temas teóricos con el avance del proyecto. Hay un punto en que esta sincronización no es posible, cuando ya deberían estar implementando aun están validando historias de usuario. Esto se presenta cuando los estudiantes no aprovechan adecuadamente el tiempo con la empresa para revisar diseños, prototipos o arquitectura.

No hemos tenido problemas consiguiendo empresas que nos faciliten proyectos para el curso. Esto que para otras universidades es difícil, en nuestro caso ha sido sencillo. En trabajos y proyectos previos con empresas se han ido abriendo espacios de cooperación. Unas semanas antes de iniciar semestre, se indaga a las empresas si tienen proyectos y ese espacio ha estado disponible siempre. La dificultad si ha sido al momento de concretar reuniones porque son muy ocupados. Entonces, definir el proyecto y su alcance es algo que tarda un tiempo y se debe hacer con anterioridad. Cuando no se ha podido concretar proyectos con empresas, hay la opción de recurrir a la División de Tecnologías de la Información de la misma Universidad y encontrar proyectos. Sin embargo para los estudiantes es más atractivo trabajar con empresas de afuera para tener la opción de vincularse a futuro cercano con la empresa.

8. *Qué otras estrategias ha empleado para enseñar arquitectura de software que puedan ser replicables por otros docentes en otras universidades?* Mi estrategia, a pesar que mi fuerte no es la ingeniería de software, ha sido estar en contacto permanente con las empresas para ver sus necesidades e ir re-acomodando los temas del curso. Sería bueno poder contar con el apoyo de otros docentes de otras universidades. Además tener un respositorio de ejemplos, material para preparar las clases. Ha marcado un hito importante el contacto con Manuel Zapata de quien se ha aprendido muchas cosas de él para luego poder aplicarlas en el curso. Pero me gustaría seguir cualificando en temas de arquitectura de software.

9. *Considera que las estrategias propuestas en la guía ¿permiten acercar la formación en arquitectura de software al mundo real de la industria de software?*

Si. Sin embargo, con el objetivo de acercar a los estudiantes al sector productivo, es necesario apoyarse de profesionales en el tema. En ese sentido, debe existir una relación fuerte entre el docente y expertos en la industria del software.

10. *¿Qué le hace falta a la guía propuesta para que cumpla su propósito de proponer un conjunto de pasos y estrategias de apoyo en el diseño y ejecución de cursos de arquitecturas de software a nivel de pre-grado?*

Considero que a la guía no le falta algo concreto, es una hoja de ruta completa. Lo que haría falta para que el curso de arquitectura se imparta de la mejor forma posible, es contar con aliados del sector productivo para que los casos/problemas/problemas correspondan con los retos de las empresas.

11. *La planificación del curso de arquitectura de software resulta fácil de realizar mediante el uso de la guía (1: totalmente en desacuerdo, 5: totalmente de acuerdo)*

Respuesta: 4

12. *El uso de la guía me permite ahorrar tiempo de planificación? (1: totalmente en desacuerdo, 5: totalmente de acuerdo)*

Respuesta: 4

13. *Al usar la guía siento que soy más productivo para planificar las actividades del curso. (1: totalmente en desacuerdo, 5: totalmente de acuerdo)*

Respuesta: 4

E. Entrevista del estudio de caso de la UNLP

1. *¿Una vez leída la Guía de diseño de cursos de AS, describa qué cambios hizo en su curso a nivel de contenido, objetivos, metodología?*

La guía sirvió para armar una planificación del curso. En especial sirvió para organizar las conferencias de invitados. La organización consistió en definir los temas que se iban a presentar para que estuvieran de acuerdo a los temas que se darían en ese momento en la materia. También se planificó el ejercicio que se iba a desarrollar durante el transcurso del curso. Se acordó entre la cátedra de Patrones de Arquitectura de Software (Argentina) con la cátedra de Ingeniería de Software 2 y Laboratorio (Colombia) en desarrollar una aplicación de E-commerce similar a Mercado Libre+Delivery que llamamos OpenMarket. Lo importante de la planificación consistió en sincronizar los cursos de Colombia y Argentina para que llegaran simultáneamente al tercer sprint. Finalmente se organizaron los temas que se darían en los dos países en especial (Principios Solid, Estilos Arquitectónicos y Microservicios)

Un detalle que no se planificó pero que fue de mucha utilidad fue la posibilidad de grabar las teorías. Esto permitió que los alumnos de los dos países pudieran acceder a los contenidos teóricos en cualquier momento. Estas grabaciones también sirvieron para las charlas de expertos invitados.

2. *¿La planificación de su curso de AS mediante la guía fue de utilidad, vale la pena utilizarla, por ejemplo, le ahorró tiempo y esfuerzo, o por el contrario le agregó más trabajo?*

La guía sirvió mucho para adelantar las discusiones al principio de año. Luego lo planificado se pudo llevar a cabo a pesar de las dificultades que enumero:

- Alumnos con base distinta en los do países: Los alumnos colombianos tenían menos conocimientos iniciales, pero tuvieron más horas de dedicación. Con esto llegaron bien armados al tercer Sprint.
- Diferencia horaria y muchas materias: La diferencia horaria no era muy grande (2 hs.) pero los alumnos tenían muchas materias en simultáneo. Por esto, costaba organizar reuniones técnicas o de organización entre miembros de distintos países.
- Problemas con el laboratorio: Hubo inconvenientes con el Laboratorio en Colombia en

los 2 primeros meses y se tuvo que cambiar el profesor inicial. Luego con el aporte de Santiago Hyun y Julio Hurtado se pudo orientar bien el Laboratorio y se resolvieron los problemas iniciales.

3. *¿La Guía a través de las pasos que plantea para diseñar un curso de AS, es fácil de seguirla, tiene pasos complejos de entender?*

La planificación fue muy sencilla. Primero se escribió un borrador y con el acompañamiento de Libardo Pantoja Yepez se organizó el curso.

4. *¿Acorde a la Guía de Diseño de Cursos de AS, qué patrones de formación decidió aplicar en su curso y por qué?*

Se utilizaron principalmente dos patrones de formación:

- **Proyecto Largo**. Lo más real posible con muchos requisitos, con toma de decisiones de Arquitectura. Este patrón se ejecutó 2 veses en la cátedra de Argentina. Una para desarrollar una versión monolítica de la aplicación de OpenMarket y otra para desarrollar la misma aplicación utilizando microservicios.
- **Conferencia de Expertos invitados**. Los invitados son arquitectos reconocidos en diferentes empresas del sector de desarrollo de software. Los temas que se trataron tenían nexo con el tema que se estaba dando en ese momento del curso. Este patrón se aplicó 3 veces durante el transcurso del curso. Los invitados fueron:
  - Leandro Quiroga (Arquitecto del Proyecto E-Sidif del ministerio de Economía de Argentina)
  - Joaquin Alem y Leandro Zoi (Lider de proyecto y Arquitecto de Mercado Libre Argentina)
  - Santiago Urrizola (Arquitecto de FluxIt - Argentina)

5. *¿Explique cómo implementó (en qué momento del semestre, con qué recursos humanos y técnicos) los patrones de formación?*

Los cursos (Argentina-Colombia) se dictaron durante el primer semestre de 2023, desde Marzo a Julio. El grupo de Argentina estaba formado por: Profesor: Luis Mariano Bibbo. Colaboradores: Marcelo Barreto (especialista en Dev-Ops) y Agustín Ortú (Especialista en diseño con Objetos y Java) Los colaboradores participaban especialmente en el seguimiento del proyecto a los grupos de alumnos resolviendo las dudas técnicas. Marcelo Barreto dictó una charla de Docker y Docker-compose durante el curso.

6. *Qué impacto cree que tuvieron estos patrones en el desarrollo de habilidades de sus estudiantes a nivel de creación, documentación y evaluación de la AS?*

La aplicación de los patrones de formación fueron muy útiles. A continuación se detalla las

ventajas de su aplicación:

- Proyecto Largo. Permitió organizar las entregas de los productos desarrollados por los alumnos lo que permitió: Organizar la evaluación, ejecutar los Test de las aplicaciones desarrolladas, Evaluar la documentación, Ejecutar las colecciones de postman para probar los módulos)
- Conferencia de expertos invitados. Permitió transmitir la utilidad de los temas tratados. Los alumnos pudieron ver casos reales aplicados en la industria. Esto aportó: Motivación y cercanía al mundo de la industria del software.

7. Qué dificultades tuvo aplicando los patrones de formación durante el desarrollo del curso?

- Una de las principales dificultades fue la falta de tiempo en general que tienen los estudiantes en los últimos años de la carrera. En el caso de Argentina los alumnos en los últimos años están trabajando y tienen en general poco tiempo para dedicarse a la facultad. En el caso de los alumnos de Colombia, en el semestre tienen muchas materias que atender.
- Alumnos con base distinta en los 2 países: Los alumnos colombianos tenían menos conocimientos iniciales, pero tuvieron más horas de dedicación. Con esto llegaron bien armados al tercer Sprint.
- Diferencia horaria y muchas materias: La diferencia horaria no era muy grande (2 hs.) pero los alumnos tenían muchas materias en simultáneo. Por esto, costaba organizar reuniones técnicas o de organización entre miembros de distintos países.
- Problemas con el laboratorio: Hubo inconvenientes con el Laboratorio en Colombia en los dos primeros meses y se tuvo que cambiar el profesor inicial. Luego con el aporte de Santiago Hyun y Julio Hurtado se pudo orientar bien el Laboratorio y se resolvieron los problemas iniciales.

8. ¿Considera que los patrones de formación que empleó, permiten acercar la formación en arquitectura de software al mundo real de la industria de software?

Si, especialmente por las conferencias de expertos invitados de la industria que trajeron casos reales donde se aplicaban en desarrollos en la industria los conceptos explicados en la materia.

9. ¿Qué le hace falta a la Guía propuesta para que cumpla su propósito de proponer un conjunto de pasos y estrategias de apoyo en el diseño y ejecución de cursos de arquitecturas de software a nivel de pre-grado?

Ejemplos, Estrategias de evaluación, Tutoriales

F. Entrevista del estudio de caso de la Unimayor

A continuación la transcripción de la entrevista realizada a la profesora María Isabel Bastidas, docente del curso de Diseño y Arquitectura de Software de la Institución Universitaria Mayor del Cauca (Popayán), en el período académico 2023.2.

1. *¿Una vez leída la Guía de diseño de cursos de AS, describa qué cambios hizo en su curso a nivel de contenido, objetivos, metodología?*

Un primer cambio fue el orden en que se impartieron los temas. Acorde a la guía se escogió un orden distinto y más organizado. Por ejemplo, originalmente el tema de los estilos de arquitectura estaba en el último capítulo. También modifiqué la metodología ya que aplicamos varios patrones de formación que hicieron el curso más práctico.

2. *¿La planificación de su curso de AS mediante la guía fue de utilidad, vale la pena utilizarla, por ejemplo, le ahorró tiempo y esfuerzo, o por el contrario le agregó más trabajo?*

En mi caso que era docente que impartía el curso por primera vez, la guía me ahorro bastante tiempo y esfuerzo para planificar mi curso.

3. *¿La Guía a través de las pasos que plantea para diseñar un curso de AS, es fácil de seguirla, tiene pasos complejos de entender?*

No para nada, la guía es muy fácil de entender y aplicar. Los cinco pasos que plantea y sus actividades son muy comprensibles.

4. *¿Acorde a la Guía de Diseño de Cursos de AS, qué patrones de formación decidió aplicar en su curso y por qué?*

De la lista de siete patrones de formación que presenta la guía elegí cuatro:

1. Miniproyectos.
2. Proyecto grande.
3. Casos reales con invitados de la industria.
4. Resolución de problemas de arquitectura de software con katas.

No elegí más porque el escaso tiempo del curso, tan solo son tres horas de clase semanales.

5. *¿Explique cómo implementó (en qué momento del semestre, con qué recursos humanos y técnicos) los patrones de formación?*

El patrón de formación de miniproyectos lo implementé enseñando cada patrón de arquitectura en una semana. En la sesión de dos horas del lunes les di la teoría y un ejemplo en código java. El dia martes en la sesión de una hora los estudiantes trabajaron en una extensión de ese proyecto agregando una funcionalidad al ejemplo. Los ejemplos fueron tomados del material de ayuda que Usted me facilitó.

El patrón de formación de proyectos largos fue implementado asignando los requisitos de un proyecto a cada equipo. Se formaron 9 equipos de tres estudiantes cada uno. Este proyecto ya venían trabajando los estudiantes desde el semestre anterior y tenían hecha la ingeniería de requisitos y modelado una versión monolítica con C4. A cada equipo se le asignó un estilo de arquitectura como requisito y ellos eligieron la tecnología. Las tecnologías que usaron fueron variadas: php, java, .NET, python, javascript y flutter.

El patrón de casos reales con invitados de la industria, se planearon tres charlas, pero finalmente pudimos orientar dos. La primera la realizó el ingeniero Santiago Hyun quien mostró un caso de diseño de una empresa de productos fabricados con el gusano de seda. La segunda, fue orientada por el ingeniero Nelson Chantre quien mostró cómo es el proceso de desarrollo de la empresa Home Center.

Finalmente el patrón de katas de arquitectura se realizó haciendo ejercicios previos de unas katas como trabajo individual a partir de un video realizado por el ingeniero Libardo. Luego, al final de semestre se realizó un kata de arquitectura en clase invitando a dos ingenieros de Unicauca y dos arquitectos de la industria quienes evaluaron las katas de los equipos y dieron su realimentación.

6. *Qué impacto cree que tuvieron estos patrones en el desarrollo de habilidades de sus estudiantes a nivel de creación, documentación y evaluación de la AS?*

Siento que los cuatro patrones influyeron mucho en las habilidades de AS en los estudiantes porque permitieron hacer cosas prácticas y tener un punto de encuentro con representantes de la industria.

7. *¿Qué dificultades tuvo aplicando los patrones de formación durante el desarrollo del curso?*

La principal dificultad es el factor tiempo. Tiempo porque soy ocasional, trabajo en casa y no puedo dedicarme mucho tiempo a la asignatura. Además, el tiempo de los invitados a la clase tambien es complejo de planificar pues las clases son en la noche y muchos despues del horario de trabajo les queda complicado asistir a actividades planeadas para la asignatura. Finalmente, la asignatura tenía dos horas los lunes y durante el semestre hubo muchos

festivos, lo cual hizo complejo impartir las actividades y temas planificadas para el curso.

8. *¿Considera que los patrones de formación que empleó, permiten acercar la formación en arquitectura de software al mundo real de la industria de software?*

Definitivamente si. Todos los patrones de formación tenían el propósito de acercar al estudiante al mundo de la industria de software.

9. *¿Qué le hace falta a la Guía propuesta para que cumpla su propósito de proponer un conjunto de pasos y estrategias de apoyo en el diseño y ejecución de cursos de arquitecturas de software a nivel de pre-grado?*

Yo pienso que la guía está bastante bien, a mi me ayudó mucho. No le agregaría ni quitaría nada.

G. Evaluación de la validez del experimento

Validez de conclusión. Esta validez se refiere a la relación entre el tratamiento y el resultado. Queremos asegurarnos de que existe una relación estadística, es decir, con una significación determinada. Esta validez está relacionada con la fiabilidad del análisis cualitativo. Para ello tendremos en cuenta, la elección de pruebas estadísticas, la elección del tamaño de las muestras, el cuidado en la ejecución y la medición del experimento.

Las amenazas que debemos evitar para favorecer la validez de la conclusión son:

- *Bajo poder estadístico.* La potencia de una prueba estadística es la capacidad de la prueba para revelar un patrón verdadero en los datos. Si la potencia es baja, existe un alto riesgo de que se extraiga una conclusión errónea. Usaremos la evaluación de t-student que es para pocos datos.
- *Vulneración de los supuestos de las pruebas estadísticas.* Algunas pruebas se basan en supuestos como, por ejemplo, la distribución normal y la independencia de las muestras. La violación de estos supuestos puede llevar a conclusiones erróneas. Sabemos que la prueba óptima para este experimento es t-student.
- *La pesca.* Esta amenaza contiene dos partes separadas. La búsqueda o "pesca" de un resultado específico es una amenaza, puesto que los análisis ya no son independientes y los investigadores pueden influir en el resultado buscando un resultado específico. En nuestro caso los grupos estuvieron separados para que no busquen un resultado.
- *Fiabilidad de las mediciones.* La validez de un experimento depende en gran medida de la fiabilidad de las mediciones. Ésta, a su vez, puede depender de muchos factores diferentes, como una mala formulación de las preguntas, una instrumentación deficiente o una mala configuración de los instrumentos. El principio es que cuando se mide un fenómeno dos veces, el resultado será el mismo. En otras palabras, las medidas objetivas, que pueden repetirse con el mismo resultado, son más fiables que las subjetivas. En nuestro caso hemos diseñado con cuidado las preguntas e instrumentos.
- *Fiabilidad de la aplicación del tratamiento.* La implementación del tratamiento significa la aplicación de los tratamientos a los sujetos. Existe el riesgo de que la aplicación no

sea similar entre las distintas personas que aplican el tratamiento o entre las distintas ocasiones. Por lo tanto, la aplicación debe ser lo más estándar posible en diferentes sujetos y ocasiones. Vamos a aplicar los tratamientos igual para los dos grupos.

- *Irrelevancias aleatorias en el entorno experimental.* Elementos ajenos al entorno experimental pueden perturbar los resultados, como el ruido fuera de la sala o una interrupción repentina del experimento. En nuestro caso, los sujetos estuvieron en una sala sin ruido.

Validez interna. Si se observa una relación entre el tratamiento y el resultado, debemos asegurarnos de que se trata de una relación causal y de que no es el resultado de un factor sobre el que no tenemos control o que no hemos medido. En otras palabras, que el tratamiento causa el resultado (el efecto). Los factores que influyen en la validez interna son cómo se seleccionan los sujetos y se dividen en diferentes clases, cómo se trata y compensa a los sujetos durante el experimento, si se producen acontecimientos especiales durante el experimento, etc. Todos estos factores pueden hacer que el experimento muestre un comportamiento que no se deba al tratamiento sino al factor perturbador.

Las amenazas que evitamos para favorecer la validez de la conclusión son:

- *Maduración.* Es el efecto de que los sujetos reaccionan de forma diferente a medida que pasa el tiempo. Por ejemplo, cuando los sujetos se ven afectados negativamente (cansancio o aburrimiento) durante el experimento, o positivamente (aprendizaje) durante el transcurso del experimento. Para evitar la situación anterior, hicimos pausas activas y dimos refrigerios para evitar el cansancio.
- *Instrumentación.* Es el efecto causado por los artefactos utilizados para la ejecución del experimento, como los formularios de recogida de datos, el documento que debe inspeccionarse en un experimento, etc. Si éstos están mal diseñados, el experimento se ve afectado negativamente. Para ello, hemos verificado los instrumentos.
- *Selección.* Es el efecto de la variación natural en el rendimiento humano. Dependiendo de cómo se seleccionen los sujetos de un grupo mayor, los efectos de selección pueden variar. Además, el efecto de dejar participar a voluntarios en un experimento puede influir en los resultados. Los voluntarios suelen estar más motivados y ser más aptos para una nueva tarea que el conjunto de la población. De ahí que el grupo seleccionado no sea representativo de toda la población. La selección de sujetos son docentes del sur occidenete colombiano, de distintas universidades y enfoques de cursos.
- *Mortalidad.* Este efecto se debe a los distintos tipos de personas que abandonan el experimento. Es importante caracterizar los abandonos para comprobar si son representativos de la muestra total. Si los sujetos de una categoría específica abandonan, por ejemplo, todos los profesores con la Guia del estado del arte, la validez del experimento se ve muy afectada. Las dos guias tuvieron cargas similares para evitar que

algún grupo abandone el experimento.

- *Difusión o imitación de tratamientos.* Este efecto se produce cuando un grupo de control se entera del tratamiento por el grupo del estudio experimental o intenta imitar el comportamiento del grupo del estudio. Par evitar esta amenaza separamos físicamente los grupos para limitar la visibilidad.
- *Igualación compensatoria de los tratamientos.* Si a un grupo de control se le da una compensación por ser un grupo de control, como compensación por no recibir tratamientos, esto puede afectar al resultado del experimento. Si al grupo de control se le enseña otro método nuevo como compensación por no habérsele enseñado el método basado en la perspectiva, su rendimiento puede verse afectado por ese método. En nuetro caso, las dos guías tienen aspectos similares.
- *Desmoralización resentida.* Es lo contrario de la amenaza anterior. Un sujeto que recibe tratamientos menos deseables puede darse por vencido y no rendir tan bien como lo hace generalmente. El grupo que utiliza el método tradicional no está motivado para hacer un buen trabajo, mientras que aprender algo nuevo inspira al grupo que utiliza el nuevo método. Los grupos no supieron cual es al Guía les tocó trabajar.

Validez de constructo. Esta validez se refiere a la relación entre la teoría y la observación. Si la relación entre causa y efecto es causal, debemos asegurarnos de dos cosas: (1) que el tratamiento refleja bien el constructo de la causa y (2) que el resultado refleja bien el constructo del efecto. Las amenazas a la validez de constructo se refieren al grado en que el escenario del experimento refleja realmente el constructo estudiado.

Las amenazas que debemos evitar para favorecer la validez interna son:

- *Explicación preoperativa inadecuada de los constructos.* Esta amenaza, a pesar de su extenso título, es bastante simple. Significa que los constructos no están suficientemente definidos, antes de que se traduzcan en medidas o tratamientos. La teoría no es lo suficientemente clara y, por tanto, el experimento tampoco puede serlo. Para evitar esta amenaza los constructos fueron pensados detenidamente con sus respectivas escalas.
- *Temor a la evaluación.* Algunas personas tienen miedo a ser evaluadas. Una forma de tendencia humana es intentar parecer mejor cuando se es evaluado, lo que se confunde con el resultado del experimento. En nuestro caso, hicimos una introducción mencionándoles que no iban a ser evaluados.

Validez externa. La validez externa tiene que ver con la generalización. Si existe una relación causal entre el constructo de la causa y el efecto, ¿puede generalizarse el resultado del estudio fuera del ámbito de nuestro estudio? ¿Existe una relación entre el tratamiento y el resultado? La validez externa se ve afectada por el diseño del experimento elegido, pero también por los objetos del experimento y los sujetos elegidos. Hay tres riesgos principales: tener como sujetos

a participantes equivocados, realizar el experimento en el entorno equivocado y llevarlo a cabo con una sincronización que afecte a los resultados.

Las amenazas que evitamos para favorecer la validez externa fueron:

- *Interacción de la selección y el tratamiento*. Se trata de un efecto de tener una población de sujetos no representativa de la población a la que queremos generalizar, es decir, que en el experimento participen las personas equivocadas. En este caso, hemos elegido las personas adecuadas, profesores de AS.
- *Interacción del entorno y el tratamiento*. Es el efecto de que el entorno o el material del experimento no sean representativos. El entorno fue una sala de cómputo, con la que los docentes están acostumbrados a interactuar.
- *Interacción de la historia y el tratamiento*. Es el efecto de que el experimento se realice en un momento o día especial que afecta a los resultados. Las amenazas a la validez externa la redujimos haciendo que el entorno experimental sea lo más realista posible.

H. Script R para análisis de datos del experimento

```
historial <- read.csv("./data.csv")
View(historial)
library(ggplot2)
library(dplyr)


#Verificar que los datos siguen una distribucion normal
#y no tenga datos atipicos

ggplot(historial, aes(Minutes, fill=Group, color=Group))+
geom_density(alpha=0.2)+
xlim(100, 200)

#Generamos un grafico qq donde se dibuja una linea
#y si los datos estan cerca a la recta, tienen una
#distribucion normal

qqnorm(historial$Minutes)
qqline(historial$Minutes, col="red")

#Shapiro-Wilk test dice con certeza si los datos
#tienen una distribucion normal. Si p-value >0.05
#no se rechaza la hipotesis nula. La hipotesis
#nula dice que la muestra sigue una distribucion normal.

shapiro.test(historial$Minutes)

#Ahora vamos a ver si hay datos atipicos en la muestra
#con un grafico de cajas. No deben aparecer datos
```

```
#arriba de los limites superiores ni inferiores

ggplot(historial, aes(Group, Minutes, fill=Group, color=Group ))+
geom_boxplot(alph=0.4)+
theme(legend.position = "none")


#Ahora, vamos a analizar los promedios de las muestras (Minutes)
historial %>%
group_by(Group) %>%
summarize (total = n(),
Minutespromedio = mean(Minutes))

#Vemos si las varianzas de los groups son iguales.
#Arroja un p-valor de 0.1805 que es mayor a 0.05,
#por lo tanto, no se rechaza la hipotesis nula,
#eso implica que las varianzas entre los dos Groups es la misma

var.test(Minutes ~ Group, data=historial)

#Finalmente hacemos la prueba de t-student para reforzar
#lo que hemos visto graficamente. Prueba t, analiza la
#variable Minutes, diferenciandola del tipo de Group usado
#asumiendo que las dos varianzas son iguales

t.test(Minutes ~ Group, data=historial, var.equal = T)
```

I. Proyecto de clase Open Market

I.1. Descripción del Problema

La empresa Colombo-Argentina Digital Asociados SA, está interesada en lanzar al mercado una plataforma tecnológica para hacer mercadeo por internet y ser competidores de plataformas conocidas como Mercado Libre. Esta empresa requiere desarrollar una aplicación web que permita a sus usuarios comprar, vender y distribuir productos en línea por medio de una aplicación web de forma segura, rápida y fácil de usar. A continuación se detallan los requisitos funcionales y no funcionales de esta aplicación.

I.2. Requisitos funcionales

1. Datos del sistema
    - *a*) Loguear un usuario con uno de los roles del sistema.
    - *b*) La creación de usuarios se puede hacer por script de BD. Se podrá utilizar productos de gestión de datos tipo Liquibase de forma que la DB evolucione al mismo tiempo que el software el schema sea versionado y se pueda inicializar con datos predefinidos.
    - *c*) Los datos de negocio (por ejemplo las comisiones) deben estar adecuadamente abstraídos y quemados en el mismo sistema.
    - *d*) Rol vendedor: dar de alta, baja y modificación de productos (CRUD)
    - *e*) Al agregar productos ofrecidos.
    - *f*) Debe tener al menos nombre, descripción, precio, categoría y ubicación, usando la latitud y longitud.
    - *g*) Modificar stock de productos ofrecidos
    - *h*) Dar de baja productos ofrecidos.
    - *i*) Suspender publicación.

j) Al publicar un producto, además del precio de venta, se puede agregar también un precio mínimo y un precio máximo.

2. Rol usuario: Búsqueda

   a) Listar productos.

   b) Búsqueda de productos. Las búsquedas son por palabras (la búsqueda se realizará usando el título y descripción).

   c) Se pueden aplicar filtros: por rango de precios, por categoría y por ubicación.

   d) Búsqueda por distancia. El comprador tiene una geolocalización, y se podrá filtrar los productos por la distancia (lineal) entre las ubicaciones de vendedor y comprador.

3. Rol usuario registrado Datos personales

   a) Datos del usuario que deben considerarse en el script: username, contraseña, nombre, apellido, email, teléfono, rol, tipo de facturación, fecha de nacimiento, puntuación.

4. Rol usuario registrado: Realizar Compra

   a) Comprar productos. Las compras se administran directamente sin la necesidad de un carritoun de compras. Rol usuario Registrado: Billetera virtual

   b) Al realizar la compra, el comprador se transfiere a un sistema de pagos externo (puede ser simulado) quién le confirma el pago realizado. Si es exitoso se le confirma a las partes y se debe actualizar el stock.

   c) El comprador al recibir el producto confirma el recibido (y puntúa de 1 a 5 el comprador) y si está correcto, a través del sistema de pagos se le transfiere el valor de la venta menos la comisión al vendedor.

I.3. Consideraciones de la arquitectura de software

La aplicación a desarrollar es una sola y permitirá a los vendedores publicar productos, a los compradores buscarlos y comprarlos. Esto implica que se debe plantear una arquitectura de software que soporte la escalabilidad (a medida que se sumen usuarios compradores y vendedores) y la seguridad de los datos (el sistema, a pesar que es uno sólo, debe aislar la información de usuario, importante definir mecanismo de autenticación y autorización seguros).

I.4. Trabajo en equipo

Se debe trabajar en grupos de cuatro o máximo cinco personas (En caso de que sean cinco personas debe agregarse la funcionalidad de un carrito de compras). Es importante que cada equipo trabaje de forma colaborativa, donde haya buenas relaciones entre todos sus integrantes. El equipo debe buscar herramientas que favorezcan la comunicación durante el proyecto y la coordinación de tareas. Metodología El equipo deberá trabajar con el marco de gestión de Scrum, en tres sprints, definiendo los roles Scrum Master, Product Owner y Equipo de Trabajo. Debe hacer seguimiento períodico al trabajo (cada sesión) y se deben seguir las ceremonias de planificación, trabajo diario, revisión y retrospectiva (debe haber evidencia). Para el seguimiento se debe utilizar una herramienta de apoyo como Trello o Jira. Respecto a las prácticas de desarrollo se deben usar las prácticas de XP. Para el trabajo en equipo la propiedad colectiva de código

I.5. Entregables

Para los tres cortes se debe entregar:

- Especificación de requisitos NO funcionales que se resuelven con este proyecto.
- ¿Qué tecnologías ayudan a cumplir con los requisitos no funcionales especificados?
- Documentación de la arquitectura ya sea con el modelo C4 o UML.
- Imagen de docker publicada en GitLab.

J. Proyecto de clase Open Market para los estudiantes de Unicauca

El objetivo de este documento es definir los entregables para cada corte del proyecto de clase Open Market de Ingeniería de Software II y Laboratorio de Ingeniería de Software II durante este periodo académico 2023.1. Se toma como referencia el documento de Open Market que contiene todos los requisitos, pero se acota los requisitos a entregar debido al tiempo disponible.

El proyecto Open Market lo deben trabajar en grupos de cuatro estudiantes. Es importante que cada equipo trabaje de forma colaborativa, donde haya buenas relaciones entre todos sus integrantes, donde existan tareas individuales y metas comunes. El equipo debe buscar herramientas que favorezcan la comunicación durante el proyecto y la coordinación de tareas. Se recomiendan: taiga.io, trello o jira.

J.1. Entregables

Para los dos primeros cortes se debe entregar (el tercer corte es distinto es una experiencia de desarrollo global):

1. Especificación de requisitos funcionales (de los más importantes, de los que generen valor al cliente) ya sea mediante escenarios de casos de uso, o Historias de Usuario y criterios de aceptación. Se recomienda antes de implementar una funcionalidad hacer prototipos de la interfaz de usuario (al estilo wireframes) y aplicar algún método de validación de usabilidad como thinking aloud.
2. Especificación de cualidades del software mediante escenarios de calidad.
3. Documentación de la arquitectura combinando vistas del modelo C4 y de UML:Vistas de la arquitectura, Vista C4, Vista de módulos, Vista de C&C, Vista de Instalación, Rationale (Justificación de las decisiones de diseño).
4. Prototipo funcional en java.
5. Documentar las decisiones de diseño, que tácticas y qué patrones de diseño se aplicaron

en la implementación (en un archivo pdf).

Para el **primer corte** del curso se debe entregar la primera iteración del proyecto que consiste en hacer una implementación mediante una arquitectura monolítica en capas.

Para el **segundo corte** del curso se debe entregar la segunda iteración que consiste en otra implementación mediante una arquitectura microkernel y por eventos.

Para el tercer corte se debe entregar una tercera iteración que consiste en una implementación utilizando una arquitectura hexagonal y microservicios. Esta iteración es diferente a los dos anteriores, pues se hará una experiencia de desarrollo global con estudiantes de arquitectura de software de la Universidad Nacional de la Plata - Argentina. Cada equipo de desarrollo se unirá a otro equipo de la UNLP y les ayudarán implementando un microservicio. Los estudiantes de la UNLP serán los encargados de pasar el diseño del microservicio a los estudiantes de Unicauca y guiarlos en la integración y despliegue mediante trabajo colaborativo.

J.2. Rúbrica de evaluación de cada corte

Los entregables de cada corte, tanto para la teoría como para el laboratorio, debe cumplir la siguiente rúbrica de evaluación (los elementos resaltados en color gris son obligatorios, si falla alguno, la nota es menor a 3.0):

- *Nivel 1: Insuficiente*. Se omiten todos los aspectos resaltados en color gris, casi no se logran las marcas. Nota: 0 – 1.0
- *Nivel 2: No alcanzado*. Se omite alguno de los aspectos resaltados en color gris, se alcanzan varias marcas. Nota: 1.1 – 2.9
- *Nivel 3: Logrado*. Se alcanzan todos los aspectos resaltados en color gris pero los entregables tienen muchas cosas por mejorar. Nota: 3.0 – 4.0
- *Nivel 4: Satisfactorio*. Se alcanza de manera avanzada las características. Cumple todas o la gran mayoría de marcas. Nota: 4.1 – 5.0

Para el laboratorio se aplicará la siguiente rúbrica de evaluación:

Tabla A1.: Rúbrica de evaluación para el laboratorio

Característica	Descripción	Obligatorio
Principios SOLID	Cumple con los principios SOLID.	Si
Patrón arquitectónico	Se aplica correctamente el patrón arquitectónico.	Si
Patrones de diseño	Se aplican patrones de diseño de forma correcta.	Si
Requisitos funcionales	Satisface los requisitos funcionales.	Si
Requisitos no funcionales	Satisface los requisitos no funcionales.	Si
Código fuente	Indentado perfectamente. Documentando con java doc Sigue las convenciones de java para nombrar paquetes, clases, métodos, constantes. Commits de todos los participantes en el repositorio git. Obedece al diseño propuesto.	No No No No Si
Pruebas unitarias	Corre satisfactoriamente las pruebas unitarias.	Si

Para la teoría se aplicará la siguiente rúbrica de evaluación:

Tabla A2.: Rúbrica de evaluación para la teoría

Característica	Descripción	Obligatorio
Requisitos funcionales	Se especifican correctamente los requisitos funcionales, ya sea como historias de usuarios y criterios de aceptación ó casos de uso.	No
Requisitos no funcionales	Se especifican correctamente mediante escenarios de calidad.	Si
Validación de los prototipos de la interfaz de usuario	Se valida la usabilidad de los prototipos de la interfaz mediante la técnica thinking aloud.	No
Decisiones de arquitectura	El tipo de aplicación es justificado: web, web single page, escritorio, móvil, híbrida (web y móvil). El estilo arquitectónico es justificado acorde a los requisitos de la aplicación. Tecnología elegida, se responden las preguntas: ¿Qué tecnologías ayudan a implementar los estilos arquitecturales seleccionados? ¿Qué tecnologías ayudan a implementar el tipo de aplicación seleccionada? ¿Qué tecnologías ayudan a cumplir con los requisitos no funcionales especificados?	✓ Si No
Gestión del proyecto	Se usa Scrum y se evidencia: backlog, pila del sprint y el burn-down chart.	Si
Documentación de la arquitectura	Se documenta adecuadamente la arquitectura de la aplicación vistas C4 y UML.	Si
Diseño detallado	El diseño propuesto involucra patrones de diseño y se utilizan correctamente.	Si

J.3. Requerimientos funcionales

A continuación, se especifican algunos requisitos funcionales del proyecto Open Market a manera de historias épicas. De cada historia épica pueden salir varias historias de usuario. La historia épica puede ser tomada como el CRUD (Create, Read, Update, Delete) completo de una funcionalidad grande; luego se deben especificar historias de usuario pequeñas por

cada épica para agregar, listar, eliminar, editar, etc.

Es importante que las historias épicas estén ordenadas por valor, es decir, de la más prioritaria a la menos prioritaria para el cliente. Cada equipo de trabajo debe especificar las historias épicas faltantes a este listado.

Tabla A3.: Historias épicas

Código	Historia épica	Contexto
HE01	**Yo como** vendedor en la plataforma **necesito** gestionar productos al sistema (CRUD) **para** posteriormente otros usuarios los puedan comprar en línea.	Un producto debe tener al menos nombre, descripción, foto(s), precio, categoría y ubicación (usando la latitud y longitud).
HE02	**Yo como** usuario anónimo y usuario registrado **necesito** buscar productos en el sistema por diferentes criterios **para** detallar los resultados y ver los datos específicos de cada producto.	Las búsquedas se aplican en el título y en la descripción. Se pueden aplicar filtros: por rango de precios, por categoría y por ubicación.
HE03	**Yo como** usuario registrado de Open Market **necesito** realizar compras incluyendo realizar el pago **para** que me envíen el producto a la casa.	
HE05	**Yo como** usuario con alguno de los roles del sistema **necesito** iniciar sesión en el sistema **para** acceder a las funcionalidades o servicios que ofrece la plataforma.	
HE06	**Yo como** usuario anónimo **necesito** registrarme en la plataforma Yo como usuario anónimo necesito registrarme en la plataforma para comprar/vender productos en la plataforma comprar/vender productos en la plataforma	

Para el primer y segundo corte de la asignatura se deben trabajar los siguientes historias de usuario como mínimo (en los dos cortes de trabajan los mismos requisitos funcionales, pero cambia el estilo de arquitectura):

Tabla A4.: Historias de usuario

HE	HU	Historia de usuario	Contexto
HE01	HU01	**Yo como** usuario registrado en la plataforma (vendedor) **necesito** agregar un nuevo producto al sistema **para** posteriormente otros usuarios lo puedan comprar en línea.	Un producto debe tener al menos nombre, descripción, foto(s), precio, categoría y ubicación (usando la latitud y longitud).
HE03	HU02	**Yo como** usuario registrado de Open Market **necesito** (comprador) agregar un producto al carrito de compras **para** posteriormente realizar el pago y que me envien el producto a la casa.	Al momento de agregar el producto al carrito se debe solicitar la cantidad a comprar.
HE03	HU03	**Yo como** usuario registrado (comprador) de Open Market **necesito** visualizar los ítems que he agregado al carrito de compras **para** posteriormente realizar el pago y que me envien el producto a la casa.	Al visualizar los ítems del carrito, se debe mostrar el valor total de la compra. Se pide listar los ítem por página de 5, 10, 20, 50 elementos.
HE04	HU04	**Yo como** entregador o repartidor de productos **necesito** entregar el producto y cambiar el estado a entregado **para** finalizar el proceso de compra	na vez el repartidor entrega el producto, entra al sistema y coloca datos de la entrega: fecha, persona que lo recibió, y el estado del producto cambia a Entregado.

Para el tercer corte se hará la experiencia de desarrollo global con los estudiantes de la UNLP. Solamente se debe entregar el microservicio implementado y probado.

J.4. Valor del proyecto de clase

A continuación los valores del proyecto de clase respecto a las dos asignaturas en cada corte.

Laboratorio de Ingeniería de Software:

- Talleres: 50
- Proyecto de clase: 50

Ingeniería de software II:

- Proyecto de clase: 25
- Talleres: 15
- Parcial: 60

Bibliografía

[1] Zahra Shakeri Hossein Abad, Muneera Bano, and Didar Zowghi. How Much Authenticity Can Be Achieved in Software Engineering Project Based Courses? In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*, ICSE-SEET '19, pages 208–219, Montreal Quebec Canada, 2019. IEEE Press.

[2] Azim Abdool and Akash Pooransingh. An industry-mentored undergraduate software engineering project. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–4, Madrid, Spain, 2014. IEEE.

[3] Pekka Abrahamsson, Muhammad Ali Babar, and Philippe Kruchten. Agility and Architecture: Can They Coexist? *IEEE Software*, 27(2):16–22, 2010.

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

[5] Céline Madeleine Aldenhoven and Ralf Sascha Engelschall. The beauty of software architecture. In *2023 IEEE 20th International Conference on Software Architecture (ICSA)*, pages 117–128, 2023.

[6] Chistopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language: Towns, Buildings, Construction*. Oxford university press, 1977.

[7] Razieh Alidoosti, Patricia Lago, Eltjo Poort, and Maryam Razavian. Ethics-Aware DecidArch Game: Designing a Game to Reflect on Ethical Considerations in Software Architecture Design Decision Making. In *2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C)*, pages 96–100, 2023.

[8] Razieh Alidoosti, Patricia Lago, Eltjo Poort, Maryam Razavian, and Antony Tang. Incorporating Ethical Values into Software Architecture Design Practices. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 124–127, 2022.

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

[10] Matthias Backert, Thomas Blum, Rüdiger Kreuter, Frances Paulisch, and Peter Zimmerer. Software Curriculum @ Siemens — The Architecture of a Training Program for Architects. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–6, Munich, Germany, 2020. IEEE.

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

[12] Paula Bartel, Alexander Bartel, and Georg Hagel. Flipped Teaching in Software Engineering Education.: Results of a Long-Term Study. In *Proceedings of the 5th European Conference on Software Engineering Education*, ECSEE '23, pages 93–101, New York, NY, USA, 2023. Association for Computing Machinery.

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

[14] Frank Buschmann and Kevlin Henney. Architecture and Agility: Married, Divorced, or Just Good Friends? *IEEE Software*, 30(2):80–82, 2013.

[15] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, Chichester, UK, 1996.

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

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

[18] Daniela Castelluccia, Bari Aldo, Giuseppe Visaggio, Bari Aldo, Daniela Castelluccia, and Giuseppe Visaggio. Teaching evidence-based software engineering: learning by a collaborative mapping study of open source software. In *ACM SIGSOFT Software Engineering Notes*, volume 38, pages 1–4, Madrid, Spain, 2013. ACM.

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

[20] Robert Chatley and Tony Field. 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*, pages 117–126, Buenos Aires, 2017. IEEE Press.

[21] Muhammad Aufeef Chauhan, Christian W Probst, and Muhammad Ali Babar. *Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods*, pages 325–351. Springer Singapore, Singapore, 2019.

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

[23] Peter J Clarke, Jairo Pava, Yali Wu, and Tariq M King. Collaborative Web-Based Learning of Testing Tools in SE Courses. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 147–152, New York, NY, USA, 2011. Association for Computing Machinery.

[24] Jane Cleland-Huang, Muhammad Ali Babar, and Mehdi Mirakhorli. An Inverted Classroom Experience: Engaging Students in Architectural Thinking for Agile Projects. In *Companion Proceedings of the 36th International Conference on Software Engineering*, ICSE Companion 2014, pages 364–371, New York, NY, USA, 2014. Association for Computing Machinery.

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

[26] G J Collins. Integrating Industry Seminars within a Software Engineering Module to Enhance Student Motivation. In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1497–1502, Opatija, Croatia, 2020. IEEE.

[27] James Coplien and Douglas Schmidt. *Pattern Languages of Program Design 1st Edition*. Addison-Wesley Professional, 1 edition, 1995.

[28] Cristóbal Costa-Soria and Jennifer Pérez. Teaching Software Architectures and Aspect-Oriented Software Development Using Open-Source Projects. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE '09, page 385, New York, NY, USA, 2009. Association for Computing Machinery.

[29] Patrick de Beer and Samuil Angelov. 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, New York, NY, USA, 2015. Association for Computing Machinery.

[30] Remco C. de Boer, Rik Farenhorst, and Hans van Vliet. A community of learners approach to software architecture education. In *2009 22nd Conference on Software Engineering Education and Training*, pages 190–197, 2009.

[31] Remco C de Boer, Patricia Lago, Roberto Verdecchia, and Philippe Kruchten. DecidArch V2: An Improved Game to Teach Architecture Design Decision Making. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 153–157, 2019.

[32] Remco C De Boer and Hans Van Vliet. On the similarity between requirements and architecture. *Journal of Systems and Software*, 82(3):544–550, 2009.

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

[34] Padmashree Desai, G H Joshi, and M Vijayalaskhmi. A novel approach to carrying out mini project in Computer Science Engineering. In *2012 IEEE International Conference on Engineering Education: Innovative Practices and Future Trends (AICERA)*, pages 1–4, Kottayam, Kerala, India, 2012. IEEE.

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

[36] Felix Dobslaw, Kristian Angelin, Lena-Maria Öberg, and Awais Ahmad. The Gap between Higher Education and the Software Industry — A Case Study on Technology Differences. In *Proceedings of the 5th European Conference on Software Engineering Education*, ECSEE '23, pages 11–21, New York, NY, USA, 2023. Association for Computing Machinery.

[37] Tobias Eigler, Florian Huber, and Georg Hagel. Tool-Based Software Engineering Education for Software Design Patterns and Software Architecture Patterns - a Systematic Literature Review. In *Proceedings of the 5th European Conference on Software Engineering Education*, ECSEE '23, pages 153–161, New York, NY, USA, 2023. Association for Computing Machinery.

[38] Martin Fowler. Who Needs an Architect? *IEEE Software*, 20(5):11–13, 2003.

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

[40] E Gamma, R Helm, R Johnson, and J Vlissides. *Design Patterns: Elements of Reusable Software*. Addison-Wesley Professional, 1 edition, 1996.

[41] David Garlan and Mary Shaw. An introduction to software architecture. Technical Report CMU/SEI-94-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1994.

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

[43] John C Georgas, James D Palmer, and Michael J McCormick. Supporting Software Architecture Learning Using Runtime Visualization. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, pages 101–110, 2016.

[44] Fáber D Giraldo, Sergio F Ochoa, Myriam Herrera, Andrés Neyem, José Luis Arciniegas, Clifton Clunie, Sergio Zapata, and Fulvio Lizano. Applying a distributed CSCL activity for teaching software architecture. In *International Conference on Information Society (i-Society 2011)*, pages 208–214, London, United Kingdom, 2011. IEEE.

[45] Anderson Cavalcante Gonçalves, Valdemar Vicente Graciano Neto, Deller James Ferreira, and Uyara Ferreira Silva. Flipped Classroom Applied to Software Architecture Teaching. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–8, 2020.

[46] Tihana Galinac Grbac, Željka Car, and Marin Vuković. Requirements and Architecture Modeling in Software Engineering Courses. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW '15, New York, NY, USA, 2015. Association for Computing Machinery.

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

[48] Lile P Hattori and Michele Lanza. On the nature of commits. In *2008 23rd IEEE/ACM International Conference on Automated Software Engineering - Workshops*, pages 63–71, 2008.

[49] Mauricio Hidalgo, Hernán Astudillo, and Laura M Castro. Challenges to Use Role Playing in Software Engineering Education: A Rapid Review. In Hector Florez and

Marcelo Leon, editors, *Applied Informatics*, pages 245–260, Cham, 2023. Springer Nature Switzerland.

[50] Rune Hjelsvold and Deepti Mishra. Exploring and Expanding GSE Education with Open Source Software Development. *ACM Transactions on Computing Education*, 19(2), 2019.

[51] Zhewei Hu, Yang Song, and Edward F Gehringer. 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, pages 40–48, New York, NY, USA, 2018. Association for Computing Machinery.

[52] IEEE. IEEE 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems, 2000.

[53] A Jansen and J Bosch. Software Architecture as a Set of Architectural Design Decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 109–120, Pittsburgh, PA, USA, 2005. IEEE.

[54] Stan Jarzabek. Teaching advanced software design in team-based project course. In *2013 26th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 31–40, San Francisco, CA, USA, 2013. IEEE.

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

[56] Lynette Johns-Boast and Shayne Flint. Simulating industry: An innovative software engineering capstone design course. In *2013 IEEE Frontiers in Education Conference (FIE)*, pages 1782–1788, Oklahoma City, OK, USA, 2013. IEEE.

[57] Jeevamol Joy and V. G. Renumol. Activity oriented teaching strategy for software engineering course: An experience report. *Journal of Information Technology Education: Innovations in Practice*, 17:181–200, 2018.

[58] Jayden Khakurel and Jari Porras. The Effect of Real-World Capstone Project in an Acquisition of Soft Skills among Software Engineering Students. In *2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–9, Munich, Germany, 2020. IEEE.

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

[60] Arvind W Kiwelekar. A Software Architecture Teacher's Dilemmas. Technical Report Icsa, University Lonere-402 103, 2021.

[61] Patricia Lago, Jia F. Cai, Philippe Kruchten, Remco C. de Boer, and Roberto Verdecchia. Decidarch: Playing cards as software architects. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, volume 2019-Janua, pages 7815–7824, 2019.

[62] Patricia Lago and Hans Van Vliet. Teaching a Course on Software Architecture. In *18th Conference on Software Engineering Education Training (CSEE&T05)*, pages 35–42, Ottawa, ON, Canada, 2005. IEEE.

[63] Jaejoon Lee, Gerald Kotonya, Jon Whittle, and Christopher Bull. Software Design Studio: A Practical Example. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 389–397, Florence, Italy, 2015. IEEE.

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

[65] Weigang Li. Teaching Reform and Practice of Software Architecture Design Course under the Background of Engineering Education. In *Proceedings of the 2019 International Conference on Advanced Education, Management and Humanities (AEMH 2019)*, volume 352, pages 17–21, Wuhan, China, 2019. Atlantis Press.

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

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

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

[69] Eng Lieh Ouh, Benjamin Kok Siew Gan, and Yunghans Irawan. Did our Course Design on Software Architecture meet our Student's Learning Expectations? In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Uppsala, Sweden, 2020. IEEE.

[70] Adriana Lopes, Igor Steinmacher, and Tayana Conte. UML Acceptance: Analyzing the Students' Perception of UML Diagrams. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, SBES 2019, pages 264–272, New York, NY, USA, 2019. Association for Computing Machinery.

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

[72] Matti Luukkainen, Arto Vihavainen, and Thomas Vikberg. Three Years of Design-Based Research to Reform a Software Engineering Curriculum. In *Proceedings of the 13th Annual Conference on Information Technology Education*, SIGITE '12, pages 209–214, New York, NY, USA, 2012. Association for Computing Machinery.

[73] Christoph Matthies, Ralf Teusner, and Guenter Hesse. Beyond Surveys: Analyzing Software Development Artifacts to Assess Teaching Efforts. In *2018 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, San Jose, CA, USA, 2018. IEEE.

[74] Kun May, Bo Yang, Jin Zhou, Yongzheng Lin, Kun Zhang, and Ziqiang Yu. Outcome-based school-enterprise cooperative software engineering training. In *ACM International Conference Proceeding Series*, pages 15–20, Shanghai China, 2018. Association for Computing MachineryNew YorkNYUnited States.

[75] Andrew Meneely and Samuel Lucidi. Vulnerability of the Day: Concrete Demonstrations for Software Engineering Undergraduates. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1154–1157, San Francisco CA USA, 2013. IEEE Press.

[76] Gerard Meszaros and Jim Doble. A pattern language for pattern writing. *Pattern languages of program design*, 3:529–574, 1998.

[77] Sriram Mohan, Stephen Chenoweth, and Shawn Bohner. Towards a Better Capstone Experience. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, SIGCSE '12, pages 111–116, New York, NY, USA, 2012. Association for Computing Machinery.

[78] Claudia Hidalgo Montenegro and Hernán Astudillo. A role-playing game to teach ATAM (Architecture Trade-off Analysis Method) a simulation tool and case study. In *2014 IEEE ANDESCON*, page 1, 2014.

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

[80] Luis Freddy Muñoz, Julio Ariel Hurtado, and Francisco Alvarez-Rodriguez. Agile architecture in action (AGATA). *Ingeniería y Universidad:*, 22, 2017.

[81] Usman Nasir. Using Architectural Kata in Software Architecture Course: An Experience Report. In *Proceedings of the 5th European Conference on Software Engineering Education*, ECSEE '23, pages 215–219, New York, NY, USA, 2023. Association for Computing Machinery.

[82] Brauner R. N. Oliveira and Elisa Yumi Nakagawa. Towards a case-based learning approach to support software architecture education. *ArXiv*, abs/2210.04794, 2022.

[83] J L Ortega-Arjona. *PATTERNS FOR PARALLEL SOFTWARE DESIGN*. Wiley India Pvt. Limited, 2010.

[84] Eng Lieh Ouh and Yunghans Irawan. 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, pages 457–463, New York, NY, USA, 2019. Association for Computing Machinery.

[85] Nicolás Martín Paez. A Flipped Classroom Experience Teaching Software Engineering. In *2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)*, pages 16–20, Buenos Aires, Argentina, 2017. IEEE.

[86] Maria Palacin-Silva, Jayden Khakurel, Ari Happonen, Timo Hynninen, and Jari Porras. Infusing Design Thinking into a Software Engineering Capstone Course. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, pages 212–221, Savannah, Georgia, USA, 2017. IEEE.

[87] W. Libardo Pantoja and Julio Ariel Hurtado. Dificultades y retos de la formación de arquitectos de software en programas de pregrado: Causas y efectos. In *Tercer Simposio Doctoral del Doctorado en Ciencias de la Electrónica*, Popayán, Colombia, 2021.

[88] W. Libardo Pantoja, Julio Ariel Hurtado, Bandi Ajay, and Arvind W Kiwelekar. Training Software Architects Suiting Software Industry Needs: A Literature Review. *Education and Information Technologies*, 2023.

[89] W. Libardo Pantoja, Julio Ariel Hurtado, Luis Mariano Bibbó, Alejandro Fernández, and Bandi Ajay. Towards a Software Architecture Training Pattern Language. In *Pattern Languages Of Programs Conference PLOP2023*, Illinois, USA, 2023.

[90] W. Libardo Pantoja, Julio Ariel Hurtado, Andrés Solano, and Bandi Ajay. Visión de las competencias en arquitectura de software integrando las perspectivas de la industria y la academia. In *Sexto Congreso Andino de computación, informática educación*, page 10, Pasto, Colombia, 2023.

[91] Wilson Libardo Pantoja, Julio Ariel Hurtado, and Arvind W Kiwelekar. Aligning Software Architecture Training with Software Industry Requirements. *International Journal of Software Engineering and Knowledge Engineering*, 33:435–460, 2023.

[92] Cécile Péraire. Dual-Track Agile in Software Engineering Education. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 38–49, Montreal, QC, Canada, 2019. IEEE.

[93] Juanan Pereira. Leveraging Final Degree Projects for Open Source Software Contributions. *Electronics*, 10(10), 2021.

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

[95] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic Mapping Studies in Software Engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, pages 68–77, Swindon, GBR, 2008. BCS Learning & Development Ltd.

[96] Ravi Shankar Pillutla and Anuradha Alladi. Methodology to bridge the gaps between engineering education and the industry requirements. In *Eurocon 2013*, pages 926–932, Zagreb, Croatia, 2013. IEEE.

[97] Gustavo Pinto, Clarice Ferreira, Cleice Souza, Igor Steinmacher, and Paulo Meirelles. 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*, pages 147–157, Montreal Quebec Canada, 2019. IEEE.

[98] Carlos Portela, Alexandre Vasconcelos, Sandro Oliveira, and Maurício Souza. The Use of Industry Training Strategies in a Software Engineering Course: An Experience Report. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE&T)*, pages 29–36, Savannah, Georgia, 2017. IEEE.

[99] N Pratheesh and T Devi. Assessment of student's learning style and engagement in traditional based software engineering education. In *2013 International Conference on Intelligent Interactive Systems and Assistive Technologies*, pages 25–31, Coimbatore, India, 2013. IEEE.

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

[101] Linda Rising, editor. *Design Patterns in Communications Software*. Cambridge University Press, USA, 2001.

[102] E. Ronchieri and M. Canaparo. Metrics for software reliability: A systematic mapping study. *Journal of Integrated Design and Process Science*, 22(2):5–25, 2019.

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

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

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

[106] Douglas C. Schmidt and Zach McCormick. Producing and delivering a MOOC on pattern-oriented software architecture for concurrent and networked software. In *SPLASH 2013 - Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, and Applications: Software for Humanity*, pages 167–176, Indianapolis Indiana USA, 2013. ACM.

[107] Till Schúmmer and Stephan Lukosch. *Patterns for Computer-Mediated Interaction*. Wiley & Sons, Chichester, UK, 1 edition, 2007.

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

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

[110] Fernanda Gomes Silva, Paulo Ezequiel Dias dos Santos, and Christina von Flach G. Chavez. FLOSS in Software Engineering Education: Supporting the Instructor in the Quest for Providing Real Experience for Students. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 234–243, Salvador Brazil, 2019. ACM.

[111] Williamson Silva, Bruno Gadelha, Igor Steinmacher, and Tayana Conte. What Are the Differences between Group and Individual Modeling When Learning UML? In *Proceedings of the XXXII Brazilian Symposium on Software Engineering*, SBES '18, pages 308–317, New York, NY, USA, 2018. Association for Computing Machinery.

[112] Thérèse Smith, Robert McCartney, Swapna S. Gokhale, and Lisa C. Kaczmarczyk. Selecting Open Source Software projects to teach software engineering. In *SIGCSE 2014 - Proceedings of the 45th ACM Technical Symposium on Computer Science Education*,

SIGCSE '14, pages 397–402, New York, NY, USA, 2014. Association for Computing Machinery.

[113] Tamires A S Sousa and Anna B S Marques. LEARN Board Game: A Game for Teaching Software Architecture Created through Design Science Research. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, SBES '20, pages 834–843, New York, NY, USA, 2020. Association for Computing Machinery.

[114] Francisco Carlos Souza, Alinne Santos, Stevão Andrade, Rafael Durelli, Vinicius Durelli, and Rafael Oliveira. Automating Search Strings for Secondary Studies. In Shahram Latifi, editor, *Information Technology - New Generations*, pages 839–848, Cham, 2018. Springer International Publishing.

[115] L Sun, Ting-qin Lv, and Lichun Pan. Reinforcing practice teaching of Software Engineering for fostering the creative talent. In *2011 International Conference on Consumer Electronics, Communications and Networks (CECNet)*, pages 1548–1551, Xianning, China, 2011. IEEE.

[116] Charles Thevathayan and Margaret Hamilton. Imparting Software Engineering Design Skills. In *Proceedings of the Nineteenth Australasian Computing Education Conference*, ACE '17, pages 95–102, New York, NY, USA, 2017. Association for Computing Machinery.

[117] Juan Urrego and Dario Correal. Archinotes: A tool for assisting software architecture courses. In *Software Engineering Education Conference, Proceedings*, pages 80–88, 2013.

[118] Arie Van Deursen, Maurício Aniche, Joop Aué, Rogier Slag, Michael De Jong, Alex Nederlof, and Eric Bouwers. A Collaborative approach to teaching software architecture. In *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, pages 591–596, Seattle Washington USA, 2017. ACM.

[119] Melina Vidoni, Jorge Montagna, and Aldo Vecchietti. Project and Team-Based Strategies for Teaching Software Architecture. *International Journal of Engineering Education*, 34:1701–1708, 2018.

[120] Alf Inge Wang. Extensive Evaluation of Using a Game Project in a Software Architecture Course. *ACM Trans. Comput. Educ.*, 11(1):28, 2011.

[121] Anthony Ward, Adeyosola Gbadebo, and Bidyut Baruah. Using job advertisements to inform curricula design for the key global technical challenges. In *2015 International Conference on Information Technology Based Higher Education and Training (ITHET)*, pages 1–6, Antalya, Turkey, 2015. IEEE.

[122] Gero Wedemann. Scrum as a Method of Teaching Software Architecture. In *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, pages 108–112, New York, NY, USA, 2018. Association for Computing Machinery.

[123] Shahani Markus Weerawarana, Amal Shehan Perera, and Vishaka Nanayakkara. Promoting creativity, innovation and engineering excellence. In *Proceedings of IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE) 2012*, pages T1C–12–T1C–17, Wuhan, China, 2012. IEEE.

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

[125] Kevin D Wendt, Ken Reily, and Mats P E Heimdahl. First Steps towards Exporting Education: Software Engineering Education Delivered Online to Professionals. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 241–245, Dallas, TX, USA, 2016. IEEE.

[126] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Springer, Germany, 2012.

[127] Bian Wu and Alf Inge Wang. Comparison of Learning Software Architecture by Developing Social Applications versus Games on the Android Platform. *Int. J. Comput. Games Technol.*, 2012, 2012.

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

[129] Olaf Zimmermann. Architectural Decisions as Reusable Design Assets. *IEEE Software*, 28(1):64–69, 2011.

[130] Camelia Şerban, Virginia Niculescu, and Andreea Vescan. Attaining competences in software quality oriented design based on cyclic learning. In *2020 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, Uppsala, Sweden, 2020. IEEE.