

ACTIVIDAD

CONTENIDO TEMÁTICO DE UNIDAD I

SEPTIMO CUATRIMESTRE

**ING. DESARROLLO Y GESTION DE
SOFTWARE MULTIPLATAFORMA**

PRESENTA:

T.S.U. MONTES RODRIGUEZ OLIVER EDSON

MATERIA:

**ARQUITECTURAS DE
SOFTWARE**

CATEDRATICO:

LIC. JUAN IBARRA PEREZ

JALPAN DE SERRA, QRO.

02 SEPTIEMBRE DE 2025

INDICE

¿Quién soy?	3
Objetivo de la asignatura.....	4
Competencias de asignatura	4
Hoja de asignatura.....	5
Ponderaciones	6
PROGRAMA DE UNIDAD I: FUNDAMENTOS DE ARQUITECTURAS DE SOFTWARE	7
Definición del concepto de arquitectura de software	8
Características de las Arquitecturas de Software	8
Frameworks de Arquitectura de Software.....	9
Identificación de Requerimientos Funcionales y No Funcionales	12
Estilos de Arquitecturas de Software	13
UNIDAD II	19
UNIDAD III	20
CONCLUSIONES	21
BIBLIOGRAFIA.....	22

¿Quién soy?

Mi nombre es Oliver Edson Montes Rodríguez, pero regularmente las personas solo se dirigen a mí por mi primer nombre, yo nací aquí en Jalpan de Serra, Qro, Actualmente cuento con 22 años de edad y me encuentro cursando la carrera de Ingeniería en Desarrollo y Gestión de Software Multiplataforma.

Yo elegí esta área de estudios, ya que desde pequeño siempre me ha llamado la atención todo lo relacionado con el área de tecnologías, y en algún momento poder trabajar en algo relacionado y que realmente me gustara y en posteriormente tener un gran impacto en la sociedad con los proyectos o trabajos que llegué a realizar.

Yo soy una persona muy activa, por lo que yo practico varios deportes, uno de mis favoritos era el basquetbol, aunque actualmente ya no la practico con tanta frecuencia como lo hacía antes, ya que ahora practico con más Frecuencia el levantamiento de pesas al fallo muscular, pero más adelante en algún momento quiero volver a jugar básquet.

Objetivo de la asignatura

El alumno establecerá arquitecturas de software con base en el análisis de requerimientos para satisfacer los atributos de calidad del software y servir como guía en el desarrollo

Competencias de asignatura

Construir soluciones de software seguro y sistemas inteligentes mediante la dirección y el liderazgo en la gestión de proyectos, integrando metodologías y arquitecturas de desarrollo para la optimización de proyectos de investigación, innovación, desarrollo tecnológico y emprendimiento, bajo la normatividad aplicable.

Hoja de asignatura



INGENIERÍA EN DESARROLLO Y GESTIÓN DE SOFTWARE
EN COMPETENCIAS PROFESIONALES



ASIGNATURA DE ARQUITECTURAS DE SOFTWARE

1. Competencias	Construir soluciones de software seguro y sistemas inteligentes mediante la dirección y el liderazgo en la gestión de proyectos, integrando metodologías y arquitecturas de desarrollo para la optimización de proyectos de investigación, innovación, desarrollo tecnológico y emprendimiento, bajo la normatividad aplicable.
2. Cuatrimestre	Séptimo
3. Horas Teóricas	21
4. Horas Prácticas	54
5. Horas Totales	75
6. Horas Totales por Semana Cuatrimestre	5
7. Objetivo de aprendizaje	El alumno establecerá arquitecturas de software con base en el análisis de requerimientos para satisfacer los atributos de calidad del software y servir como guía en el desarrollo

Unidades de Aprendizaje	Horas		
	Teóricas	Prácticas	Totales
1. Fundamentos de arquitecturas de software	10	5	15
2. Modelado de arquitecturas de software	5	25	30
3. Patrones de diseño	6	24	30
Totales			75

ELABORÓ:	Comité de Directores de la Carrera de Ingeniería en Desarrollo y Gestión de Software	REVISÓ:	Dirección Académica	
APROBÓ:	C. G. U. T. y P.	FECHA DE ENTRADA EN VIGOR:	Septiembre de 2020	

Ponderaciones

SABER 40%

SABER HACER 50%

SER Y CONVIVIR 10%


PROGRAMA DE UNIDAD I: FUNDAMENTOS DE ARQUITECTURAS DE SOFTWARE

ARQUITECTURAS DE SOFTWARE

UNIDADES DE APRENDIZAJE

1. Unidad de aprendizaje	1. Fundamentos de arquitecturas de software
2. Horas Teóricas	10
3. Horas Prácticas	5
4. Horas Totales	15
5. Objetivo de la Unidad de Aprendizaje	El alumno determinará la arquitectura de software de acuerdo a requerimientos para guiar la construcción de los componentes de software.

Temas	Saber	Saber hacer	Ser
Introducción a las arquitecturas de software	Definir el concepto de arquitectura de software Describir las características las arquitecturas de software		Analítico razonamiento Deductivo uso De Procesos Cognitivos razonamiento Lógico
Frameworks de arquitecturas	Describir los frameworks de arquitectura de software.		Analítico razonamiento Deductivo uso De Procesos Cognitivos razonamiento Lógico
Estilos de arquitectura de software	Identificar requerimientos Funcionales y No funcionales Distinguir los estilos de arquitecturas de software - Cliente-Servidor - Microservicios - N capas - Orientada a servicios - Dirigida a eventos - Basada en espacio - Microkernel - Serverless	Establecer la arquitectura de software cumpliendo con los requerimientos funcionales y no funcionales.	Analítico razonamiento Deductivo uso De Procesos Cognitivos razonamiento Lógico

ELABORÓ:	Comité de Directores de la Carrera de Ingeniería en Desarrollo y Gestión de Software	REVISÓ:	Dirección Académica	
APROBÓ:	C. G. U. T. y P.	FECHA DE ENTRADA EN VIGOR:	Septiembre de 2020	

Definición del concepto de arquitectura de software

La arquitectura de software constituye la organización fundamental de un sistema de software, definida por sus componentes principales, las relaciones entre dichos componentes, el entorno en el que operan, y los principios rectores que guían su diseño y evolución posterior. Esta definición, basada en el estándar IEEE/ISO/IEC 42010, establece que la arquitectura trasciende la mera codificación para convertirse en el blueprint conceptual que determina las características de calidad del sistema resultante.

La arquitectura de software se distingue del diseño detallado por su nivel de abstracción y alcance de impacto. Mientras el diseño detallado se enfoca en algoritmos específicos y estructuras de datos particulares, la arquitectura establece decisiones de alto nivel que afectan múltiples componentes y determinan atributos sistémicos como escalabilidad, seguridad y mantenibilidad.

Características de las Arquitecturas de Software

Las arquitecturas de software se caracterizan por cuatro elementos constitutivos fundamentales:

- **Componentes:** Unidades computacionales que encapsulan funcionalidades específicas del sistema. Pueden representar módulos, servicios, objetos o cualquier elemento que realice procesamiento computacional definido.
- **Conectores:** Mecanismos que facilitan la interacción y comunicación entre componentes. Incluyen protocolos de comunicación, interfaces de

programación (APIs), buses de mensaje y cualquier medio que permita intercambio de datos o control.

- **Configuración:** La topología estructural que describe cómo los componentes y conectores se relacionan para formar el sistema completo. Define la organización espacial y temporal de los elementos arquitectónicos.
- **Restricciones:** Limitaciones y directrices que gobiernan el diseño, implementación y evolución del sistema. Incluyen restricciones tecnológicas, de rendimiento, de seguridad y de negocio.

Frameworks de Arquitectura de Software

Los frameworks de arquitectura proporcionan metodologías estructuradas y herramientas conceptuales para el desarrollo sistemático de arquitecturas de software empresariales. Estos marcos de trabajo establecen procesos, artefactos y mejores prácticas que guían a los arquitectos en la toma de decisiones complejas.

TOGAF (The Open Group Architecture Framework)

TOGAF representa uno de los frameworks más comprehensivos para el desarrollo de arquitecturas empresariales. Su metodología central, el Architecture Development Method (ADM), proporciona un enfoque iterativo que abarca ocho fases principales:

- **Fase Preliminar:** Preparación y iniciación del esfuerzo arquitectónico
- **Fase A - Visión de Arquitectura:** Establecimiento del alcance y expectativas

- **Fase B - Arquitectura de Negocio:** Desarrollo de la arquitectura de procesos de negocio
- **Fase C - Arquitectura de Sistemas de Información:** Diseño de arquitecturas de datos y aplicaciones
- **Fase D - Arquitectura Tecnológica:** Definición de la infraestructura tecnológica
- **Fase E - Oportunidades y Soluciones:** Planificación inicial de implementación
- **Fase F - Planificación de Migración:** Desarrollo del plan detallado de implementación
- **Fase G - Gobierno de Implementación:** Supervisión de la implementación
- **Fase H - Gestión de Cambios de Arquitectura:** Mantenimiento y evolución continua

El Enterprise Continuum de TOGAF proporciona un repositorio de activos arquitectónicos reutilizables, incluyendo modelos de referencia, patrones arquitectónicos y componentes específicos de industria.

Zachman Framework

El Zachman Framework establece una taxonomía para organizar la complejidad arquitectónica mediante una matriz bidimensional. Las seis perspectivas (filas) representan diferentes puntos de vista de stakeholders:

- **Planificador (Contextual):** Establece el alcance y motivaciones del sistema
- **Propietario (Conceptual):** Define conceptos de negocio y requerimientos
- **Diseñador (Lógico):** Especifica modelos independientes de tecnología
- **Constructor (Físico):** Desarrolla modelos considerando restricciones tecnológicas
- **Implementador (As Built):** Produce componentes configurados y desplegados
- **Operador:** Opera y mantiene el sistema en producción

Las seis interrogantes (columnas) abordan aspectos fundamentales: Qué (datos), Cómo (función), Dónde (red), Quién (personas), Cuándo (tiempo), Por qué (motivación).

SABSA (Sherwood Applied Business Security Architecture)

SABSA extiende principios arquitectónicos tradicionales para abordar sistemáticamente la seguridad de sistemas. Su estructura en seis capas proporciona un marco integrado:

- **Capa Contextual:** Define el ambiente de riesgo y objetivos de seguridad de negocio

- **Capa Conceptual:** Desarrolla la arquitectura de seguridad de alto nivel
- **Capa Lógica:** Especifica servicios y mecanismos de seguridad
- **Capa Física:** Define la implementación tecnológica de controles de seguridad
- **Capa de Componente:** Detalla productos y configuraciones específicas
- **Capa Operacional:** Establece procesos de operación y mantenimiento de seguridad

Identificación de Requerimientos Funcionales y No Funcionales

Los requerimientos funcionales especifican las funcionalidades, servicios y comportamientos que el sistema debe proporcionar a sus usuarios. Estos requerimientos describen "qué" debe hacer el sistema en términos de procesos de negocio, casos de uso y interacciones usuario-sistema.

Los requerimientos no funcionales, también denominados atributos de calidad, especifican "cómo" debe comportarse el sistema en términos de características operacionales:

- **Rendimiento:** Tiempos de respuesta, throughput, latencia
- **Escalabilidad:** Capacidad de manejar incrementos en carga de trabajo
- **Disponibilidad:** Tiempo operativo del sistema (uptime)
- **Seguridad:** Protección contra amenazas y vulnerabilidades

- **Usabilidad:** Facilidad de uso y experiencia del usuario
- **Mantenibilidad:** Facilidad para realizar cambios y correcciones
- **Portabilidad:** Capacidad de operar en diferentes entornos

Estilos de Arquitecturas de Software

Cliente-Servidor

Este estilo establece una separación clara entre entidades que solicitan servicios (clientes) y entidades que proporcionan servicios (servidores). Características principales:

- Distribución de responsabilidades entre cliente y servidor
- Comunicación mediante protocolos bien definidos (HTTP, TCP/IP)
- Escalabilidad vertical mediante fortalecimiento del servidor
- Simplicidad conceptual y facilidad de implementación

Ventajas: Separación clara de responsabilidades, facilidad de mantenimiento, escalabilidad vertical directa

Desventajas: Punto único de falla, limitaciones de escalabilidad horizontal, potencial cuello de botella en el servidor

Microservicios

Arquitectura que descompone aplicaciones en servicios pequeños, independientes y autónomos, cada uno ejecutándose en su propio proceso y comunicándose mediante mecanismos ligeros.

- Servicios pequeños enfocados en funcionalidades específicas de negocio
- Despliegue independiente de cada servicio
- Descentralización de datos y gobernanza
- Tolerancia a fallos mediante aislamiento de servicios

Ventajas: Escalabilidad granular, independencia tecnológica, tolerancia a fallos, equipos autónomos

Desventajas: Complejidad operacional, overhead de comunicación, consistencia de datos distribuida

N Capas (N-Tier)

Organización del sistema mediante estratificación horizontal donde cada capa proporciona servicios a la capa superior y consume servicios de la capa inferior.

- Separación lógica en capas de presentación, negocio y datos
- Comunicación unidireccional entre capas adyacentes
- Modularidad y reutilización de componentes
- Abstracción de complejidad mediante niveles

Ventajas: Modularidad, separación de responsabilidades, reutilización, facilidad de testing.

Desventajas: Potencial degradación de rendimiento, rigidez estructural, overhead de comunicación.

Orientada a Servicios (SOA)

Paradigma arquitectónico que organiza funcionalidades como servicios interoperables y reutilizables con interfaces bien definidas.

- Servicios como unidades funcionales autónomas
- Contratos de servicio explícitos y versionados
- Bus de servicios empresariales (ESB) para integración
- Reutilización de servicios a través de múltiples aplicaciones

Ventajas: Reutilización de servicios, interoperabilidad, flexibilidad de integración

Desventajas: Complejidad de gobernanza, overhead de ESB, dependencias entre servicios

Dirigida a Eventos (Event-Driven)

Paradigma donde los componentes se comunican mediante la producción y consumo de eventos, estableciendo un acoplamiento temporal y espacial mínimo.

- Comunicación asíncrona mediante eventos

- Productores y consumidores desacoplados
- Patrones publish-subscribe y event sourcing
- Procesamiento de eventos en tiempo real

Ventajas: Desacoplamiento extremo, escalabilidad masiva, tolerancia a fallos

Desventajas: Complejidad de debugging, consistencia eventual, gestión compleja de flujos

Basada en Espacio (Space-Based)

Arquitectura que elimina la base de datos central distribuyendo datos y procesamiento a través de múltiples nodos de procesamiento.

- Eliminación de cuellos de botella de base de datos
- Distribución de datos mediante grids en memoria
- Escalabilidad horizontal sin límites teóricos
- Tolerancia a fallos mediante replicación

Ventajas: Escalabilidad extrema, tolerancia a fallos, rendimiento alto

Desventajas: Complejidad de implementación, costo de infraestructura, consistencia de datos

Microkernel (Plugin)

Arquitectura que proporciona funcionalidad mínima en un núcleo central, extendiendo capacidades mediante plugins o extensiones.

- Núcleo mínimo con funcionalidad esencial
- Plugins proporcionan características específicas
- Sistema de registro y descubrimiento de plugins
- Flexibilidad y extensibilidad del sistema

Ventajas: Flexibilidad extrema, extensibilidad, isolación de funcionalidades

Desventajas: Complejidad de desarrollo de plugins, overhead de comunicación

Serverless

Paradigma donde las funciones constituyen la unidad de despliegue y ejecución, con gestión automática de infraestructura por parte del proveedor cloud.

- Funciones como unidad de despliegue
- Ejecución dirigida por eventos
- Escalabilidad automática y transparente
- Modelo de costos basado en uso real

Ventajas: Escalabilidad automática, costo eficiente, simplicidad operacional

Desventajas: Vendor lock-in, limitaciones de tiempo de ejecución, cold start latency

UNIDAD II

UNIDAD III

CONCLUSIONES

BIBLIOGRAFIA