
—

[Introducción]:

-Atributos de calidad: No funcionales (rendimiento, etc).

-EQUIPOS: Lider-scrum master, PO, QA tester, Dev Ops, Desarrolladores [front, back yd atos] (la arquitectura depende de todo el equipo).

—

TAREAS DEL DÍA DE HOY:

-RECOMENDACIÓN:

-Hacker rank (plataforma para ver pruebas de trabajo).

-Para la siguiente semana (realizar “quick off”?).

-Que es arquitectura? (cualquier terminologia).

- **Arquitectura (General):** La arquitectura es el arte y la ciencia de diseñar y construir edificios y otras estructuras físicas. Involucra aspectos como la planificación, el diseño, la construcción y la decoración de espacios habitables y funcionales.
- **Arquitectura (Histórica):** La arquitectura es una disciplina que estudia la evolución de los estilos y técnicas constructivas a lo largo del tiempo, analizando cómo las diferentes culturas y épocas han influido en el diseño y la construcción de edificios y estructuras.
- **Arquitectura (Paisajística):** La arquitectura paisajística se enfoca en el diseño de espacios exteriores, como jardines, parques y áreas urbanas, integrando elementos naturales y construidos para crear entornos armoniosos y funcionales.
- **Arquitectura (Urbanística):** La arquitectura urbanística se centra en la planificación y el diseño de ciudades y comunidades, considerando aspectos como la infraestructura, el uso del suelo, la movilidad y la sostenibilidad para mejorar la calidad de vida de los habitantes.
- **Arquitectura de Software:** La arquitectura de software es la estructura fundamental de un sistema de software, compuesta por sus componentes, las relaciones entre ellos y las propiedades y principios que guían su diseño y evolución. Se enfoca en aspectos como la modularidad, escalabilidad, seguridad y rendimiento del software, asegurando que el sistema cumpla con los requisitos funcionales y no funcionales.

—

[Arquitectura de Nosotros]:

-La arquitectura que nosotros conocemos se basa en “UML”.

-ESTRUCTURA, INTERACCIÓN Y COMPORTAMIENTO.

-ESTRUCTURA : Paquete, Desplieque, Clase.

-DATOS:

-Pagina (Especificación de UML - Última Versión 2.5): “https://www.uml-diagrams.org/”

-***INGENIERÍA DE SOFTWARE:***

-Disciplina que aplcica métodos, procesos y gestiona un ciclo de vida para desarrollar software profecionalmente, dentro de este ciclo de software existe la programación (PERO NO SON SINONIMOS).

-PROCESO DEL DESARROLLO DE SOFTWARE (PASOS A SEGUIR):

1. Levantar requerimientos (Análisis de requerimientos)
2. diseñar (Diseño de la solución)
3. implementar (Desarrollo y Evaluación)
4. probar (Despliegue).
5. Mantenimiento y Evolución.

(Cada vuelta se llama “ITERACIÓN”).

-MÉTODOLOGÍAS / PARADIGMAAS DEL DESARROLLO DE SOFTWARE: (CASCADA, ESPIRAL, DESARROLLO ITERATIVO E INCREMENTAL, DESARROLLO ÁGIL) - CAMBIA LA MANERA EN LA QUE SE RECORRE EL CICLO

-Tradicional: Cascada (Heredado de Ingenierías tangibles como la civil).

-Agil: SCRUM / CANVA / XP [no es agil si no toca repetir algo como los requerimientos, si se hacen en varias veces si es agil] (Genera valor, entrega al usuario un producto que le permite hacer cosas - SE PUEDE IR DISELLANDO DE A MONOLITOS O A MICROS, pero en cada iteración se genera valor y que sirva para sus necesidades).

-Primero se identifican de primeras los DATOS, luego los USUARIOS (Diagramas de proceso).

-SCRUM PASO A PASO:

-1. Quick Off (lanzamiento del proyecto / idea / delcaración de la visión, que se va a hacer).

-2. Product Back Log: Las tareas priorizadas (las técnicas son las que siempre se priorizan y luego se prioriza segun desee el cliente).

-3. Sprint Backlog: Paquete de tareas que van a entrar en el Sprint (a veces se define en 2 →grumin y Planning).

-Grumin?: Los seniors son los que suelen estár ahí xd.

-Planning: En el Planning se puntua, que tan dificil es la historia de usuario / Asignar segun en la experiencia de las personas.

-4. Retrospective: Al final del Sprint se hace una reunín “RETROSPECTIVE” del equipo donde se hace un análisis de

-5. Reviews: Mostrar el valor que le damos en ese Sprint. Es la última que se hace en el SPRINT.

- Los Sprints duran de 1 a 6 semanas / Reuniones diarias que son los “Daily” → Cada reuníon tiene un entregable.
- El proyecto muere y pasa a OPERACIÓN.

-DIFICULTADES: (TOMAR DESICIONES CON RESPECTO A COSAS ESENCIALES O ACCIDENTALES)

-Esenciales: Especificación, diseño y comprobación de concepto (COSAS DE NEGOCIO, DISEÑO Y PROBA DE CONCEPTO, o sea probar si encaja en lo que necesito para elegir un framework, etc → SABER QUE NECESITO?) - INGENIERÍA.

-Va relacionado con temas de Conformidad, Invisibilidad (Es intangible para cliente y usuarios), Tolerancia al cambio y complejidad.

-1. Complejidad (No desarrollar o usaer software libre (pruebas d econcepto).

-2. Conformidad (Mockups).

-3. Tolerancia al cambio (Desarrollo evolutivo).

-4. Invisibilidad (Grandes diseñadores / ARQUITECTOS - Grandes diseños modelando correctamente TODO).

-Accidentales: Detalles de la implementación y producción actual (Elegir un lenguaje, herramienta, etc) - TECNOLOGÍA.

-Nuevas tecnologías, entornos de programacion, lenguajes de alto nivel.

-1. ().

-2. ().

-3. ().

-4. ().

CICLO DE VIDA DEL SOFTWARE:

-Disparadores: Mejora del sistema (bug, control de excepciones, etc / Nueva necesidad de usuario como una nueva caracteristica al sistema).

-Análisis de Requerimientos: (Descubrimiento / Análisis y documentación, tradicional).

-Comprensión del problema. (Comprenderlo por completo).

-Diseñar la solución (Plantear la abstracción del problema y el modelado de la nueva caracteristica). - ALTO NIVEL (Modulos funcionales - no tan especifico)

-Solución detallada (). - BAJO NIVEL (Requerimientos funcionales - más especifico).

-Desarrollo y evaluacion (programación e integración)

-Artefacto de softeware (nueva version del software)

-Despliegue

-Software Dispoble para que lo usen los usuarios.

-Mantenimiento y evolución (Arreglo de errores / Nuevas funcionaldiades).

-SISTEMA DEPRECADO (Al final se volverá obsoleto y lo que se requerirá es volverlo a construir).

TAREAS DEL DÍA DE HOY:

-Que es Arquitectura Empresarial y cuales son las funciones de los roles de (arquitecto de software, scrum master, PO (product Owner), QA, tester, desarrollador, y DevOps).

Arquitectura Empresarial en Ingeniería de Sistemas (CUMPLIR CON LOS REQUERIMIENTOS NO FUNCIONALES - BUENAS PRACTICAS ALINEADAS CON LA TOMA DE DECISIONES DE LA EMPRESA).

-PILARES: Gestión de Datos - Gestión de Infraestructura, …, y Proceso de negocio (para que se aplica el software) → Criterio fundamental: SEGURIDAD.

Definición

La Arquitectura Empresarial (AE) en el contexto de la ingeniería de sistemas es un enfoque que integra todos los aspectos de la organización, desde procesos de negocio hasta infraestructura tecnológica, para asegurar que las operaciones y estrategias de la empresa estén alineadas con sus objetivos y sean eficientes. En ingeniería de sistemas, la AE se centra en el diseño, implementación, gestión y evolución de sistemas complejos dentro de la organización.

Arquitecto de Software

Funciones:

- **Diseño del Sistema**: Crear y definir la arquitectura del software asegurando que cumpla con los requisitos funcionales y no funcionales.
- **Selección de Tecnologías**: Elegir las tecnologías y herramientas adecuadas para el desarrollo del proyecto.
- **Documentación**: Proveer documentación detallada sobre la arquitectura, diagramas y decisiones técnicas.
- **Mentoría y Soporte**: Asistir y guiar a los desarrolladores durante la implementación.
- **Evaluación y Mejora Continua**: Revisar y mejorar la arquitectura para mantener la calidad y escalabilidad del sistema (Pearson ActiveLearn).

Scrum Master - FACILITADOR (EXITO DEL PROYECTO).

Funciones:

- **Facilitador de Scrum**: Facilitar las reuniones diarias, planificaciones de sprint, revisiones y retrospectivas.
- **Remoción de Obstáculos**: Identificar y eliminar impedimentos que afecten el progreso del equipo.
- **Coaching y Educación**: Entrenar al equipo en prácticas ágiles y asegurar la adherencia a los principios de Scrum.
- **Mediador**: Actuar como intermediario entre el equipo y otros stakeholders para proteger la dinámica del equipo (Pearson ActiveLearn).

Product Owner (PO) - VISIÓN DE PRODUCTO (DEFINE CONDICIONES PARA EL PRODUCTO Y CARCTERISTICAS, ORGANIZA EL BACKLOG “HISTORIAS DE USUARIO”) → Saber de cosas técnicas.

Funciones:

- **Gestión del Product Backlog**: Definir y priorizar el backlog del producto, asegurando que el equipo trabaje en las historias de usuario más importantes.
- **Visión del Producto**: Mantener y comunicar la visión del producto al equipo y a los stakeholders.
- **Aceptación del Trabajo**: Revisar y aceptar o rechazar el trabajo completado por el equipo durante el sprint.
- **Interacción con Stakeholders**: Recoger y gestionar las expectativas de los stakeholders, asegurando que sus necesidades se reflejen en el backlog (Pearson ActiveLearn) (ActiveLearn).

QA (Quality Assurance) - DE ACUERDO A LAS HISTORIAS DE USUARIO GENERAN LOS ESCENARIOS DE CALIDAD, DISEÑAN PRUEBAS (carga y estrés, integración, end to end).

Funciones:

- **Planificación y Estrategia de Pruebas**: Desarrollar estrategias y planes de pruebas para garantizar la calidad del producto.
- **Ejecución de Pruebas**: Realizar pruebas funcionales, de integración, de regresión, de rendimiento, y otras.
- **Automatización de Pruebas**: Implementar y mantener suites de pruebas automatizadas.
- **Análisis y Reporte de Defectos**: Identificar, documentar y seguir defectos y problemas encontrados durante las pruebas.
- **Colaboración**: Trabajar estrechamente con desarrolladores y otros miembros del equipo para asegurar la calidad desde las primeras etapas del desarrollo (Pearson ActiveLearn).

Tester (APLICA LA PRUEBA REALIZADA DEL QA).

Funciones:

- **Ejecución de Pruebas**: Realizar pruebas manuales o automatizadas en las aplicaciones para encontrar defectos.
- **Creación de Casos de Prueba**: Diseñar y documentar casos de prueba basados en los requisitos del software.
- **Registro de Defectos**: Documentar y reportar errores y problemas encontrados durante las pruebas.
- **Regresión**: Ejecutar pruebas de regresión para asegurar que los cambios recientes no afecten negativamente al sistema (Pearson ActiveLearn) (ActiveLearn).

Desarrollador

Funciones:

- **Escribir Código**: Implementar funcionalidades según los requisitos definidos.
- **Depuración y Corrección de Errores**: Identificar y corregir errores en el código.
- **Colaboración**: Trabajar en equipo con otros desarrolladores, QA y stakeholders para cumplir con los objetivos del proyecto.
- **Revisión de Código**: Participar en revisiones de código para asegurar la calidad y coherencia del mismo.
- **Documentación**: Mantener la documentación técnica del código y las soluciones implementadas (Pearson ActiveLearn) (Pearson ActiveLearn).

DevOps (MANTENER TODA LA INFRA, GESTIONAR EL REPO, RESPONSABLES DE LA INTEGRACIÓN Y DESPLIEGUE) → RECOMIENDA USAR DOCKER Y USAR PERFILES DE AMBIENTE.

Funciones:

- **Automatización**: Desarrollar y mantener pipelines de CI/CD (Integración Continua/Despliegue Continuo).

- **Monitorización y Mantenimiento**: Configurar y monitorizar infraestructuras, servidores y servicios.
- **Colaboración**: Trabajar en estrecha colaboración con desarrolladores y equipos de operaciones para mejorar los procesos de desarrollo y despliegue.
- **Seguridad y Cumplimiento**: Implementar prácticas de seguridad y asegurar el cumplimiento con normativas y estándares.
- **Escalabilidad y Desempeño**: Asegurar que los sistemas sean escalables y funcionen de manera eficiente (Pearson ActiveLearn) (ActiveLearn).

[Prueba Unitaria]:

-Crear escenario, aplicar prueba y destruir el escenario

TAREAS DEL DÍA DE HOY:

-Buscar el Ciclo de vida de los datos:

-Ciclo de vida de infraestructura, que es proceso de negocio.

El ciclo de vida de los datos es el proceso que describe las etapas que atraviesan los datos desde su creación hasta su eliminación. Este ciclo incluye una serie de fases que aseguran que los datos se gestionen de manera efectiva y segura a lo largo de su existencia. A continuación se describen las etapas principales del ciclo de vida de los datos:

1. **Creación y Captura**:
    - **Generación de datos**: Los datos se crean a través de diversas fuentes, como entradas manuales, dispositivos IoT, sensores, transacciones, formularios en línea, etc.
    - **Captura**: Los datos se recopilan y se almacenan en una base de datos u otro sistema de almacenamiento.
2. **Almacenamiento**:
    - **Almacenamiento temporal**: Los datos pueden almacenarse temporalmente en cachés o memoria temporal antes de su procesamiento.
    - **Almacenamiento a largo plazo**: Los datos se guardan en bases de datos, data lakes, servidores, nubes, etc., para su uso posterior.
3. **Procesamiento**:
    - **Limpieza y transformación**: Los datos se limpian para eliminar errores, duplicados y valores nulos. Luego se transforman en un formato adecuado para el análisis.
    - **Enriquecimiento**: Los datos pueden combinarse con otros conjuntos de datos para agregar valor.
4. **Análisis**:
    - **Análisis descriptivo**: Se examinan los datos para entender qué ha sucedido (e.g., estadísticas descriptivas, visualizaciones).
    - **Análisis predictivo**: Se utilizan modelos predictivos para anticipar futuros eventos o comportamientos.

    - **Análisis prescriptivo**: Se generan recomendaciones basadas en el análisis de datos.
5. **Distribución y Uso**:
    - **Acceso y consulta**: Los usuarios acceden a los datos a través de consultas, informes y dashboards.
    - **Aplicación**: Los datos se utilizan para tomar decisiones, desarrollar productos, mejorar procesos, etc.
6. **Mantenimiento y Monitoreo**:
    - **Actualización**: Los datos se actualizan para reflejar cambios y nuevas entradas.
    - **Monitoreo**: Se supervisa el uso de los datos para asegurar su calidad y cumplimiento con las políticas de seguridad y privacidad.
7. **Archival**:
    - **Archivar**: Los datos que ya no son necesarios para el uso inmediato se archivan para un almacenamiento a largo plazo, conservándolos para futuros usos o requisitos legales.
8. **Eliminación**:
    - **Borrado seguro**: Los datos se eliminan de forma segura y definitiva cuando ya no son necesarios, asegurando que no puedan ser recuperados por medios no autorizados.
    - **Cumplimiento normativo**: La eliminación se realiza en cumplimiento con las leyes y regulaciones aplicables.

Este ciclo de vida garantiza que los datos sean gestionados de manera eficiente, segura y conforme a las normativas, desde su creación hasta su eliminación final.

[PO INFORMACIÓN]:

-Coordinarnos con Majo (la jefa), nosotros gestionamos el backlog.

- Le podemos preguntar al profe al inicio y al final de clase (y pues en Whatsapp también, NO ESRIBIR FINES DE SEMANA, FESTIVOS NI MUY TARDE COMO 11 DE LA NOCHE).

-Para una idea de proyecto: “Distribuido, componente en .Net y .Java (El ciclo de diseño es complejo, pero el desarrollo es hasta ciereto)”.

[Proceso de Arquitectura]:

1. METODOLOGÍAS TRADICIONALES:

    -Definición del problema (Entender el contexto de lo que me están pidiendo).

    -Requerimientos (funcionales/son el qué y no funcionales/son el cómo).

    -Restricciones (Cosas que a uno lo condicionan, pensada en integrarse con lo actual).

    -Riesgo (Se desea mitigarlos y un plan de acción cuando se materialice → Como el

presupuesto, gestión del tiempo, etc).

2. METODOLOGÍAS ÁGILES:

   -Planning.

   -Manejo de Backlog.

   -Retros.

   -Reviews.

-CONSEJOS:

-Tomar una decisión y llevarla si o si a cabo (No estár coambiando porque si a cada rato / Contemplar varias cosas) → Realizar Pruebas de Concepto.

-Realizar muchos diagramas (USAREMOS EL MODELO C4 / Otro que se podría es el “4+1”, pero se vuelven 20k diagramas y mantenerlos es complejo).

-Patron: Solución preestablecida para problemas recurrentes (Acceso a datos,).

-Facilitador: Scrum Master.

-Arquitectura:

-UTILIDAD (Ser útil y funcionar bien), DURABILIDAD (Mantener, probar y desarrollar facil), ELEGANCIA (Agradable).

-Arquitectura vs Diseño: Se enfoca en estructuras de alto nivel meintras que el diseño es … (TODA ARQUITECTURA CONLLEVA DISEÑO, PERO NO TODO DISEÑO ES ARQUITECTURA)

-TIPOS DE ARQUITECTURA: (todo lo engloba la empresarial, por debajo se encuentra)

-1. Empresarial: (Alinea Técnologia, datos, enfraestructura para apoyar todo el proceso de negocio / O sea que la empresa sea eficiente con lo propueso) → Togaf y ZACU.

-Proceso: Arquitectura de software (todo el proceso de datos) / 2. Arquitectura de Negocios (…) / 3. Arquitectura Apps (sistemas heredados en la empresa que soporta todo el procesamiento de la información) / 4. Arquitectura Tecnológica o Infraestructura (Manejo de servidores, licencias, protcolos de comunicación, etc) /// En todos los pilares habrá oportunidad y solucioes, Portafolio de proyectos y gobierno de implementación.

-2. Sistemas: (…).

-3. Software: (…).

-El software que diseño suele tener la misma estructura en el lugar en donde se trabaja.

-Revisar en algun momento: “Resumeme la arquitectura de ISO/IEC/IEEE 42010:2011, 3.2”.

-COMPONENTES FUNCIONALES DE ARQUITECTURA:

-Elementos de software.

-Relaciones entre ellos.

-Propiedades de ambos.

[Documentos]: (nosotros vemos como lo adaptamos):

1. Gestión.
    1. Anteproyecto: (Problematica, objetivos, alcances y limitaciones).
    2. Acta de Inicio (Nosotros no debemos hacerlo).
    3. Presupuesto (Se agregan cada cosa aunque no cueste: “ClickUp = 0”, como la nube javeriana).
    4. Acta de Cierre.
    5. Proyecto academico (Objetivo que teníamos como estudiantes al enfrentarnos a realizar este proyecto, las conclusiones relacionadas a los objetivos, lexiones aprendidas → SOLO SE HACE ACÁ EN LA MATERIA, NO SE HACE NORMALMENTE).
2. Arquitectura. (SE DIVIDEN EN “VISTAS”).
    1. SAD/DOCUMENTO DE ARQUITECTURA DE SOFTWARE (Versión ejecutiva / de alto nivel → Punto de entrada de los documentos) y debe contener lo siguiente:
        1. Drivers (Horizontes que marcan las decisiones arquitectonicas / Son cosas que no se puede elegir hacer, si el cliente las pide como una caracteristica, se debe de realizar).

    2. Modulos Funcionales (NO SE PONEN TODOS LOS REQUERIMIENTOS NI TODAS LAS HISTORIAS DE USUARIO → SE PONE SOLO UNA PARTE DEL NEGOCIO).
    3. Modulos No Funcionales (Acá van los “atributos de calidad”).
    4. Riesgos y Restricciones.
    5. Acuerdos (Acá se ponen las decisiones arquitectonicas).
    6. Modelos (HDL) → Cada uno debe tener una descripci´n y las tácticas y estrategias que nosotros implementamos y eso se ve en el siguiente punto de “Patrones”.
        1. Vsita de Software (Es más artisticos).
        2. Vista de Datos (Relación a un modelo conceptual de datos / Modelo Entidad Relación).
        3. Vista Física.
        4. Visata de Negocio (Diagramas de proceso de negocio).
    7. Justificación.
        1. Tácticas.
        2. Estrategias.
        3. Tecnologias.
        4. Estilos.
        5. Patrones.
2. Infraestructura (Describir en detalle las tecnicas usadas y porque fueron escogidas) → Bajo nivel.
    1. Diseño de Ambientes.
    2. Infraestructura.
    3. Aplicaciones (Dcok, etc).
    4. Herramientas de Seguridad (antivirus, cifrar o no hacerlo, etc).
3. Datos → Bajo nivel.
    1. Ciclo de Vida de los Datos (cual es el flujo de información de lo que tenemos, que herramienta lo toma, modifica, almacena, etc).
    2. Arquitectura de datos (Modelo arquitectonico que va a soportar esto, sistemas que captan información y la devuelven, pero hay algunas d etipo de analisis o sea herramientas con información regadas en otros sistemas y luego leerlos para disparar dichos procesos, hacer algo y mostrarlas).
    3. Diccionario de Datos (Metadatos de los Datos → Acronimos para que no se tan largo, donde diga que significa cada cosa “X de transacción”, no son solo los nombres de las varibales, si no sus opciones).
    4. Crendenciales y Accesos.
4. Negocio (Especificación de Requerimientos → ACÁ NO SE COPIAN LAS HISTORIAS DE USUARIO, SOLO SE TIENE UN LINK AL BACKLOG) → Bajo nivel.
    1. SRS:
        1. Descripciónn de usuarios.
        2. Modulos Funcionales.
        3. Requerimientos (Funcionales y No Funcionales).
        4. (OPCIONAL) Diagramas de casos de uso y diagramas de …
    2. QA (cumple con la funcionalidad y con los atributos de calidad):

            1. Describir el tipo de pruebas que se van a hacer (3 minimo = integración, unitarias, …).
            2. Métodología en la que lo vamos a aplciar (manual o automatica se tendrán HERRAMIENTAS, tener un flujo muy claro).
            3. Plantilla de documento de pruebas (Se va haciendo cada Sprint).
        3. Software
            1. SDD (C4 model o 4+1) → DEPENDE DEL FRAMEWORK QUE VAYAMOS A ESCOGER.
            2. Mockups
3. Soporte. (DEPENDE DE CADA UNO DE NOSOTROS SI AGRUPAMOS TODOS ESTOS DOCUMENTOS O LOS SEPARAMOS).

    1. Manuales: (No necesariamente debe ser un documento en word, puede ser en el READ ME del repositorio).
        1. Instalación (Instalación de las herramientas que necesitamos para desarrollar y desplegar).
        2. Configuración (Como se configura ESPECIFICAMENTE/PARTICULARMENTE para que funcione cada cosa como las maquinas, etc).
        3. Integración y Despliegue (Flujo de DevOps — Pipeines CI/CO → Filosofia para integrar y desplegar).
        4. Usuario (Documentación como preguntas frecuentes en “ayuda”).
    2. Configuración de Ambiente.
        1. Local (Ambiente Local de desarrollo, licencias, quien crea el uduario, etc).
        2. QA (Como es el manejo de ambiente / como usamos QA).
        3. Prod (Como es el manejo de ambiente / como usamos Producción).
    3. Lineamientos y Politicas.
    4. Herramientas.

PARTES DE TODO DOCUMENTO DE INGENIERIA:

-Portada: Nombre del documento, fecha (NO DE LA UNIVERSIDAD, NI PARA EL PROFESOR).

-Descripción: Introducción para saber que va a encontrar en el docuemento.

-Control de Cambios: La tipica tabla.

-Indice: Lo hacce automaticamente word poniendo subtitulos.

-Referencias cruzadas (agregar tituo a una imagen).

1. [Atributos de Calidad:]:

A) FUNCIONES DE UN ARQUITECTO DE SOFTWARE:

1. ANALIZA EL CONTEXTO: Acompañar a los POs en el proceso de definición de requerimientos (funcionales y no funcionales), además de ver cuales son los riesgos y restricciones que nacen.
2. POSEE HERRAMIENTAS DE DISEÑO: Contruir y mantener los atributos de calidad (requerimientos no funcionales construidos en un modelo de calidad y se definen los escenarios de calidad), estilos de arquitectura (lo general), patrones de arquitectura (lo especifico), tácticas/estrategia (Para cumplir un atributo de calidad existe una táctica y es lo que evoluciona más rápido).
3. DISEÑA LA SOLUCIÓN: Modelo, razonamientos (justificación de por qué escogió X táctica), documentación, implementación (codificación o configuración).

B) SKILLS QUE DEBE TENER UN ARQUITECTO DE SOFTWARE:

1. Tomar decisiones arquitectonicas (la indecisión cuesta).
2. Analizar continuamente la arquitectura.
3. Estar al día de las tendencias actuales (sin forzar usarlas).
4. Asegurar cumplimiento decisiones existentes.
5. Experiencia diversa.
6. Conocimiento del dominio de negocio (entender el contexto y problematica en la que estamos trabajando).
7. Poseer habilidades interpesonales.
8. Comprender y navegar en política empresarial.

C) COMO IDENTIFICAR A LOS INVOLUCRADOS:

1. INTERNA / CONSTRUCCIÓN DEL PRODUCTO: Analistas, diseñadores.
2. EXTERNOS / QUIENES USARAN EL PRODUCTO.

D) PRINCIPIOS SOLID:

1. S = Single Responsibility Principle (SRP).
    - Una clase debería tener una y solo una razón para cambiar (Que los métodos no sean muy largos) → Si no se cumple, el mantenimiento es muy dificil.
2. O = Open/Closed Principle (OCP).
    - Poder agregar cosas a una clases sin cambiar lo que ya existía → (nunca quitar funcionalidades, pero si agregar cosas).
3. L = Liskov Substitution Principle (LSP).
    - Si se tienen clases que heredan de otras, deben tener coherencia con la padre (saber tener la clase padre general y las otras ya mas especifficas) → Se busca hacer una reutilización de codigos y especificacion de comportamientos.
4. I = Interface Segregation Principle (ISP).
    - Haz interfaces que sean especficicas para un tipo de cliente, es decir para una finalidad concreta.
5. D = Dependency Inversion Principle (DIP).
    - Cuando se vaya a usar un objeto, no instanciarlo a partir de la clase, si no a partir de la interfaz → .

E) CARACTERISTICAS DE POO:

- Abstracción.
- Modularidad
- Encapsulamiento.
- Polimorfismo.
    - Sobrecarga de Métodos (el único que no necesita herencia ya que se hace dentro de la misma clase).
    - Sobrescritura (Cuando el apdre tiene definido un método y para cambiar su comportamiento se llama igual pero cambia la lógica dentro - y para identificar cual estoy usando uso “SUPER / THIS”).
    - Ligadura Dinamica / Enlace Dinamico (En tiempo de ejecución se puede mandar un hijo o nieto al padre, cuando se instancia un objeto se puede usar la constructora de un padre para instanciar a un hijo).
- Principio de Ocultación.
- Herencia.
- Recolector de Basura.

F) ACOPLAMIENTO Y COHESIÓN:

- **-** Acoplamiento (Dependencia → Se busca tener dependencias BAJAS, o sea no tener cosas obligatorias).
- **+** Cohesión (El grado en que elementos diferentes de un sistema permanecen unidos para alcanzar un mejor resultado / Integración → Se busca tener cohesión ALTA integración).

---

2. [PATRONES DE SOFTWARE]:

- Los patrones de diseño son modelos muestra, que sirven como guía para buscar soluciones a problemas comunes en el desarrollo de software y otros ámbitos del diseño de interacción o interfaces.

- Patrones arqutecturales:

    - Monolito (Broker pattern, presentation abstraction control, reflection, pipes and filters) → Más rápido para desarrollar.
        - Eficiencia.
        - Curva de aprendizaje.
        - Capacidad de prueba.
        - Capacidad de modificación.
    - Distribuidos (model view controller, microkernel pattern, layers pattern, blackboard pattern) → Más completo.
        - Es Modular.
        - Uso de recursos menor.
        - Adaptabilidad.
        - Disponibilidad.

- Estilo: Vision general.
- Patron: Vision especifica.
- Proceso de Arquitectura:

- MÉTODOLOGÍA “ATAM”: Para validar una arquitectura y gestionar los cambios de la arquitectura.
    - Ciclo que TODOS los Sprints se deberían de hacer (Lo naranja primero, el verde, finalmente de allí salen los azules y se repite dicho proceso de arquitectura).
- 
- Proceso de Arquitectura:

- Modelos de Calidad: (El celeste es el modelo actual).

-Atributos de Calidad #1:

- **Introducción a los Atributos de Calidad:**
    - **Definición:** Los atributos de calidad son las propiedades del sistema que afectan cómo cumple con los requisitos no funcionales (vienen de). Son esenciales para evaluar aspectos como rendimiento, seguridad y mantenibilidad.
    - **Clasificación:** Generalmente se dividen en atributos operacionales (como rendimiento y seguridad) y atributos de evolución (como mantenibilidad y escalabilidad).
- **Tipos de Atributos de Calidad:**
    - **Disponibilidad:**

        - **Definición:** La capacidad del sistema para estar operativo y disponible para su uso en cualquier momento.
        - **Ejemplo:** Un sistema con alta disponibilidad tiene mecanismos para evitar y recuperar rápidamente de fallas.
    - **Desempeño:**
        - **Definición:** Se refiere al tiempo de respuesta del sistema bajo diferentes condiciones de carga.
        - **Aspectos clave:** Incluye latencia, throughput, y uso eficiente de recursos.
    - **Seguridad:**
        - **Definición:** Protección del sistema contra amenazas internas y externas.
        - **Componentes:** Autenticación, autorización, cifrado, y auditoría.
    - **Mantenibilidad:**
        - **Definición:** Facilidad con la que se pueden hacer cambios al sistema, como corrección de errores o mejoras.
        - **Medición:** Mediante métricas como el tiempo medio de reparación (MTTR).
    - **Usabilidad:**
        - **Definición:** Facilidad con la que los usuarios pueden interactuar con el sistema.
        - **Elementos:** Incluye interfaz de usuario, accesibilidad, y documentación de ayuda.
- **Escenarios de Calidad:**
    - **Concepto:** Son descripciones específicas de cómo un sistema debe comportarse bajo ciertas condiciones, permitiendo medir un atributo de calidad.
    - **Estructura del Escenario:** Involucra un estímulo (lo que sucede), un entorno (contexto donde sucede), y una respuesta (cómo debería reaccionar el sistema).
    - **Ejemplos:** Escenario de disponibilidad podría incluir una situación donde un servidor falla y la expectativa es que el sistema cambie automáticamente a un servidor de respaldo.
- **Tácticas de Calidad:**
    - **Definición:** Acciones o estrategias que se implementan para garantizar que un atributo de calidad sea satisfactorio.
    - **Ejemplos:**
        - **Redundancia:** Implementada para mejorar la disponibilidad.
        - **Control de acceso:** Usado para fortalecer la seguridad.
        - **Caching:** Aplicado para mejorar el rendimiento.
        - **Modularidad:** Facilita la mantenibilidad al permitir cambios localizados.
- **ISO/IEC 25010 Modelo de Calidad:**
    - **Descripción:** Un estándar internacional que define y categoriza atributos de calidad en software.
    - **Características Clave:**
        - **Eficiencia:** Uso eficiente de los recursos del sistema.

        - **Compatibilidad:** Capacidad del sistema para operar en distintos entornos.
        - **Fiabilidad:** Confianza en el correcto funcionamiento del sistema.
        - **Portabilidad:** Facilidad con la que el software puede ser transferido a otro entorno.
    - **Evaluación:** Utiliza métricas específicas para cada atributo, permitiendo medir la calidad del software en cada una de estas dimensiones.

-Atributos de Calidad #2:

- **Revisión de Atributos de Calidad Clave:**
    - **Desempeño y Escalabilidad:**
        - **Desempeño:** Evalúa tiempos de respuesta y procesamiento bajo carga.
        - **Escalabilidad:** Capacidad del sistema para manejar un aumento en la carga de trabajo sin degradar el desempeño. Se discuten tipos de escalabilidad (horizontal y vertical).
    - **Disponibilidad:**
        - **Métricas:** Incluyen tiempo de actividad, tiempo medio entre fallos (MTBF), y tiempo medio para recuperación (MTTR).
    - **Seguridad:**
        - **Aspectos Evaluados:** Autenticación, autorización, cifrado, y auditorías regulares.
    - **Mantenibilidad:**
        - **Evaluación:** Se mide la facilidad de implementar cambios, corregir errores y realizar actualizaciones. Se mencionan métricas como el tiempo necesario para implementar un cambio y la modularidad del código.
    - **Portabilidad:**
        - **Importancia:** Capacidad de trasladar el sistema entre diferentes entornos o plataformas.
        - **Ejemplo:** Sistemas multiplataforma que pueden operar en Windows, Linux, y macOS.
- **Métricas para Evaluar Atributos de Calidad:**
    - **Definición y uso de métricas específicas para cada atributo.**
    - **Métricas de Desempeño:**
        - **Latencia:** Tiempo que tarda una solicitud en ser procesada.
        - **Throughput:** Número de transacciones procesadas en un tiempo determinado.
    - **Métricas de Seguridad:**
        - **Tasa de detección de intrusiones:** Proporción de ataques detectados y manejados.
        - **Número de vulnerabilidades identificadas:** Usualmente evaluadas en pruebas de penetración.
    - **Métricas de Mantenibilidad:**
        - **Complejidad ciclomática:** Una métrica que mide la complejidad del código fuente, influyendo en la mantenibilidad.
    - **Métricas de Usabilidad:**

        - **Tiempo para completar tareas:** Medición del tiempo que toma a los usuarios realizar tareas específicas usando el sistema.
- **Estudio de Caso o Ejemplos Prácticos:**
    - **Aplicación de Tácticas de Calidad en Proyectos Reales:**
        - **Caso de Disponibilidad:** Implementación de servidores redundantes y sistemas de recuperación ante desastres en una aplicación crítica.
        - **Caso de Seguridad:** Uso de encriptación y autenticación multifactor para proteger datos sensibles en un sistema financiero.
    - **Aprendizajes Clave:** Discusión de cómo se aplicaron las tácticas y el impacto que tuvieron en la calidad del sistema.

-Atributos de Calidad #3:

- Atributos de Calidad (AC): Se describen como las expectativas del usuario sobre el rendimiento y funcionamiento de un producto de software. Ejemplos de estos atributos incluyen rendimiento, escalabilidad, modificabilidad, interoperabilidad, seguridad, disponibilidad, usabilidad, y otros, que están alineados con la norma ISO/IEC 25000:2014.
- Tensión entre Atributos: Se discute cómo los AC no son independientes entre sí, y cómo algunos pueden entrar en conflicto. Por ejemplo, un sistema altamente seguro puede ser menos usable. Este tipo de conflicto es común en la arquitectura de software, y es crucial decidir qué atributos priorizar en función del proyecto y los stakeholders.
- Acuerdos (Trade-offs): Las decisiones arquitectónicas importantes a menudo implican compromisos que afectan los AC, la estructura, las dependencias, las interfaces y las técnicas de construcción. Una herramienta para gestionar estos compromisos es la "Matriz de puntos de acuerdo por AC".
- Gestión de Atributos de Calidad: Se enfatiza que no es posible satisfacer todos los AC al 100%, ya que mejorar un atributo puede afectar negativamente a otros. Aquí es donde entran los "trade-offs". Además, se menciona el método ATAM (Architecture Trade-Off Analysis Method) como una estrategia para evaluar y evolucionar la arquitectura.
- Evolución de Arquitectura: Se destaca la importancia de la evaluación y evolución continua de la arquitectura del sistema para mantener y mejorar los AC a lo largo del tiempo.
- Escenarios y Tácticas: Se introduce el concepto de escenarios y tácticas como herramientas para diseñar sistemas que respondan a estímulos específicos que afectan los AC. Cada escenario se clasifica según el AC que aborda, como disponibilidad, mantenibilidad, eficiencia de ejecución, seguridad, capacidad de prueba y usabilidad.
- Caso de Estudio: Finalmente, se presenta un caso de estudio que aplica los conceptos discutidos, demostrando cómo identificar y manejar los AC, escenarios de calidad y acuerdos en un proyecto real.

[CLASE DE HOY]: (EN EL SAD DEBEN ESTAR TODAS ESTAS COSAS)

- Atributos de Calidad: Vienen de los Requerimientos no funcionales.

- Proceso de requerimientos:

    - ESPACIO DEL PROBLEMA: Que queremos resolver? como identificamos si estamos resolviendo el problema? Espectativas de cada uno de los usuarios?
    - SOLUCIÓN: Diseño, desarrollo, evaluación (el profe es muy riguroso con esta), criterios de aceptación, despliegue.
- Clasificación de los requerimientos:

    - Proyecto: Recursos, certificaciones, infraestructura, plan de despliegue, acuerdos de servicio, capacitaciones, documentación de Usuario, Licencias, Plan de Transición.
    - Producto:
        - Dominio: de negocio, usuario, función (normas, leyes, lineamientos de la empresa).
        - Software: Requerimientos funcionales (lo que debe de hacer) y no funcionales (condiciones en la que debe funcionar esa app).
            - Descripción de modulos funcionales (meta requerimientos) → O sea la descripción global de las funcionalidades (que se hace, que se espera, como se debe de hacer).

- Riesgos:
    - En la toma de requerimientos: (Telefono roto = Dificultad / Complejidad).
    - En los atributos de calidad: (Incertidumbre, cuanto mas incertidumbre, mas alto es el riesgo → Toca ser lo mas detallado para reducir la incertidumbre).
    - Conocimiento del Dominio (Riesgo prorípico, son aquellos que podemos atacar de forma estándar → O sea no conocer ni saber el tema de la empresa → Toca tomarse el tiempo para aprender del tema, toca validar los mockups con el cliente).
- Restricciones: Pueden haber restricciones de tecnología, la empresa, etc / Tambien pueden ser dadas por el ciclo de vida del software.
    - Por ejemplo en nuestro caso son los componentes .Net, etc.

    - Cuando se tienen claras las restricciones se puede hacer la elección tecnologica y el resto de herramientas de manera correcta.
- Atributos de Calidad: Las caracteristicas medibles de interes para usuarios o desarrolladores / La calidad es el proceso de medición de caracteristicas que se debería de cumplir.
    - Axioma de Calidad: Lo que no se mide, no se controla / Lo que no se controla, no se mejora.
    - Pueden especificarse mediante escenarios (nosotros medimos el modelo, pero debemos decir que los escenarios de calidad para cada atributo de calidad, donde se tendrá una métrica y prioridad ya que algunos atributos entrarán en tensión y el que suele pelear siempre es el de seguridad).
    - Que nos va a determinar estos atributos?: Condicionan el estilo.
    - La que se usa actualmente es la celeste.

- Iso 2500 no es la única que hay de Modelos de calidad de Software.
    - Modelos de calidad de Proceso: (ITIL, 15504, PSP, COBIT, ISO 90003, CMMI, ISO/IEC 20000, etc).
    - Modelo de caldiad del Producto: (McCallm GQM, Boehm, FURPS+, Gilb, ISO 9126-1, SQAE, WebQEM, iSO 25000).
- Modelo de Calidad de Software (FURPS+): (No se usa en la industria).

- Modelos de calidad de Software (ISO/IEC 25000:2014): (Esta si se usa en la industria).

- Los 10 principios de funcionalidad de “Jacob Nielsen”.

[Recorderis]:

-Arquitectura de Alto nivel: Cualquiera debe poder entenderlo (no tiene sintaxis). Es una estructura que define como el esquema general.

[Estilos y Patrones]:

- Podemos entregar aplicaciones:
- Puertos: Servicios de sistemas operativos (se consumen a traves de la red).
- Terminal: Pura (con argumentos de entrada) / La otra que no copié xd.
- Menú de interacción: Aplicaciones V1 previas a la versión gráfica (Excel de los 90s).
- Escritorio/Pantalla Grafica: Mouse, táctil, mandos, o robots de ingeniería industrial (maquinas con botones y palancas).
- Web: Sitios web, página web, aplciaciones web (hibridas, web-apps o sea un solo contenedor web, PWA).
- HCI: Voz, Gestos, Imagen, (como con Alexa → Comandos por voz, etc / Kinect).
- Realidad Aumentada: Muy Inmersivo.
- Máquinas Industriales M2M: Como las IOTs, protocolos para que las maquinas operen solas sin intervención humana.
- Vehículos: Los tipicos carros.

-[Cada una de estas aplicaciones cumple DIFERENTES ATRIBUTOS DE CALIDAD/PATRON/ARQUITECTURAS/ETC. (todo por entregarla en un mecanismo diferente)].

-Tipos de Ambiente para el software:

- Locales (Se trabaja en el mismo dispositivo) → PC, dispositivos, móviles.
    - Aplicaciones de escritorio, Videojuegos, Sistemas Operativos, Procesos Bach, Aplicaciones Portables, Aplicaciones de configuración.
- En Redes (Hay privadas / Hay publicas.) → Orientados a conexión y no orientados a conexión.
    - Web, PWA, Web Apps, middleware, servicios, etc.

-Despliegue de Software:

- Mainframes.
- Servidores.
- PC.
- Dispositivos.
- Maquinas Virtuales.
- Servidores Web.
- Servidores de Aplicaciones.
- Interpretes.

- Contenedores.
- Cloud.

-Tipos de Ambientes para el Software / Entornos:

- Desarrollo (ambiente aislado y no tiene data real → CONTROL DE VERSIONES).
- Entorno de Pruebas (servidor pruebas → Servidor integración) → Se hacen pruebas: “SIT” y “UAT”.
    - A veces se divide con otro llamado “Staging” que es el entorno de ensayo.
- Entorno de Producción (Servidor producción granja servidores → Esta data se DEBE PROTEGER SI O SI).

-En que se divide cualquier arquitectura:

- Elementos, Relaciones, Restricciones. y Lista de atributos.
- Se divide en componentes y conectores.

-Componentes y conectores:

- Componentes:
    - Representan elementos del dominio y contexto, dependiendo del nivel de abstracción pueden representar (funcionalidades, sistemas, productos de software, dispositivos, datos) y pueden ser (componentes y subcomponentes).
- Conectores (su forma siempre es así):
    - Llamado a procedimiento (Llaman a una función).
    - Enlace (Linkean una función).
    - Evento (Publican un encolamiento).
    - Adaptador (Transforman una comunicación).
    - Acceso a datos (Para lectura y Para escritura).
    - Flujo de datos (Información que va avanzando entre componentes y en cada paso agrega información).
    - Arbitraje (Coordinación → En Api Gateaway se hace esto y “distribuidor”).
    - Distribuidor.
- -Representación:
    - No alcancé a copiarlos, pero es lo de las flechitas.

[Patrones y Estilos Arquitectonicos]:

-Estilos puros: En la realidad no se implementan (NUNCA SE MODIFICA UN PATRON, EL ESTILO SI SE PUEDE MODIFICAR).

-Sirven para saber que ventajas yd esventajas nos condicionan un estilo en particular.

-Todo responde a los ESCENARIOS DE CALIDAD propuestos.

-RECORDAR QUE UN ESTILO ES LA ESTRUCTURA, MIENTRAS QUE EL PATRON ES UNA SOLUCIÓN ESPECIFICA (Ej: Estilo: Microservicios y eso es un tema distribuido o sea que toca asumir muchos problemas de un problema distribuido como la escalabilidad-rendimiento-etc, y cada problema tiene una solución, los cuales son los patrones). (estilo: tipo de edificación (lo generico / la estructura) / problema: como hago para subir y bajar la gente de una plata a otra / patrón: Escalera electrica, escalera, cuerda, ascensor, etc).

-PATRONES ARQUITECTURALES:

-Gran bola de lodo: Cuando no se repeta la arquitectura o si de por si no hay (entonces se conectan entre si como se quieren las cosas y eso no cumple ningun principio ni atributos de calidad) → SE ESPERA NUNCA LLEGAR A ESTE ERROR.

-MVC: Vista, modelo, acción del usuario.

-Capas: (Suele confundirse con MVC), pero la diferencia es que las capas van unas sobre las otras mientras que MVC son relaciones entre ellas, acá en capas solo se comunican con la capa de arriba y de abajo (controlle, service, repositorio y el usuario solo conoce al controller).

-Orientado a Eventos: Sistemas distribuidos (SOA), requieren de componentes y usa un bus de eventos, es muy usado el encolamiento para el alto rendimiento en nubes.

-Microkernel: Orientado para monolitos, y si quiere ser escalable en funcionalidades se usa este.

-Comparte-Nada: