UNIVERSIDAD NACIONAL DE INGENIERÍA

SÍLABO

INFORMACIÓN GENERAL

ASIGNATURA	:ARQUITECTURA DE SOFTWARE
DOCENTE	:GERSON GARRIDO
HORAS POR SEMANA	:06 (TEORÍA: 04, LABORATORIO: 02)

SUMILLA

El curso es de naturaleza teórico-práctica y estudia las decisiones de un sistema que cuesta cambiar: en qué partes se divide, cómo se comunican esas partes, dónde viven los datos y qué pasa cuando algo falla. Esas decisiones las determinan los atributos de calidad (*quality attributes*), y el curso enseña a decidir: escribir lo que se necesita de forma que se pueda medir, comparar alternativas, elegir una, dejar escrito por qué y comprobar con una medición que la decisión sirvió. Se organiza en cuatro partes: **fundamentos y atributos de calidad** (qué es la arquitectura, qué requisitos la determinan, cómo se escribe y se mide un atributo de calidad, y cómo se parte un sistema en módulos); **estilos, tácticas y decisiones** (estilos monolíticos y distribuidos, tácticas para cada atributo, y cómo se documenta y se evalúa una decisión: ADR, C4, arc42, ATAM); **datos y sistemas distribuidos** (replicación, sharding, cache, transacciones entre servicios, mensajería y consistencia, vistos como decisiones con costos); y **fallas, operación y evolución** (resiliencia y observabilidad, fitness functions, deuda de arquitectura, cómo se parte un monolito y la arquitectura de sistemas con modelos de lenguaje). El sistema del curso es Tinkuy, una plataforma de venta de entradas para conciertos y partidos escrita para el curso en TypeScript con Node.js 26, PostgreSQL, Redis y NATS. Empieza como un monolito pequeño que funciona; durante el semestre le llegan requisitos de calidad que ese diseño no aguanta, y los estudiantes lo rediseñan seis veces.

COMPETENCIAS

- Explica qué es la arquitectura de un sistema, distingue sus estructuras (módulos, componentes y conectores, asignación) e identifica los requisitos que la determinan.
- Escribe atributos de calidad como escenarios medibles, los prioriza en un utility tree y los mide en un sistema que corre (latencia por percentiles, throughput, disponibilidad).
- Divide un sistema en módulos con límites explícitos, razona sobre su acoplamiento y su cohesión, y hace que las reglas de dependencia se comprueben de forma automática.
- Compara estilos de arquitectura monolíticos y distribuidos, y elige tácticas para modificabilidad, rendimiento, disponibilidad y seguridad, diciendo qué se gana y qué se paga con cada una.
- Documenta una arquitectura con vistas (C4, arc42), registra sus decisiones como ADR y la evalúa frente a sus escenarios con un método como ATAM.
- Diseña el manejo de los datos de un sistema distribuido (replicación, sharding, cache, transacciones, sagas y mensajería) y explica qué garantías de consistencia ofrece cada diseño.

- Diseña para las fallas (timeouts, retries, circuit breakers, load shedding), observa un sistema en operación con SLOs y comprueba el diseño inyectando fallas.
- Hace evolucionar una arquitectura: la protege con fitness functions, reconoce la deuda de arquitectura, extrae un servicio de un monolito y relaciona la estructura del sistema con la de los equipos.

UNIDADES DE APRENDIZAJE

Unidad	Contenido	Semanas
I	Fundamentos y atributos de calidad	1–3
II	Estilos, tácticas y decisiones	4–8
III	Datos y sistemas distribuidos	9–13
IV	Fallas, operación y evolución	14–16

Los textos principales son *Software Architecture in Practice*, para la primera mitad del curso, y *Designing Data-Intensive Applications*, para la segunda. Ninguno de los dos es gratuito: por eso cada semana indica sus capítulos y, además, lecturas libres con las que se puede seguir el curso.

PROGRAMACIÓN SEMANAL

Cada semana comprende dos sesiones de teoría de 2 horas (A y B) y una sesión de laboratorio de 2 horas. Cada capítulo de los apuntes indica qué secciones se leen antes de clase, cuáles se explican en clase y cuáles quedan para el repaso, junto con las lecturas de la guía de la semana. Las sesiones marcadas con (*) pueden pasar a lectura autónoma si se pierden clases por feriados.

Sem.	Sesión A	Sesión B	Laboratorio
1	Presentación del curso. Qué es la arquitectura: estructuras y decisiones que cuesta cambiar	Qué determina una arquitectura: requisitos significativos, restricciones y stakeholders	Entorno; recorrer Tinkuy y dibujar lo que hay. *P0 publicado*
2	Atributos de calidad y escenarios de seis partes	Medir y priorizar: percentiles, disponibilidad, utility tree y trade-offs	Escribir escenarios para Tinkuy y medirlos con k6
3	Modularidad: ocultamiento de información, cohesión y acoplamiento	Dependencias: ciclos, inversión de dependencias, ports y adapters	Reglas de dependencia con dependency-cruiser; romper un ciclo. *Entrega P0, publicado P1*
4	Estilos monolíticos: capas y modular monolith	Pipeline y microkernel. Big ball of mud (*)	Un medio de pago como plug-in
5	Estilos distribuidos: service-based y microservicios; las falacias de la computación distribuida	Event-driven: broker y mediator. Cómo se elige un estilo	El mismo caso en dos estilos, comparado con escenarios. *Entrega P1, publicado P2*
6	Tácticas de modificabilidad y de rendimiento	Tácticas de disponibilidad y de seguridad	Pool de conexiones y cache: medir antes y después
7	Decidir y documentar: ADR, vistas, C4 y arc42	Diseñar y evaluar: diseño guiado por atributos y ATAM. Risk storming (*)	Un ATAM reducido sobre Tinkuy
8	Repaso	**Examen parcial**	— *Entrega P2, publicado P3*

Sem.	Sesión A	Sesión B	Laboratorio
9	Trade-offs en los sistemas de datos: transaccional y analítico, un nodo y muchos	Modelos de datos, formatos y evolución de esquemas y contratos	Cambiar un contrato sin romper a quien lo usa
10	Replicación con un líder: síncrona y asíncrona, retraso y failover	Varios líderes y sin líder: conflictos y quórums. Dynamo (*)	Una réplica de PostgreSQL: retraso, read-your-writes y failover
11	Sharding: por rango, por hash, rebalanceo y hot spots	Cache y escalabilidad: cache-aside, invalidación, estampida, servicios sin estado	Cache-aside con Redis en Tinkuy y una estampida. *Entrega P3, publicado P4*
12	Transacciones y niveles de aislamiento: cómo se vende dos veces un asiento	Transacciones entre servicios: two-phase commit, sagas, idempotencia y outbox	No vender el mismo asiento dos veces, de tres maneras
13	Mensajería: colas y logs, garantías de entrega y orden	Event sourcing y CQRS. Consistencia y consenso para quien diseña	Outbox y consumidor idempotente con NATS JetStream
14	Resiliencia: timeouts, retries, circuit breaker, bulkhead y load shedding	Observabilidad: logs, métricas, trazas y SLOs	Una pasarela de pagos lenta y caída, con Toxiproxy. *Entrega P4, publicado P5*
15	Evolución: fitness functions, deuda de arquitectura, strangler fig y ley de Conway	Arquitectura de sistemas con modelos de lenguaje: RAG, agentes, guardrails, costo y latencia	Fitness functions para Tinkuy
16	Casos: la arquitectura de sistemas reales. El rol de quien hace arquitectura (*)	Presentaciones finales	Defensa oral del proyecto. *Entrega P5*
—	**Examen final**, en el período de exámenes		

METODOLOGÍA

- **Teoría:** clases expositivas con discusión de casos reales, ejercicios en clase (escenarios, grafos de dependencias, comparación de estilos, ADRs) y demostraciones en vivo sobre Tinkuy. Toda decisión se presenta con al menos una alternativa y con lo que cuesta. El estudiante lee antes de cada semana las secciones de los apuntes que indica su ruta de lectura, y después las lecturas de la guía semanal.
- **Laboratorio:** prácticas guiadas sobre Tinkuy, en un contenedor común para todos los estudiantes, con Node.js 26, PostgreSQL 18, Redis, NATS con JetStream, k6 para las pruebas de carga, Toxiproxy para inyectar fallas de red y dependency-cruiser. En cada práctica se mide antes y después de cambiar el diseño.
- **Proyecto:** un solo sistema, Tinkuy, que se rediseña seis veces. Cada entrega trae un requisito de calidad nuevo, escrito como escenario con su medida, que el diseño anterior no cumple. El equipo decide cómo cambiar la arquitectura, lo implementa, lo documenta (un ADR por decisión y los diagramas C4 actualizados) y comprueba el escenario con las pruebas públicas: reglas de dependencia, pruebas de carga con umbrales e inyección de fallas. Cada entrega debe seguir pasando las pruebas de las anteriores. P0 es individual; para P1–P5: [confirmar] (parejas o equipos, y si se mantienen todo el semestre).
- **Inteligencia artificial:** lo que se puede usar en el proyecto, en los laboratorios y en los exámenes está en la sección *Uso de herramientas e inteligencia artificial*.
- Los términos técnicos se usan en inglés cuando así se dicen y así aparecen en la bibliografía y

en la documentación (trade-off, sharding, circuit breaker, broker). Los que de verdad se usan en español (atributo de calidad, disponibilidad, replicación, cola) se presentan en español, con su equivalente en inglés la primera vez.

Proyecto: Tinkuy, un sistema que se rediseña seis veces

N.º	Entrega	Lo que agrega	Semanas
P0	Reconocimiento	Leer el monolito inicial: diagramas C4 de contexto y de contenedores, tres escenarios de calidad medidos con k6 y una regla de dependencia que ya no se viola (individual)	1–3
P1	Módulos con límites	Modular monolith con límites que se comprueban en cada build, y un medio de pago nuevo agregado como plug-in sin tocar el núcleo	3–5
P2	La apertura de ventas	La venta de un concierto bajo carga: latencia p95 dentro del umbral y ninguna entrada vendida dos veces, con las tácticas elegidas justificadas en ADRs	5–8
P3	Lecturas que escalan	Una réplica de lectura y una cache: el catálogo responde aunque el principal esté ocupado, y quien compra ve su compra de inmediato	8–11
P4	Pagos como servicio	Los pagos salen del monolito a un servicio propio: saga de compra, outbox, mensajes con NATS y consumidores idempotentes	11–14
P5	Cuando algo falla	La pasarela de pagos lenta o caída no tumba la venta: timeouts, circuit breaker y load shedding comprobados con Toxiproxy, SLOs, fitness functions y el documento de arquitectura final, con defensa oral	14–16

USO DE HERRAMIENTAS E INTELIGENCIA ARTIFICIAL

[confirmar] La tabla siguiente es la propuesta por defecto; cada fila la decide el docente antes de publicar el curso.

Herramienta	Proyecto y laboratorios	Exámenes
Diagnósticos del compilador y del intérprete (`tsc`, los errores de Node.js) y herramientas que revisan el programa (`depcruise`)	[confirmar]	[confirmar]
Documentación oficial (Node.js, PostgreSQL, Redis, NATS, las herramientas del curso) y búsqueda en la web	[confirmar]	[confirmar]
Autocompletado del editor que completa nombres y líneas, sin generar soluciones enteras	[confirmar]	[confirmar]
Corrector ortográfico, traductor y herramientas de accesibilidad	[confirmar]	[confirmar]
Formateadores (el formateador del editor)	[confirmar]	[confirmar]
Inteligencia artificial generativa para explicar conceptos, errores o código, sin que escriba lo entregado	[confirmar]	[confirmar]
Inteligencia artificial generativa que escribe código, pruebas, ADRs o diagramas	[confirmar]	[confirmar]

Declaración. [confirmar] Qué se escribe y dónde cuando se usa una herramienta permitida que debe declararse.

Explicación oral. A cualquier estudiante se le puede pedir que explique de forma oral cualquier parte de lo que entregó su equipo: qué decisión se tomó, qué alternativas se descartaron y cómo se comprobó. La defensa oral de la semana 16 es una de esas ocasiones, pero no la única.

BIBLIOGRAFÍA

Textos principales

1. Bass, L., Clements, P. y Kazman, R. (2021). *Software Architecture in Practice* (4.ª ed.). Addison-Wesley. https://www.sei.cmu.edu/library/software-architecture-in-practice-fourth-edition/
2. Kleppmann, M. y Riccomini, C. (2026). *Designing Data-Intensive Applications* (2.ª ed.). O'Reilly. https://dataintensive.net/

Textos complementarios

3. Richards, M. y Ford, N. (2025). *Fundamentals of Software Architecture* (2.ª ed.). O'Reilly. https://fundamentalsofsoftwarearchitecture.com/
4. Ford, N., Richards, M., Sadalage, P. y Dehghani, Z. (2021). *Software Architecture: The Hard Parts*. O'Reilly.
5. Nygard, M. (2018). *Release It!* (2.ª ed.). Pragmatic Bookshelf. https://pragprog.com/titles/mnee2/release-it-second-edition/
6. Ford, N., Parsons, R., Kua, P. y Sadalage, P. (2022). *Building Evolutionary Architectures* (2.ª ed.). O'Reilly. https://evolutionaryarchitecture.com/
7. Beyer, B. y otros (2016). *Site Reliability Engineering*. O'Reilly. Disponible gratuitamente en línea: https://sre.google/sre-book/table-of-contents/

Recursos en línea

- El modelo C4: https://c4model.com/; arc42: https://arc42.org/; los ADR: https://adr.github.io/
- Kazman, R., Klein, M. y Clements, P. *ATAM: Method for Architecture Evaluation*. Reporte del SEI: https://www.sei.cmu.edu/library/atam-method-for-architecture-evaluation/
- *Cloud Design Patterns*, Azure Architecture Center: https://learn.microsoft.com/en-us/azure/architecture/patterns/
- Joshi, U. *Patterns of Distributed Systems*: https://martinfowler.com/articles/patterns-of-distributed-systems/
- Kleppmann, M. Apuntes y videos del curso de sistemas distribuidos de la University of Cambridge: https://www.cl.cam.ac.uk/teaching/2627/ConcDisSys/
- *The Architecture of Open Source Applications*: https://aosabook.org/en/
- Curso 17-423 *Software System Design*, Carnegie Mellon University: https://cmu-swdesign.github.io/
- Curso *Software Architecture*, TU Delft: https://se.ewi.tudelft.nl/delftswa/
- Curso CS 446 *Software Design and Architectures*, University of Waterloo: https://pengyunie.github.io/cs446-1261/
- Sitio del curso: guías semanales, apuntes, diapositivas, código y enunciados.