4. Catálogo de patrones de formación en arquitectura de software

Resumen

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

4.1. Introducción

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

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

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

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

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

4.2. Metodología para extraer los patrones de formación de AS

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

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

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

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

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

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

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

A continuación explicamos los detalles de los pasos anteriores.

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

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

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

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

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

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

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

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

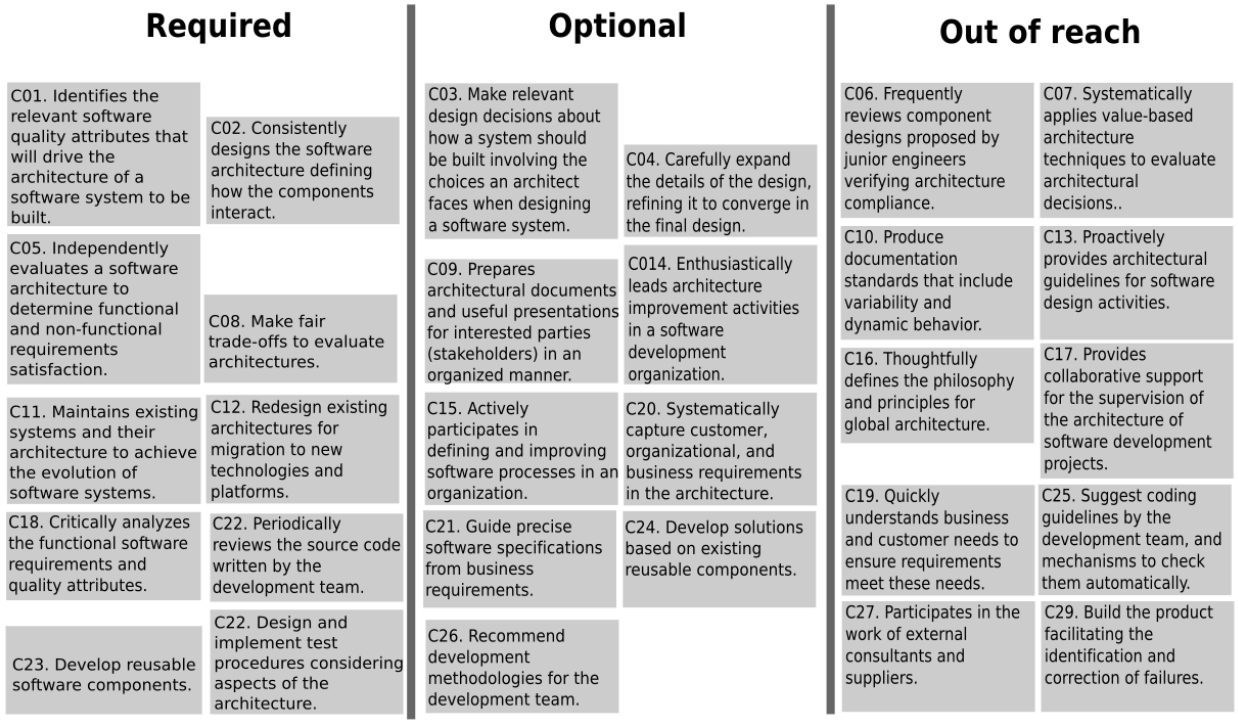

Figura 4-1.: Clasificación final de las competencias en AS

4.2.3. Definición de la estructura interna para especificar los patrones

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

Tabla 4-2.: Plantillas de patrones

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

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

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

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

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

4.2.4. Extracción de los patrones

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

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

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

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

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

- Aprendizaje basado en juegos

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

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

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

Tabla 4-4.: Patrones de formación encontrados

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

4.3. Catálogo de patrones

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

4.3.1. Patrón 1: Mini-Projects-based training

Nombre: Mini-Projects-based training.

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

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

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

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

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

Fuerzas:

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

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

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

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

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

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

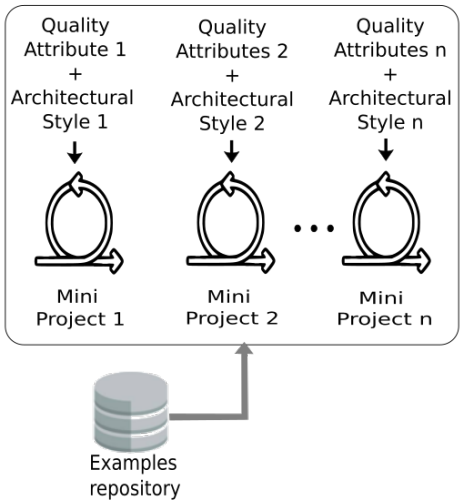

Figura 4-2.: Small Project-based Training

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

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

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos

Deseables:

- Sistemas operativos
- Sistemas Distribuidos
- Ingeniería de Software

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

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

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

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

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

Boot, MicroProfile, WildFly Thorntail, Cricket, etc.

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

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

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

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

Ventajas:

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

Desventajas:

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

Patrones relacionados:

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

Experiencias reportadas del uso del patrón:

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

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

4.3.2. Patrón 2: Large Project-based Training

Nombre: Large Project-based Training.

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

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

Fuerzas:

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

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

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

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

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

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

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

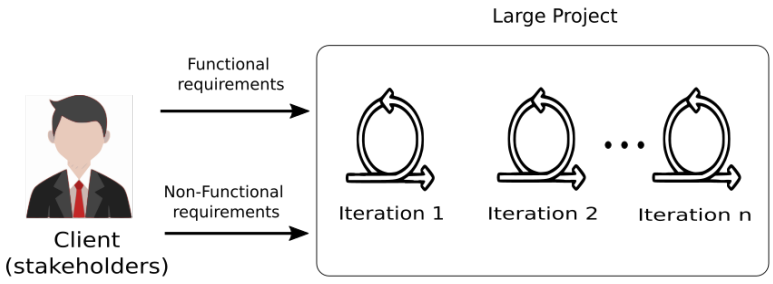

Figura 4-3.: Large Project-based Training

Requisitos previos de los estudiantes:

Obligatorios:

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

Deseables:

- Sistemas operativos
- Sistemas Distribuidos
- Ingeniería de Software

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

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

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

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

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

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

Variantes para llevarlo a la práctica: Ninguna.

Ventajas:

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

Desventajas:

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

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

Experiencias reportadas del uso del patrón:

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

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

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

Nombre: Open-source projects-based training.

Contexto problemático:

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

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

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

Fuerzas:

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

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

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

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

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

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

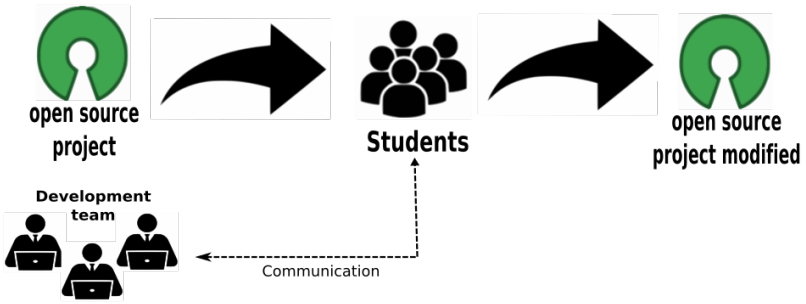

Figura 4-4.: Open-source projects-based training

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

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

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

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

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

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

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

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

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

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

Requisitos previos de los estudiantes:

Obligatorios:

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

Deseables:

- Sistemas operativos
- Sistemas Distribuidos
- Ingeniería de Software

Ejemplo de uso del patrón:

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

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

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

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

Competencias obligatorias: C11, C22, C23.

Competencias opcionales: C24, C14.

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

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

.

Ventajas:

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

Desventajas:

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

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

Experiencias reportadas del uso del patrón:

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

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

Nombre: In-house project-based training.

Contexto problemático:

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

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

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

Fuerzas:

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

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

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

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

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

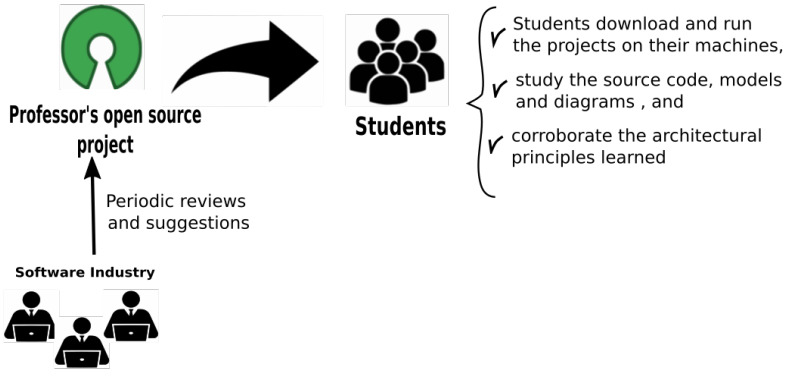

Figura 4-5.: In-house project-based training

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

Requisitos previos de los estudiantes:

Obligatorios:

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

Deseables:

- Sistemas operativos
- Sistemas distribuidos

- Ingeniería de software

Ejemplo de uso del patrón:

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

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

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

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

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

enfoque DevOps.

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

Competencias obligatorias: C01, C05, C08.

Competencias opcionales: C11, C12.

Variantes para llevarlo a la práctica: Ninguna.

Ventajas:

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

Desventajas:

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

Patrones relacionados:

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

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

Experiencias reportadas del uso del patrón:

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

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

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

4.3.5. Patrón 5: Cases-based training

Nombre: Cases-based training.

Contexto problemático:

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

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

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

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

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

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

Fuerzas:

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

Solución:

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

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

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

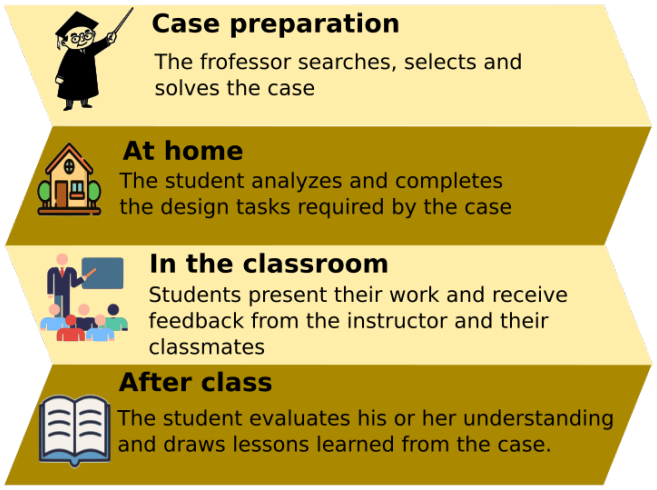

Figura 4-6.: Cases-based training

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

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

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

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

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

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

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

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

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

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

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

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos

Deseables:

- Sistemas operativos

- Sistemas Distribuidos
- Ingeniería de Software

Ejemplo de uso del patrón:

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

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

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

Competencias opcionales: C03.

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

Ventajas:

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

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

Desventajas:

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

Patrones relacionados:

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

Experiencias reportadas del uso del patrón:

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

4.3.6. Patrón 6: Problem-solving-based training

Nombre: Problem-solving-based training.

Contexto problemático:

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

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

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

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

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

Fuerzas:

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

Solución:

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

Los pasos para aplicar el enfoque basado en problemas son:

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

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

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

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

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

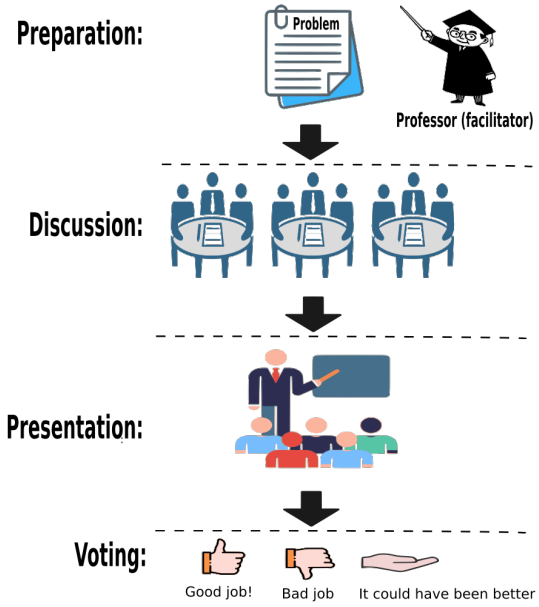

Figura 4-7.: Problem-solving-based training (Por ejemplo, Architecture katas)

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

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

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

5. Finalmente, viene la *votación*. El resto de equipos votan a partir de la presentación:

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

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

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos

Deseables:

- Sistemas operativos
- Sistemas Distribuidos
- Ingeniería de Software

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

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

Los requerimientos de este sistema son:

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

Contexto adicional:

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

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

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

Competencias opcionales: C03.

Variantes: Ninguna

Ventajas:

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

software.

Desventajas:

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

Patrones relacionados:

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

Experiencias reportadas del uso del patrón:

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

4.3.7. Patrón 7: Games-based training

Nombre: Games-based training.

Contexto problemático:

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

de ingeniería real [19].

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

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

Fuerzas:

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

Solución:

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

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

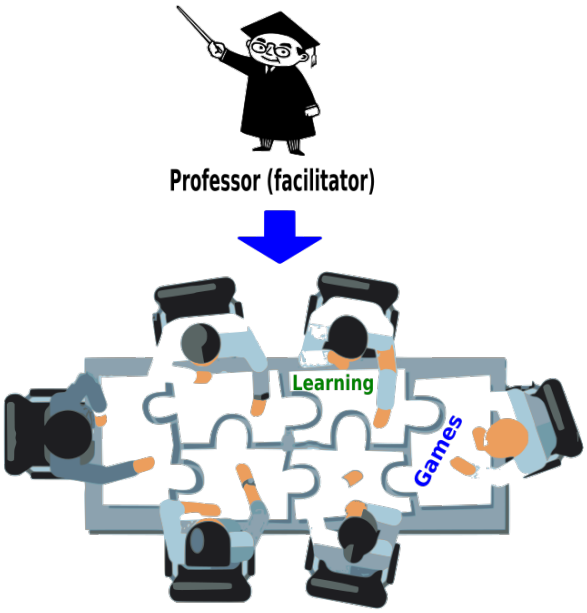

Figura 4-9.: Games-based training

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

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

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

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

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

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

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

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

Requisitos previos de los estudiantes:

Obligatorios:

- Programación Orienta a Objetos
- Diseño de bases de datos

Deseables:

- Sistemas operativos
- Sistemas distribuidos
- Ingeniería de software

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

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

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

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

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

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

Competencias obligatorias: C01, C08, C18.

Competencias opcionales: Ninguna.

Variantes para llevarlo a la práctica: Ninguna

Ventajas:

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

Desventajas:

- No se interactúa con sistemas reales.

Patrones relacionados:

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

Experiencias reportadas del uso del patrón:

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

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

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

4.4. Lenguaje de patrones

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

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

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

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

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

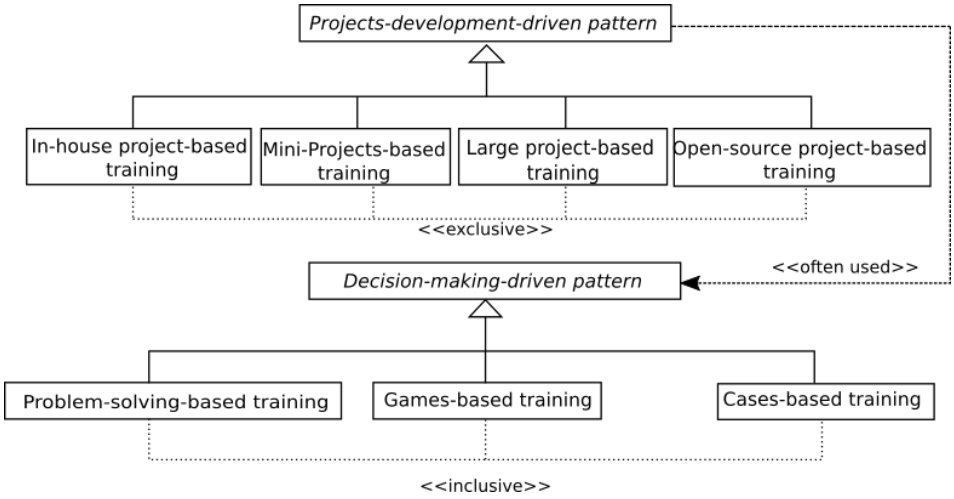

Figura 4-10.: Lenguaje de patrones de formación de AS

4.5. Guía de diseño de cursos de AS

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

A continuación explicamos cada uno de los pasos.

4.5.1. Paso 1: Preparar el curso

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

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

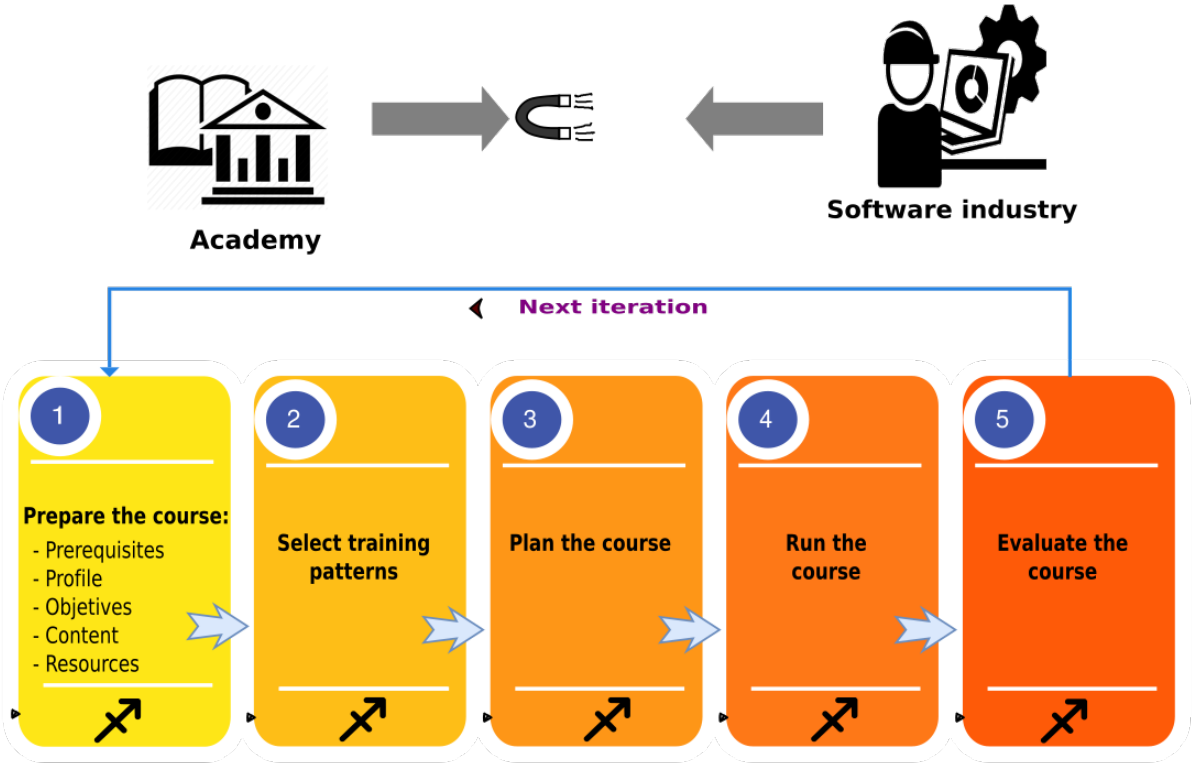

Figura 4-11.: Los cinco pasos de la Guía SAGITA

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

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

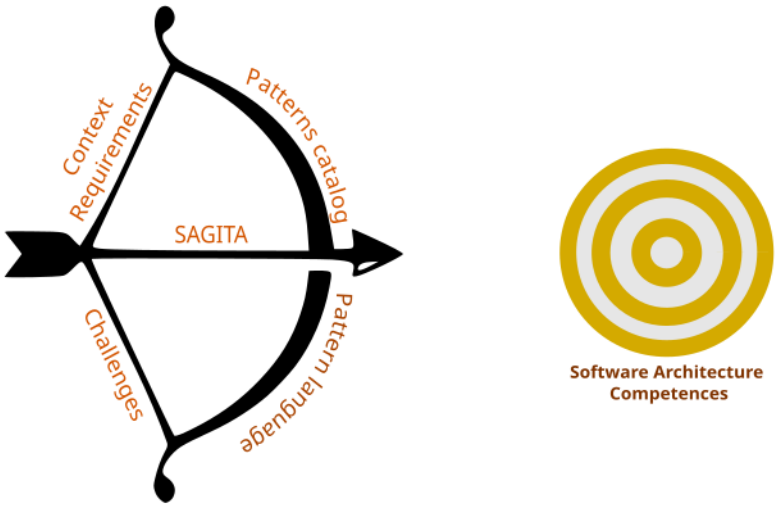

Figura 4-12.: SAGITA representa la dirección necesaria para alcanzar las competencias en arquitectura de software

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

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

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

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

la libertad para tomar y modificar elementos de este listado.

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

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

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

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

4.5.3. Paso 3: Planificar el curso

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

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

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

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

4.5.4. Paso 4: Ejecutar el curso

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

4.5.5. Paso 5: Evaluar el curso

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

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

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

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

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

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

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

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

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

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

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

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

Semana No	Fecha	Temática	Actividades	Observación

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