Guia 001 para crear cursos de Arquitectura de Software acordes a las expectativas de la Industria

W. Libardo Pantoja Yepez

September 12, 2023

Abstract

Este documento contiene una guía que permite al docente en cinco pasos diseñar su curso de Arquitectura de Software y seleccionar los patrones de formación que permiten desarrollar las competencias de creación, evaluación y documentación de arquitecturas de software acordes con las expectativas de la industria de software.

Contents

1 Introducción

En los últimos años debido a la expansión del software a gran escala en internet el tema de la arquitectura de software se ha vuelto más relevante [16]. Las aplicaciones informáticas son cada vez más complejas y exigentes a nivel de requisitos de calidad, por ejemplo, deben ser seguras, confiables, procesar muchas solicitudes de muchos usuarios concurrentes, manejar volúmenes de datos muy grandes, funcionar en ambientes distribuidos con tolerancia a fallos, fáciles de mantener, funcionar en distintos dispositivos, entre otras características. A medida que los sistemas software se vuelven más críticos, es necesario considerar los atributos de calidad y el tema de la Arquitectura de software [3].

En la actualidad la demanda de ingenieros con habilidades en arquitectura de software por parte de la industria es alta, por lo tanto se requieren cursos de arquitectura de software en las universidades [2]. Sin embargo, enseñar AS es una tarea difícil por múltiples desafíos que enfrenta el docente al asumir este tipo de cursos. En primera instancia, el rol del arquitecto es multifacético ya que requiere mucha experiencia en el diseño y desarrollo de sistemas reales, análisis para resolver problemas rápidamente y detectar la raíz, tomar decisiones importantes para el proyecto en base al contexto y habilidades de comunicación [35]. Además, la enseñanza de la Arquitectura es compleja porque requiere un contexto real, trabajo en equipo, proyectos con la suficiente complejidad y un acompañamiento permanente al alumno [2]. Es un desafío para el docente enseñar conceptos abstractos como principios, heurísticas y patrones a estudiantes de pregrado que carecen de experiencia en la industria del software, estos conceptos son difíciles de entender [18] [17] [41].

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 [27] [1] [2] [39]. Esto hace que planear y desarrollar un curso de arquitectura es una labor bastante compleja.

En este documento 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, que desarrollen 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 a partir de las distintas experiencias reportadas por profesores que proponen estrategias para formar sus estudiantes con habilidades en arquitectura de software. 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?

2 Descripción de la Guía

La guía que proponemos permite al docente estructurar su curso y seleccionar los patrones que utilizará en cinco pasos (ver Figura 1). A continuación explicamos brevemente cada uno de los pasos.

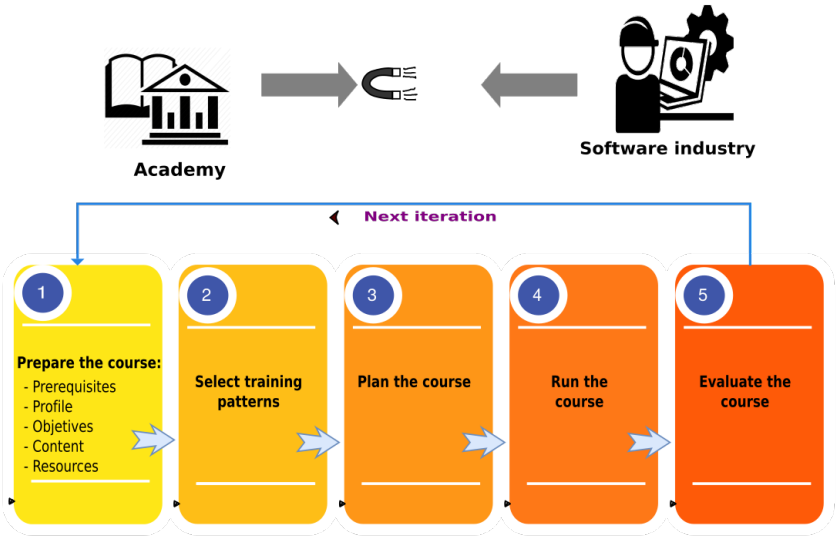

Figure 1: Guía en cinco pasos

2.1 Paso 1: Preparar el curso

Este paso consiste en realizar las primeras actividades para estructuras el cursos de AS. A continuación describimos cada una de ellas.

2.1.1 Tener en cuenta los prerrequisitos del curso

Los 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 [16]. 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.

Por lo tanto, el docente debe tener en cuenta las siguientes recomendaciones:

1. 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.

2. Ubicar la asignatura al menos en el semestre VI para que el estudiante haya cursado la mayoría de prerrequisitos.

3. Tener recursos disponibles como video tutoriales, guias, etc., que permitan recordar y/o reforzar temas de prerrequisitos que sean fundamentales para abordar el curso.

.

2.1.2 Definir el perfil del curso

Un curso de arquitectura de software puede tener varios perfiles. 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. A continuación mencionamos algunos posibles perfiles.

1. *Perfil técnico.* Un perfil técnico persigue 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 aplicaciones flexibles ante cambios futuros.

2. *Perfil administrativo.* Un perfil administrativo fomenta en los estudiantes la adquisición de las habilidades blandas de un arquitecto, tales como: comunicación, liderazgo y negociación y trabajo en equipo (ver Tabla 1).

3. *Perfil híbrido.* Un perfil con la combinación de características de los perfiles anteriores.

No	Habilidad blanda	Descripción
1	Comunicación	Las habilidades de los arquitectos incluyen habilidades para hablar, escribir y hacer presentaciones para abordar problemas complejos con un diseño aparentemente simple que es fácil de entender [33]. Además, deben poder interactuar y comunicarse con expertos en todas estas disciplinas. Los arquitectos deben comprender la terminología, los problemas típicos, las necesidades y las diferentes culturas (forma de trabajar y pensar) [3]. La arquitectura del software trata de la comunicación entre las partes interesadas y lograr el consenso [35]. También debe escuchar y resolver problemas con quienes implementan un diseño [39].
2	Trabajo en Equipo	Los arquitectos de software tienden a supervisar y trabajar en estrecha colaboración con otros miembros del equipo de desarrollo, por ejemplo, los programadores [21] [22]. Aprender arquitectura es un proceso intensivo de trabajo en equipo [17]. Las decisiones arquitectónicas deben analizarse en equipo [2].
3	Liderazgo y negociación	La comunicación y el liderazgo son esenciales para que un arquitecto pueda liderar, presentar, negociar y justificar sus diseños arquitectónicos y decisiones [1] [19] [33]. Estas habilidades incluyen tomar decisiones, tomar la iniciativa e innovar, demostrar un juicio independiente, ser influyente e imponer respeto [4].
4	Habilidades artísticas	Capacidad artística para hacer diseños aparentemente simples que son fáciles de entender y, sin embargo, resuelven problemas complejos [33] [39]. Las habilidades artísticas implican el conocimiento de diferentes herramientas y técnicas de diseño, cómo diseñar sistemas complejos multiproducto, análisis y diseño orientado a objetos, diagramas UML y modelado de análisis UML [4].

Table 1: Habilidades blandas fundamentales del arquitecto de software

Respecto al perfil, recomendamos para el docente:

1. Definir el perfil que tendrá su curso de arquitectura de software. Se recomienda tener en cuenta el tipo del curso, el perfil del egresado, y los perfiles mencionados en esta sección.

2.1.3 Definir los objetivos de aprendizaje

Una vez definido el perfil del curso de Arquitecturas de Software se procede a definir los objetivos de aprendizaje. Recomendamos (en base a nuestra experiencia) los siguientes objetivos de aprendizaje para un primer curso de arquitecturas de software:

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.

Como un complemento interesante para el docente, la Tabla 2 muestra un listado de objetivos de aprendizaje y un mapeo con los perfiles de cursos. Varios objetivos son bastante similares, simplemente se acude a diferentes formas de expresar la misma idea.

No	Perfil	Objetivo de aprendizaje
1	Técnico	Analizar los requisitos de negocio para extraer y definir las características arquitectónicas [17].
2	Técnico	Seleccionar qué patrones y estilos de arquitectura encajarían en el dominio del problema [17].
2	Técnico	Explicar cómo afectan los atributos de calidad al diseño de la arquitectura del sistema [14].
3	Técnico	Elaborar el diseño detallado del software mediante la creación de diagramas de clases (y otros diagramas UML seguń del caso) para cada componente, la creación de pantallas de interfaz de usuario, el desarrollo y pruebas del código fuente.
4	Técnico, Administrativo	Definir y explicar conceptos centrales de arquitectura de software, usar y describir patrones de diseño/arquitectura, métodos para diseñar arquitecturas de software, métodos/técnicas para lograr cualidades de software y métodos para documentar la arquitectura de software y evaluar la arquitectura de software [36].
5	Técnico	Corregir problemas arquitectónicos en sistemas existentes [33].
6	Técnico	Analizar y diseñar arquitectura de software y adquirir experiencia relevante en sistemas a gran escala [41].
7	Técnico	Revisar arquitecturas de sistemas de software y métodos de ingeniería de software relacionados para diseñar sistemas complejos de software intensivo [7].
8	Técnico	Enseñar a los estudiantes métodos modernos de arquitectura de software y hacer posible que los estudiantes se conviertan en buenos arquitectos de sistemas dentro de los 5 u 8 años posteriores a la graduación [14].
9	Técnico	Enseñar a los estudiantes métodos modernos de arquitectura de software y hacer posible que los estudiantes se conviertan en buenos arquitectos de sistemas dentro de los 5 u 8 años posteriores a la graduación [14].
10	Técnico	Demostrar la capacidad de tener éxito en una experiencia de diseño de software similar al mundo real. [22].

Table 2: Posibles objetivos de aprendizaje para el curso de Arquitecturas de Software

Si en lugar de objetivos de aprendizaje el docente trabaja competencias, la Tabla A contiene un listado de competencias del rol de arquitecto de software

clasificadas por al Industria en obligatorias, opcionales y fuera de alcance para un recién egresado.

Finalmente, respecto a los objetivos de aprendizaje, recomendamos para el docente:

1. Acorde al perfil elegido del curso de arquitectura de software, definir los objetivos de aprendizaje basándose en los objetivos recomendados para un nuevo curso y en la Tabla 2.

2.1.4 Definir el contenido del curso

Una vez definidos los objetivos de aprendizaje se deben definir los contenidos del curso. La Tabla 3 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.

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.

Table 3: Contenido de un curso de Arquitecturas de Software para programas de pregrado

Un libro actualizado que puede servir como referencia para la mayoría de los temas del curso es Fundamentals of software Architecture de Mark Richards [32].

Respecto al contenido del curso, recomendamos para el docente:

1. Con los objetivos de aprendizaje claros definir los contenidos del curso basándose en la Tabla 3.

2.1.5 Establecer los recursos disponibles

Un curso de arquitectura de software acorde a las necesidades de la industria es algo esencial en los planes de estudio de programas de informática y áreas afines [17]. Sin embargo, formar estudiantes de pregrado con las habilidades que demanda la industria tiene muchos desafíos [33]. 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 en proyectos reales, entre otros [33] [41] [18] [2] [35] [19]. En resumen, enseñar arquitecturas de software sigue siendo complicado a menos que el docente sea un diseñador de profesion [33].

Debido a la situación anterior, 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. En seguida se explica cada uno de ellos.

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 [33]. 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 [36].

Enseñar arquitecturas de software sigue siendo una tarea difícil. 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 [27] [1] [2] [39]. Esto hace que planear y desarrollar un curso de arquitectura es una labor bastante compleja para el docente.

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 colaborativo en un entorno del mundo real. Los obstáculos para trabajar con aprendizaje colaborativo suelen ser tres: (i) la resistencia al cambio de paradigma que requieren estudiantes e instructores, (ii) contar con un diseño adecuado de la actividad colaborativa a aplicar, y (iii) tener una buena solución tecnológica para apoyar la actividad (especialmente en entornos distribuidos) [30] [29] [31] [35] [26].

Recursos humanos-clientes. Formular proyectos de software y clientes reales es algo beneficioso para los futuros arquitectos de software pero hay dificultad para involucrar clientes y expertos de la industria [9]. 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 [9].

Recursos tecnológicos-contenidos. Los docentes van a necesitar material de arquitectura de software actualizados 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 [10]. 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 [14] [41] [41] [14].

Finalmente, en cuanto a los recursos que necestia el docente para su curso de arquitectura de software, damos las siguietnes recomendaciones:

1. Debido a que muchos profesores siempre han sido académicos y no tienen experiencia en el diseño de grandes proyectos reales, se recomienda involucrar a estudiantes de maestría y doctorado provenientes del mundo de la industria de software. De esta manera trabajar en equipo permite no sólo tener contactos con la industria, sino además repartir las tareas que involucra impartir un curso de arquitecturas de software.

2. Hay dificultad para involucrar clientes y expertos de la industria. 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.

3. Es importante contar con un repositorio de material que esté siendo alimentado y actualizado constantemente por los docentes para poder preparar las clases del curso. Los blogs y las lecturas sobre las últimas tendencias arquitectónicas son buenas fuentes de conocimiento.

2.2 Paso 2: Seleccionar y adecuar los patrones de formación del catálogo

En este paso, el docente puede elegir uno o varios de los patrones de formación en AS que le permitirán a los estudiantes desarrollar habilidades de arquitectura alineadas con las expectativas de la industria.

La Tabla 4 muestra un resumen de los patrones de formación.

No	Patrón	Resumen
1	Small Project-based Training	Trabajando con varios proyectos pequeños los estudiantes pueden llevar a la práctica los conceptos de patrones de arquitectura y tácticas de arquitectura para favorecer el cumplimiento de los atributos de calidad
2	Large Project-based Training	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
3	Training contributing to open-source projects	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.
4	Open-source projects to the classroom	Trabajar con proyectos open-source, a pesar de las ventajas de enfrentar a los estudiantes a modificar sistemas reales, puede 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. Se trata de una experiencia de aprendizaje basado en proyectos, que se centra en un sistema completo de código abierto para enseñar de manera efectiva la arquitectura de software distribuido.
5	Cases and flipped classroom training	Son talleres relacionados con la industria en forma de caso de estudio, que puedan guiar y hacer que los participantes pasen por los procesos de pensamiento de un arquitecto junior. Los casos se suelen enseñar mediante el aula invertida.
6	Problem-solving training	El enfoque de aprendizaje basado en problemas permite trabajar en equipos en corto tiempo mejorando las habilidades en diseño de arquitectura.
7	Games-based training	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.

Table 4: Catálogo de los patrones de formación en AS

El formato que elegimos para documentar los patrones de formación es el siguiente:

- **Nombre**: es una frase corta que describe el nombre del patrón.

- **Contexto problemático**: describe la situación problemática o circunstancia en la cual el patrón es aplicado.
- **Nivel**: Describe el nivel de complejidad del curso: básico, intermedio o avanzado.
- **Fuerzas**: Representan un escenario concreto que provee la motivación para el uso del patrón.
- **Solución**: Describe la solución al problema aplicando el patrón.
- **Requisitos previos**: Los requisitos previos de los estudiantes para poder aplicar el patrón.
- **Ejemplo**: Un ejemplo sobre cómo utilizar el patrón.
- **Competencias**: Lista de competencias que los estudiantes desarrollan al aplicar el patrón.
- **Variantes**: describe los casos o situaciones alternativas sobre cómo aplicar el patrón.
- **Ventajas**: Consecuencias positivas que trae aplicar el patrón.
- **Desventajas**: Consecuencias negativas que trae aplicar el patrón.
- **Patrones relacionados**: 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 de artículos que aplicaron el patrón en alguna experiencia de formación de AS.

La Tabla 5 sirve para definir los patrones de formación seleccionados por el docente para su curso los cuales deben ser seleccionados tomando como base los patrones de la Tabla 4.

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

Table 5: Plantilla definir los patrones de formación elegidos por el docente.

A continuación describimos en detalle cada uno de los patrones de formación. Puede leer solamente los que resulten de su interés para su curso.

2.2.1 Patrón 1

Nombre: Architecture-Style-Driven Mini Projects.

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 [4]. De acuerdo con Sherman [34], 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 [18].

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 alumnos 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 [6]. 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 ex-

periencia concreta de diseño arquitectónico teniendo en cuenta las restricciones de tiempo y de recursos tanto de los alumnos como de la infraestructura que cuentan las universidades.
Nivel: Básico

Fuerzas:

- Mediante la aplicación de estilos y tácticas arquitectónicas, el profesor quiere formar a los estudiantes para que consigan 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 profesor 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 alumnos 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 proyectos pequeños 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 [33, 17].

Los estudiantes desarrollan en equipo uno o 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 [33]. 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 [17].

Es recomendable que los proyectos y ejemplos estén desarrollados en tecnologías con los que los estudiantes y profesores estén familiarizados desde sus cursos previos, java, otras en C#, C++, python, etc. 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 2 muestra los principales elementos de la solución del patrón.

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

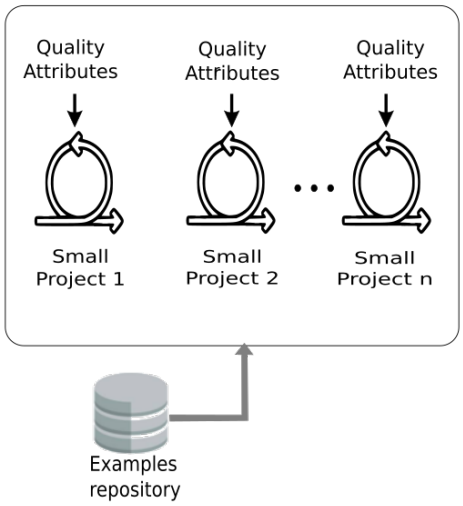

Figure 2: Small Project-based Training

- Introducción o contexto del sistema de software a desarrollar.
- Objetivos de aprendizaje a desarrollar en los alumnos (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. La Figura 3 muestra una arquitectura en capas.

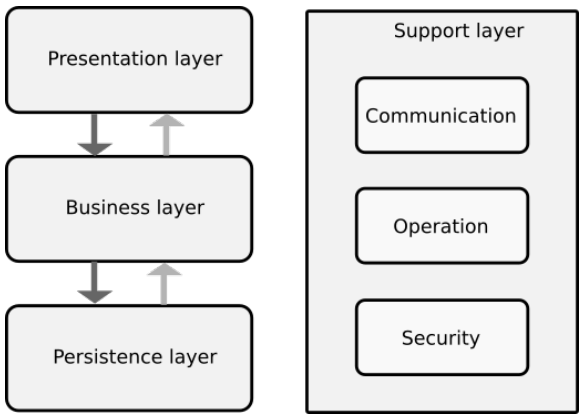

Figure 3: Arquitectura en capas

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 (ver Figura 4). La arquitectura también sirve como enchufe para conectar estas extensiones y coordinar su colaboración.

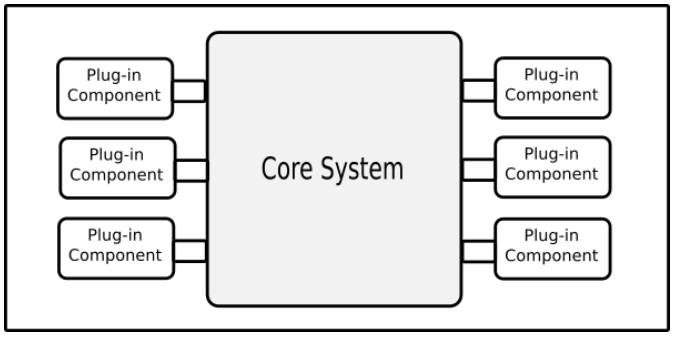

Figure 4: Arquitectura microkernel

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** (ver Figura 5).

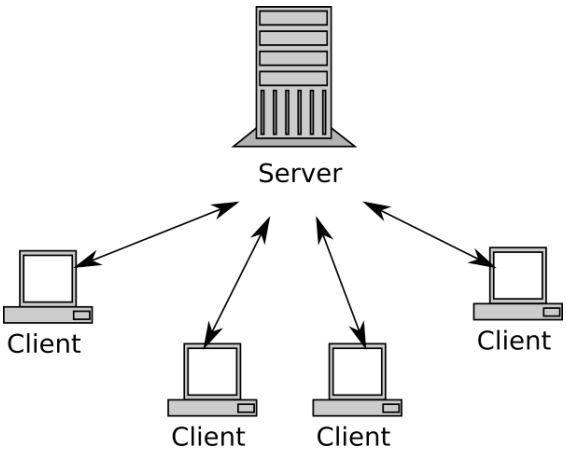

Figure 5: Arquitectura Cliente-Servidor

El cuarto miniproyecto, el profesor puede trabajar la arquitectura **dirigida por eventos** con una topologia 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 (ver Figura 6). 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 (ver Figura 7). 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 Anexo A: 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

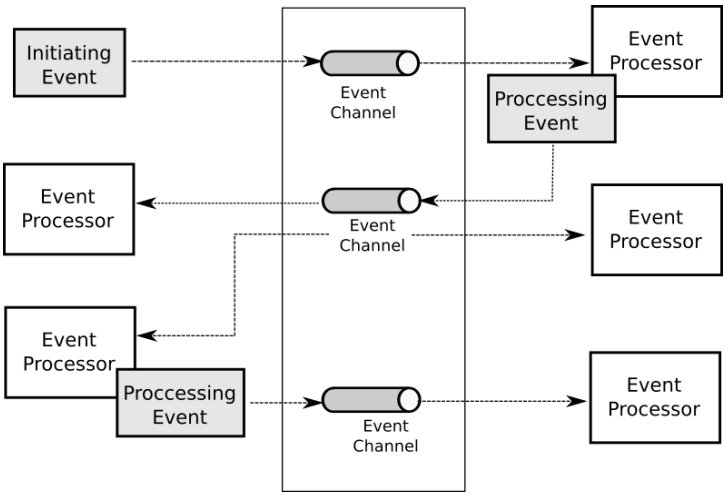

Figure 6: Broker topology

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 cpuede combinarse en un curso de SA con otros patrones de formación que suponen poco esfuerzo para el alumno y permiten desarrollar diversas competencias para la toma de decisiones. Por ejemplo, Case and Flipped Classroom Training, Problem-Solving Training, y Games-Based Training.

Experiencias reportadas del uso del patrón:

- Teaching Software Architecture to Undergraduate Students: An Experience Report [33].

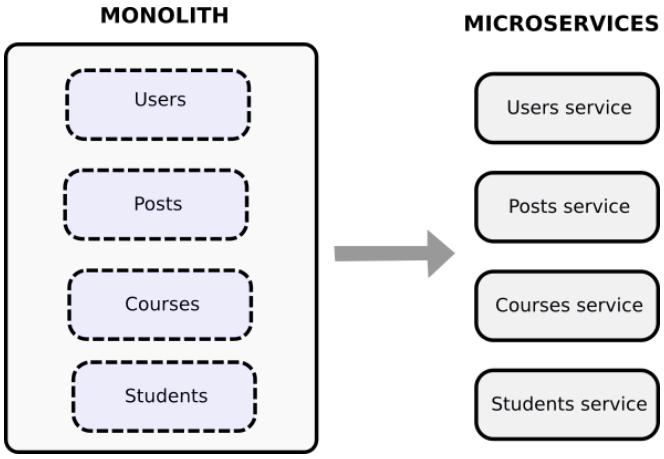

Figure 7: Arquitectura de microservices

- Using Public and Free Platform-as-a-Service (PaaS) based Lightweight Projects for Software Architecture Education [17].
- Lean learning - Applying lean techniques to improve software engineering education [6]

2.2.2 Patrón 2

Nombre: Large Project-based Training.

Contexto problemático: Los alumnos tienen competencias suficientes en desarrollo de software para afrontar proyectos de desarrollo medianos y grandes y en dominios complejos [6]. Por tanto, los alumnos 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 alumnos 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 alumnos..

Nivel: Intermedio

Fuerzas:

- El profesor 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.

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 [36]. 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 industriales significativos y realistas para los proyectos se requiere que la Universidad tenga convenios con empresas [9] y disponer de un banco de proyectos. Muchas universidades disponen de convocatorias semestrales donde las empresas postulan ideas de proyectos y los docentes evaluan 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 [33]. 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 [2]. 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 8 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 [38]. 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 [33].

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 [9].

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.

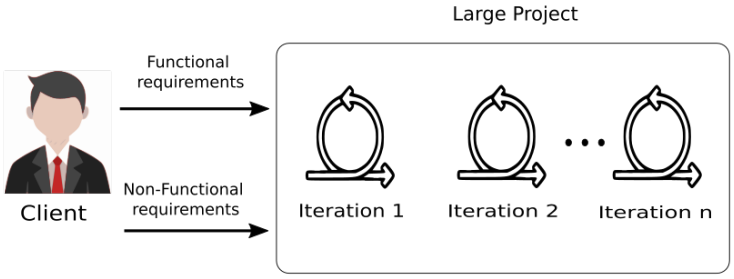

Figure 8: Large Project-based Training

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: 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 [27]. 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 Anexo A: 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 [9].

Patrones relacionados: El patrón Large Project-based Training pattern se puede combinar en un curso de AS con otros patrones de formación que involucrarn poco esfuerzo de los estudiantes y permiten desarrollar varias competencias en toma de decisiones. Por ejemplo, Case and Flipped Classroom

Training, Problem-Solving Training, y Games-Based Training.

Experiencias reportadas del uso del patrón:

- Exploration on Theoretical and Practical Projects of Software Architecture Course [41].
- Extensive Evaluation of Using a Game Project in a Software Architecture Course [36].
- Using game development to teach software architecture [33].
- Comparison of learning software architecture by developing social applications versus games on the android platform [40].
- Scrum as a Method of Teaching Software Architecture [37].
- Designing and applying an approach to software architecting in agile projects in education [2].
- Fontys ICT, Partners in Education Program: Intensifying Collaborations Between Higher Education and Software Industry [9]

2.2.3 Patrón 3

Nombre: Training contributing to open-source projects.

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. Sin embargo, 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 del producto [13].

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 [38]. En el argot de Ingeniería de Software coloquial, 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 consideran fáciles de abordar los proyectos de escenarios totalmente nuevos. 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 [38].

Nivel: Avanzado

Fuerzas:

- El profesor 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 estudiantes tienen cierta experiencia trabajando proyectos de desarrollo complejos.

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 [30]. Además, 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 [35]. La Figura 9 muestra los principales elementos de la solución del patrón.

El docente puede previamente seleccionar los proyectos open-source teniendo en cuenta su complejidad arquitectural y las habilidades de los estudiantes. Además, los estudiantes pueden la oportunidad de elegir el proyecto de su preferencia y de esta forma aumentar el nivel de motivación en el proceso de aprendizaje.

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

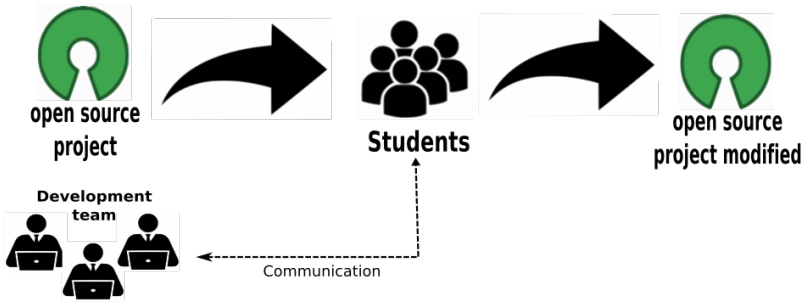

Figure 9: Training contributing to open-source projects

- Sistemas Distribuidos
- Ingeniería de Software

Ejemplo de uso del patrón:

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 6:

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

Table 6: Ejemplo de listado de proyectos open-source [30]

Luego, los estudiantes de manera individual o en equipos eligen el proyecto donde harán la contribución guiados por el instructor y por sus intereses [30].

Después de elegido el proyecto el próximo paso es seleccionar las tareas que el estudiante trabajará en el curso [30]. 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 se de 4 tipos según Hattori and Lanza [12]:

1. *Ingeniería directa*, agregando nuevas características. Ejemplo, una contribución al repositorio de LibreOffice Impress, es permitir al usuario cambiar diapositivas usando un teléfono inteligente.

2. *Reingeniería*, actividades de refactorización. Ejemplo: refactorizar algunas capas que no son del todo apropiadas según las mejores prácticas de Android

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.

4. *Gestión*, por ejemplo, actualización de documentación, reportar bugs, agregó etiquetas en problemas (issues), seguir los procesos de scrum (sprints, historias de usuario, planificación poker, etc.).

Finalmente, el docente hace seguimiento de los proyectos [30]. 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.

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Anexo A: Tabla de Competencias):
Competencias obligatorias: C11, C22, C23.
Competencias opcionales: C24, C14.

Variantes para llevarlo a la práctica: Ninguna.

Ventajas:

- Los estudiantes trabajan con proyectos reales [30].
- Se generan habilidades interactuando con sistemas de control de versiones.
- Los estudiantes se vuelven miembros de una comunidad de desarrollo activa [30].
- 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 [7].

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 [30].
- 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.
- 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 linea de comando, etc.
- Esta estrategia resulta más compleja para los instructores para orientar adecuadamente a los estudiantes [30].

Experiencias reportadas del uso del patrón:

- Training software engineers using open-source software: the students' perspective [30].

- A Collaborative approach to teaching software architecture [35].
- Promoting creativity, innovation and engineering excellence [38]

2.2.4 Patrón 4

Nombre: Open-source projects to the classroom.

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 [30].

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 [39].
Nivel: Intermedio

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. Por ejemplo, los docentes pueden tener su propio sistema E-commerce B2C como herramienta pedagógica para trabajar atributos de disponibilidad, seguridad y escalabilidad [39]. 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 10).

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 [39]. 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 la lección. 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. Los estudiantes pueden descargar los proyectos en sus máquinas, estudiar el código fuente, los manuales, ejecutarlos y probarlos.

Si se unen esfuerzos de varios docentes de distintas Universidades, se puede llegar a tener un repositorio de proyectos open-source realistas de sectores industriales [39], 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.

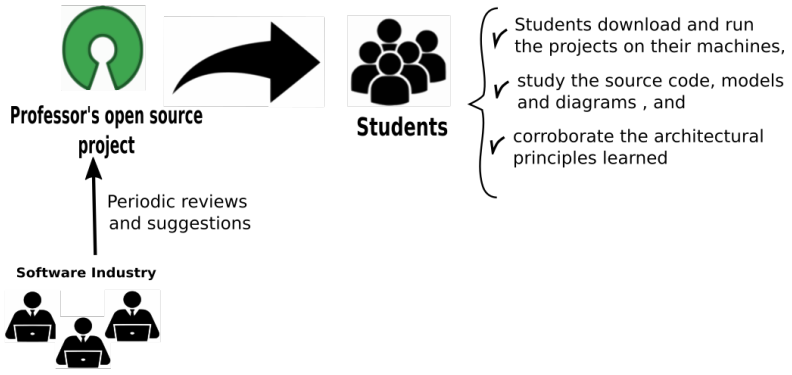

Figure 10: Open-source projects to the classroom

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 ser un excelente socio para que las universidades sepan si, con el tiempo, la aplicación sigue siendo válida como aplicación empresarial.

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:

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 [39].

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 alumnos 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 [39]. 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, podemos pasar a la arquitectura distribuida [39]. 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 Anexo A: 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 Open-source projects to the classroom se puede combinar en un curso de AS con otros patrones de formación que involucren poco esfuerzo para el estudiante y permita desarrollar varias competencias en toma de decisiones. Por ejemplo, Case and Flipped Classroom Training, Problem-Solving Training, y Games-Based Training.

Experiencias reportadas del uso del patrón:

- Teaching Distributed Software Architecture by Building an Industrial Level E-Commerce Application [39].
- A Collaborative Approach to Teaching Software Architecture [35]

2.2.5 Patrón 5

Nombre: Cases and flipped classroom 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. Además, las decisiones proporcionan una elección realizada por el arquitecto de software en un contexto específico junto con su justificación o razón de ser [32].

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 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 [32]. 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). Por

último, 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?

Nivel: Intermedio

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 aplicar compensaciones (trade-off) dentro de un contexto realista [20].

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 [20]. Los principales pasos de este patrón de formación se pueden apreciar en la Figura 11.

Los casos se suelen enseñar mediante el aula invertida. El aprendizaje de aula invertida promueve una mayor inmersión en un tema que la enseñanza tradicional, mejorando el compromiso, la interacción y la cooperación de los estudiantes al ofrecerles el contenido antes del aula [11]. El aula invertida tiene tres momentos antes del encuentro con el docnente, durante y después.

El patrón de formación propone realizar talleres relacionados con la industria en forma de caso de estudio, que puedan guiar y hacer que los participantes pasen por los procesos de pensamiento de un arquitecto junior [20]. Estos talleres podrían incluir ejemplos de proyectos reales que muestren situaciones por las que pasa un arquitecto de soluciones en un entorno de trabajo real. El profesor 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 [24]:

1. Primero el docente enseña los fundamentos de arquitecturas.

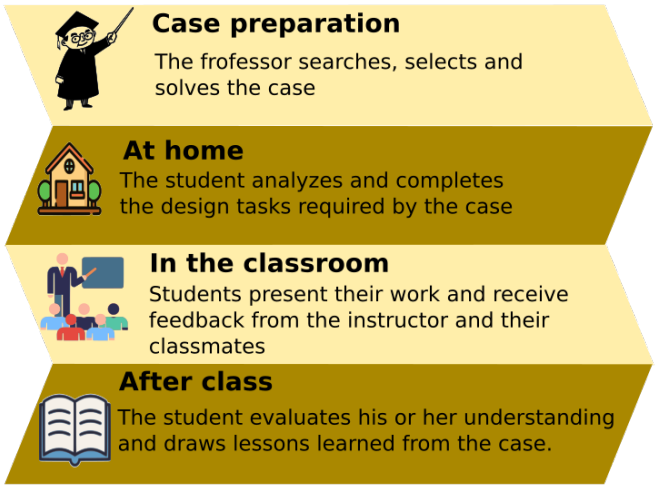

Figure 11: Cases and flipped classroom training

2. El docente define el tema y objetivos de aprendizaje del caso.

3. El docente busca y selecciona el caso. El profesor puede buscar casos 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 alumnos. Podría incluir descripciones detalladas del caso, datos relevantes, reflexiones, preguntas para el debate y recursos adicionales para ayudar a los alumnos a comprender el contexto.

6. Trabajo individual o en grupo. Los alumnos 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 profesor 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 alumnos. 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 alumnos 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 alumnos 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 alumnos, 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 alumnos 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 alumnos 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 alumnos 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 alumno 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.
- Los estudiantes pueden leer capítulos de libros de texto, artículos académicos o documentos relacionados con el caso que se tratará en clase.
- Los alumnos pueden 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 alumnos a investigar en Internet o en otras fuentes para profundizar en el tema y descubrir información adicional.
- Se puede invitar a los alumnos 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.
- Los estudiantes pueden participar en foros o plataformas en línea donde discuten contenidos previos con sus compañeros, responden a preguntas planteadas por el profesor o generan debates.
- Los alumnos pueden 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 alumnos deben poder recomendar acciones de mitigación [25].

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Anexo A: 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 [8] [41]. 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 [18]. Por lo tanto, el docente tiene un gran reto al momento de resolver el caso.

Patrones relacionados:
El patrón Cases and Flipped Classroom Training se puede combinar en un curso de AS con otros patrones de formación que involucran el desarrollo de un proyecto. Por ejemplo, Architecture-Style-Driven Mini Projects, Large Project-Based Training, Training Contributing to Open-Source Projects, y Open-Source projects to the Classroom.

Experiencias reportadas del uso del patrón:

- Applying case-based learning for a postgraduate software architecture course [25].
- Did our Course Design on Software Architecture meet our Student's Learning Expectations? [20].
- Exploring Experiential Learning Model and Risk Management Process for an Undergraduate Software Architecture Course [18].
- Improved Teaching Model for Software Architecture Course [14].
- Flipped Classroom Applied to Software Architecture Teaching [11].

2.2.6 Patrón 6

Nombre: Problem-solving 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 [32]. 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 alumnos 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 alumnos, 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 alumnos 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 alumnos pueden sentirse presionados para ceder en sus puntos de vista, lo que puede afectar a la calidad de la toma de decisiones.

Nivel: Básico

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 alumnos 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 [19]. 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 alumnos, relacionado con los objetivos del curso de AS.

2. Presentación del problema. El instructor presenta el problema a los alumnos de forma clara y concisa. Proporciona la información necesaria para que los alumnos comprendan el contexto del problema.

3. Creación de equipos. El instructor organiza a los alumnos en equipos de colaboración. 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 alumnos 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 ejemplo del enfoque 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 (ver Figura 13).

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 Figure 12). 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 [32].

4. Luego, viene la *presentación* de las propuestas. Para ello se elige una o dos personas por equipo para que expongan sus propuestas.

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 [28].

Figure 12: Ejemplo de un ejercicio de kata generado por el sitio architecturalkatas.com

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:

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

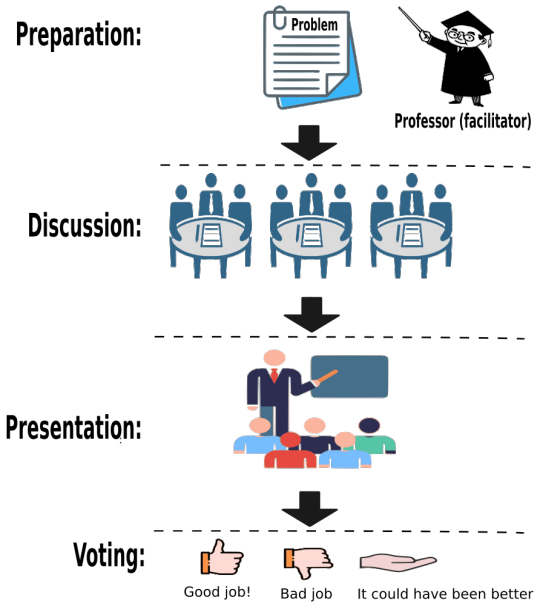

Figure 13: Problem-solving training (Architecture katas)

- 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 Anexo A: 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 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, Architecture-Style-Driven Mini Projects, Large Project-Based Training, Training Contributing to Open-Source Projects, and Open-Source projects to the Classroom.

Experiencias reportadas del uso del patrón:

- Teaching adult learners on software architecture design skills [19] [11].
- Applying case-based learning for a postgraduate software architecture course [25].
- Aligning Software Architecture Training with Software Industry [28].

2.2.7 Patrón 7

Nombre: Games-based training.

Contexto problemático:
Enseñar arquitectura de software es una disciplina difícil porque el rol de arquitecto es multifascetico [18]. 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. Sin embargo, 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. [5].

Los docentes necesitan métodos de enseñanza que sean divertidos y permitan 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 alumnos. 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 alumnos 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 alumnos. Los alumnos pueden perder interés o frustrarse si el juego es demasiado fácil o difícil.

Nivel: Básico

Fuerzas:

- El docente quiere desarrollar habilidades en sus estudiantes relacionados con la toma de decisiones de arquitectura de un sistema software nuevo de manera divertida.
- 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:

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 [15]. Los juegos permiten desarrollar habilidades en diseño de software de manera divertida y atractiva para los estudiantes (ver Figura 14).

El juego no es un sustituto de la instrucción "tradicional" sobre diseño, sino más bien como un complemento de dicha instrucción [15]. 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.

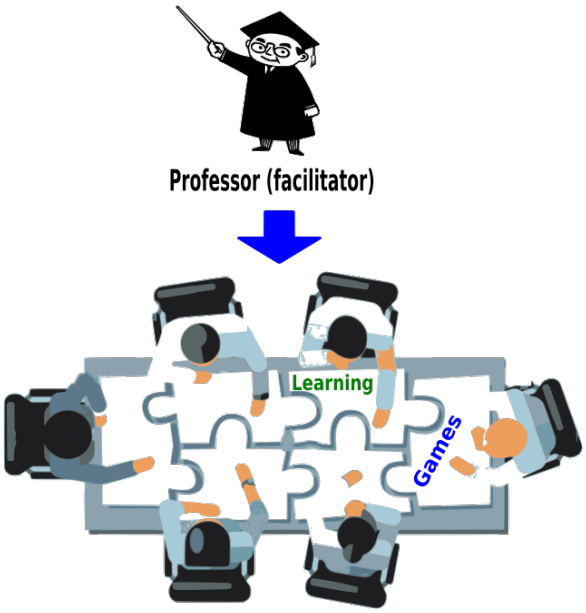

Figure 14: Games-based training

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.

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: A continuación describimos 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) [5]. 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. A continuación, 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.

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 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 greenfield. Los detalles y la mecánica del juego se describen en [5].

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 [15]. 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 [15].

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 [23].

Competencias que se abordan: Con este patrón de formación se logran desarrollar las siguientes competencias (Ver Anexo A: 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, Architecture-Style-Driven Mini Projects, Large Project-Based Training, Training Contributing to Open-Source Projects, y Open-Source projects to the Classroom.

Experiencias reportadas del uso del patrón:

- Smart Decisions: An Architectural Design Game [5]
- A role-playing game to teach ATAM (Ar- chitecture Trade-off Analysis Method) a simulation tool and case study [23].
- DecidArch: Playing cards as software architects [15]y [28].

2.3 Paso 3: Planificar el curso

La planificación del curso de AS debe tener en cuenta las decisiones tomadas en los pasos anteriores. Como elementos importantes, se debe tener claros los objetivos de aprendizaje establecidos, los contenidos y los patrones de formación elegidos.

La Tabla 7 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.

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 training	Grupos de tres personas, se trabajarán dos ejercicios de katas.
13	...	...	...
16	Examen Final	Exámen en grupos de dos	Previo se dejará un taller de preparación

Table 7: Plantilla de planificación del curso de AS con algunos ejemplos para guiar al docente.

Finalmente, en este paso de planificación damos las siguientes recomendaciones para el docente:

1. La planificación se debe realizar con el suficiente tiempo para que se realice de forma lo más cercana posible a su ejecución.

2. Una buena planificación permite prevenir problemas que se pudieran presentar o, en su defecto, tener tiempo para solucionar y sobreponerse ante cualquier contratiempo que surja fuera de lo establecido.

2.4 Paso 4: Ejecutar el curso

La ejecución del curso consiste en llevar a cabo los aspectos definidos en la planificación. 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 muesta la Tabla 8.

Semana No	Fecha	Temática	Actividades	Observación

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

2.5 Paso 5: Evaluar el curso

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 Tabla A. 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.

References

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] Alf Inge Wang. Extensive evaluation of using a game project in a software architecture course. *ACM Trans. Comput. Educ.*, 11(1):28, 2011.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

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

A Tabla de Competencias

Estas competencias fueron obtenidas de una revisión bibliográfica y a cada una le asignamos un identificador C1, C2, 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 components/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.