Final Preguntas Inge 2

1. **¿Qué es y para qué sirve un indicador?**

- **Definición:** Un indicador es un elemento de información que se construye combinando distintas métricas.
- **Para qué sirve:** Su función principal es establecer una "pauta" o estándar de qué se considera una medida correcta para un atributo específico. Esto le permite al gestor del proyecto realizar ajustes en el producto, el proceso o el proyecto basándose en datos concretos proporcionados por el indicador.
- **Explicación fácil:** Imagina que las métricas son los "ingredientes" (como cuántos errores hay o cuánto tiempo se tardó) y el indicador es la "receta terminada" que te dice si el plato salió bien o si le falta sal (ajuste).

pagina 18: Un indicador es un elemento de información que se forma a partir una combinación de métricas y establece una pauta de cuál es la medida correcta para un atributo. Esto permite al gestor del proyecto ajustar el producto, proceso o proyecto en función de la información brindada por el indicador.

Ubicación: Página 18 del PDF de contenido.

Un indicador es un elemento de información que se construye a partir de la combinación de varias métricas. Su función principal es establecer una pauta o referencia sobre cuál es el valor correcto o esperado para un atributo específico, lo que permite a los responsables de un proyecto realizar ajustes en el producto, el proceso o el proyecto mismo según sea necesario.

Para entenderlo mejor, existe una jerarquía que va desde el dato más simple hasta el indicador:

- Medida: Es una indicación cuantitativa bruta de una propiedad, como el tiempo transcurrido o la cantidad de líneas de código escritas.
- Métrica: Es un conjunto de reglas o fórmulas que se aplican a una medida para obtener información más significativa sobre lo que se está evaluando. Por ejemplo, en lugar de solo saber cuántas líneas hay, una métrica analiza la recopilación de esas medidas para dar un valor con mayor contexto.
- Indicador: Se obtiene al recopilar y analizar los resultados de las métricas. Mientras que la métrica te da un número procesado, el indicador te dice si ese número es bueno, malo o requiere una acción inmediata.

¿Para qué se utilizan los indicadores?

En la gestión de proyectos, los indicadores son herramientas fundamentales para que los profesionales tomen decisiones informadas. Sirven para:

- Evaluar la calidad: Ayudan a determinar si el producto cumple con los estándares esperados.

- Controlar procesos: Permiten entender y supervisar lo que ocurre durante el desarrollo y el mantenimiento del software.
- Mejorar el rendimiento: Proporcionan una visión cuantitativa que facilita la optimización continua del proyecto.

En resumen, un indicador es como una "señal" que utiliza los datos de las métricas para orientar al equipo y asegurar que se alcancen los objetivos del proyecto de manera exitosa.

2. **¿Cuáles son los requerimientos no funcionales que se ven afectados por la arquitectura?**

La arquitectura tiene un impacto directo y crítico en los siguientes requerimientos no funcionales:

- **Rendimiento:** Se optimiza agrupando operaciones críticas en pocos subsistemas para evitar comunicaciones excesivas que lo ralenticen.
- **Seguridad:** Se logra mediante arquitecturas en capas, protegiendo los datos más críticos en las capas más internas.
- **Protección:** Se busca centralizar las operaciones de protección en un solo subsistema para reducir costos y facilitar la validación.
- **Disponibilidad:** Se asegura diseñando componentes redundantes que puedan reemplazarse sin detener el sistema (tolerancia a fallos).
- **Mantenibilidad:** Requiere componentes pequeños (granularidad fina) y autocontenidos que sean fáciles de cambiar.

Ubicación: Página 66 del PDF de contenido.

Requerimiento	Acción	¿Por qué?
Rendimiento	Juntar	Menos viajes = más velocidad
Seguridad	Encapas	Más capas = Más difícil entrar.
Protección	Centralizar	Un solo guardián = Mejor control
Disponibilidad	Duplicar	Tener repuesto = No se corta el servicio.
Mantenibilidad	Achicar	Piezas chicas = fácil de cambiar

Requerimientos no funcionales: Son las características que definen la calidad y el comportamiento de un sistema. Son las características que definen **cómo** debe funcionar un sistema, en lugar de qué debe hacer. Se encargan de medir la calidad del producto. Básicamente, no se trata de qué hace el software (funciones), sino de cómo lo hace.

- El requerimiento funcional (el QUÉ): "El sistema debe dejar que el usuario inicie sesión".
- El requerimiento no funcional (el CÓMO): "El inicio de sesión debe tardar menos de 2 segundos" o "Los datos deben estar encriptados".

Piensa en adjetivos:

- Velocidad: ¿Qué tan rápido responde?
- Seguridad: ¿Qué tan protegido está?
- Usabilidad: ¿Qué tan fácil es de aprender?
- Disponibilidad: ¿Funciona las 24 horas?

Si el "qué" es que un auto te lleve a tu destino, el "cómo" (requerimiento no funcional) es que te lleve rápido, seguro y cómodo.

3. ¿Qué es un riesgo? ¿Cuáles son los ítems de más alto riesgo según Boehm?

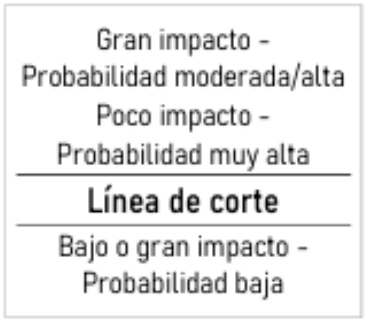
- **Definición de riesgo:** Es un evento no deseado que, de ocurrir, trae consecuencias negativas para el proyecto. Se caracteriza por tener **incertidumbre** (la probabilidad de que pase) y **pérdida** (el impacto o daño que causaría).
- **Ítems de alto riesgo según Boehm:** Si bien el PDF menciona que Boehm sugiere supervisar los **10 riesgos más altos** de la lista (ordenados por impacto y probabilidad), también destaca factores de riesgo clave como:
    - Falta de compromiso de los usuarios finales.
    - No entender por completo los requisitos.
    - Ámbito o requisitos del proyecto inestables.
    - Falta de habilidades adecuadas en el equipo.

Pagina 55:

La lista de riesgos se ordena considerando su impacto y probabilidad, y se coloca una línea de corte para establecer cuáles riesgos serán atendidos. Mientras Boehm sugiere supervisar los 10 riesgos más altos, ese número debe ser manejable y adaptarse al proyecto.

Los riesgos por encima de la línea reciben atención primaria, mientras que los debajo se reevaluarán en supervisiones y tendrán una prioridad secundaria.

Es importante equilibrar impacto y probabilidad. Riesgos de baja probabilidad, así tengan un alto impacto, no deben consumir excesivo tiempo, por lo que se establece la línea de corte sobre estos. Riesgos con probabilidad moderada o alta, con poco o mucho impacto merecen mayor atención y se ubicarán sobre la línea de corte.



Riesgo	Categoría	Probabilidad	Impacto
AAA	Proyecto	80%	2
FDS	Producto	80%	3
DFA	Negocio	75%	3
GFE	Proyecto	60%	2
GTR	Proyecto	50%	2
UID	Negocio	40%	1
HWR	Producto	20%	2

Ubicación: Páginas 51, 52 y 55 del PDF de contenido

¿A qué me refiero en el contexto de riesgo con anticipar, prevenir, evitar y minimizar?

El equipo deberia anticipar los riegos haciendo una lista de ellos, para poder prevenirlos, en el mejor de los casos evitarlos y en el peor poder minimizar su impacto MAL Se define explicitamente como **REDUIR LA PROBABILIDAD** de que el riesgo ocurra, **no** se refiere al **impacto.** El plan de contingencia es la estrategia que se usa cuando el riesgo ya ocurre (o se asume que ocurrira) y se busca mitigar las consecuencias (impacto) . **Evitar (Anular):** Es diseñar el sistema para que el riesgo **no pueda ocurrir** (probabilidad cero).

Rta corregida: El equipo debería anticipar los riesgos... para intentar **evitarlos** (anularlos), si no es posible **minimizarlos** (bajar su probabilidad) y, en el peor caso, tener un **plan de contingencia** para mitigar su impacto.

Evitar: Eliminar el riesgo por completo (Riesgo = 0).
Minimizar: Bajar la chance de que pase (Bajar Probabilidad).
Contingencia: Bajar el daño si llega a pasar (Bajar Impacto).

Anticipar: Es la responsabilidad de prever la posibilidad de que ocurran eventos no deseados durante el desarrollo o mantenimiento del proyecto. Se relaciona con una actitud proactiva de mirar hacia el futuro para identificar problemas antes de que sucedan.

Prevenir: Consiste en elaborar planes para impedir la aparición de los eventos riesgosos identificados. Busca actuar antes de que el problema se presente, en contraposición a reaccionar ante una crisis.

Evitar (Anular): Es una estrategia de planificación específica que implica diseñar el sistema de tal manera que el evento de riesgo **no pueda ocurrir** bajo ninguna circunstancia (riesgo cero).

Minimizar: Se refiere a la estrategia que busca **reducir la probabilidad** de que el riesgo ocurra.

- *Nota:* Aunque en la introducción general del texto se menciona "minimizar sus efectos negativos" , en la sección detallada de estrategias de planificación, **minimizar** se define específicamente como bajar la probabilidad, mientras que la mitigación de consecuencias (impacto) se asocia a los *Planes de Contingencia*

¿Qué tipos de riesgos hay?

	GENÉRICOS	ESPECÍFICOS
Características	Estos riesgos son independientes del dominio en el que se trabaje y pueden aplicarse a cualquier proyecto.	Estos riesgos están relacionados con el dominio específico de desarrollo de software.
Conocidos	Son riesgos que se han experimentado en otros proyectos similares.	Son riesgos que los expertos en el dominio saben que podrían ocurrir.
Predecibles	Son riesgos que son fáciles de anticipar, como cambios en los requisitos.	Son riesgos que se pueden imaginar fácilmente en el contexto del dominio.
Impredecibles	Son riesgos difíciles de prever y pueden surgir en cualquier proyecto.	Son riesgos que podrían surgir, pero son muy difíciles de anticipar.

¿Qué tipos de estrategias de gestión de riesgos hay?

Hay dos enfoques principales, estrategias proactivas y reactivas. Una gestión efectiva de riesgos combina elementos proactivos y reactivos y se adapta a las necesidades de la organización y los riesgos involucrados.

- Reactivas → Enfoque en reaccionar ante un problema una vez que ocurre, manejando la crisis en el momento. No hay prevención, solo se toma acción cuando el problema se presenta. Por ejemplo, no tomar medidas anticipadas para evitar una posible falla en un dispositivo y solucionar el problema solo cuando se rompe.
- Proactivas → Enfoque en anticipar y prevenir problemas. Se reconoce que los riesgos pueden surgir y se planifican estrategias de tratamiento para evitar que los riesgos tengan un impacto crítico. Por ejemplo, reservar recursos para posibles fallas de dispositivos, lo que ahorra tiempo y contempla los gastos potenciales.

4. ¿Cuáles son las consideraciones que se deben tener en cuenta a la hora de diseñar una interfaz?

(pag 58 diseño de interfaz Define cómo se comunicará el software tanto internamente como con sistemas externos y personas. Describe la forma en que los usuarios interactuarán con el sistema y cómo se intercambiará información con otras aplicaciones.)

El diseño de la interfaz de usuario (UI) debe centrarse en que la comunicación entre el hombre y la máquina sea eficiente y fácil de aprender. Las principales consideraciones son:

- **Adaptación al usuario:** La tecnología debe adaptarse a las personas, no al revés. Se debe considerar la diversidad de los usuarios (edad, nivel de estudios, experiencia técnica).
- **Prevención de errores:** La interfaz debe ser lo suficientemente clara para evitar que el usuario se equivoque, ya que una UI compleja genera rechazo.
- **Inclusividad y Eficiencia:** Debe permitir un acceso rápido al contenido sin sacrificar la comprensión, integrando diversas tecnologías (sonido, video, etc.) y hardware (teclado, ratón, pantallas táctiles).

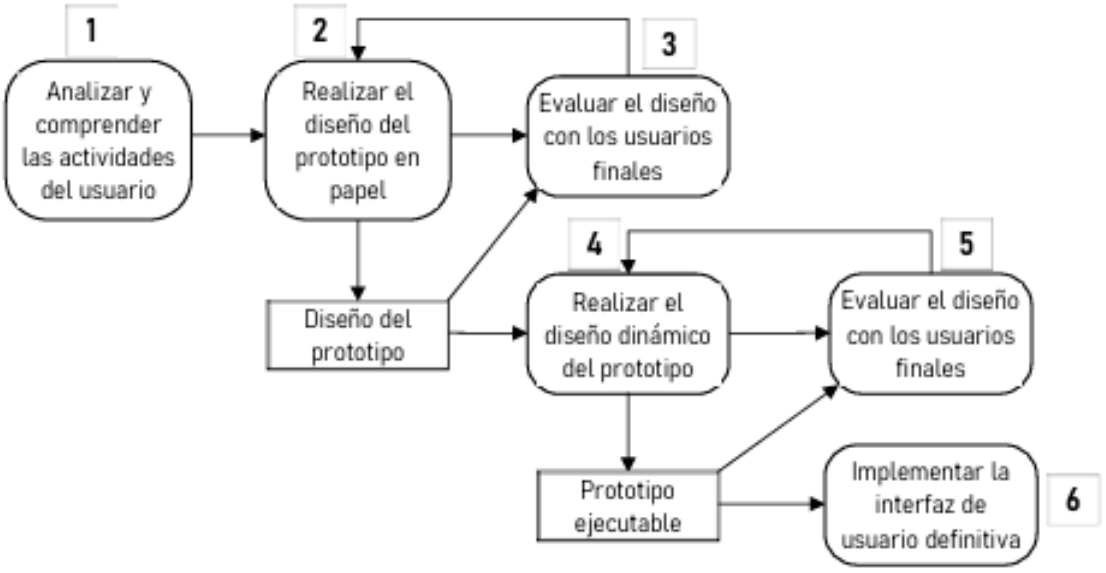
Las 3 Reglas de Theo Mandel para el DISEÑO:

a. Dar el control al usuario.
b. Reducir la carga de memoria del usuario (no obligarlo a recordar muchas cosas).
c. Lograr que la interfaz sea consistente (que siempre funcione igual).

Ubicación: Páginas 86, 87 y 95 del PDF de contenido.

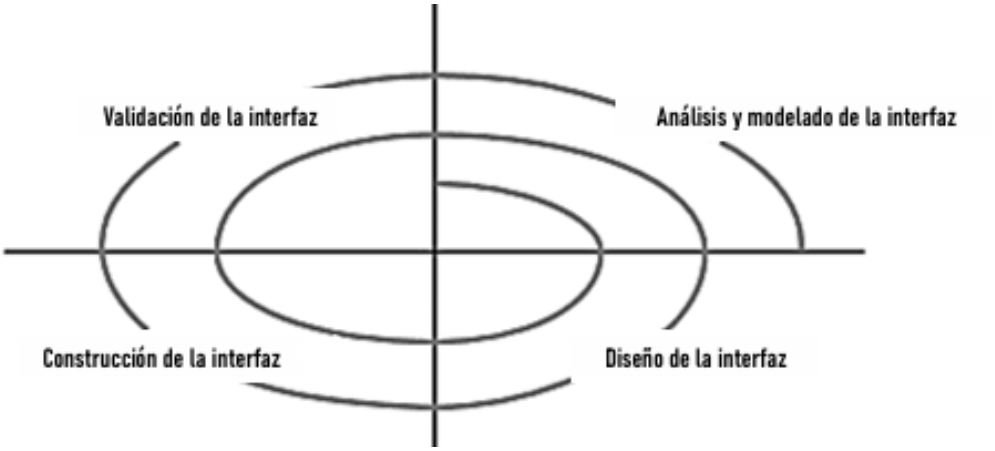
Pagina 87: Los 6 principios para el diseño UI

1. Enfoque en la experiencia del usuario: Comprender las necesidades y expectativas del usuario.
2. Prototipos en papel
3. Iteración con usuarios finales: Los diseños en papel se evalúan y refinan con los usuarios finales, permitiendo iteración antes de avanzar.
4. Diseño dinámico: Los prototipos se convierten en diseños dinámicos utilizando software o herramientas como powerPoint para simular interacciones.
5. Evaluación continua: Los diseños dinámicos se someten a evaluación y refinamiento con los usuarios, posibilitando ajustes antes de avanzar
6. Prototipo funcional: Una vez se logra el consenso, se desarrolla un prototipo funcional en el lenguaje final de implementación.



¿Cuál es el proceso del diseño UI?

1. Analisis y Modelado: se definnen los elementos y acciones de la interfaz
2. Diseño de la interfaz: se establecen las acciones que los usuarios realizaran para cambiar el estado de la interfaz (papel o programas de diseño)
3. Construcción de la interfaz: se basa en los diseños previos.
4. Validacion: se evalúa cómo el usuario interpreta el estado del sistema a partir de la información proporcionada por la interfaz.



Este ciclo se repite hasta que se logre una interfaz efectiva y eficiente que satisfaga las necesidades del usuario y cumpla con sus expectativas.

//Considerar poner preguntas sobre la EXPERIENCIA de usuario UX

5. ¿Qué es la gestión de configuración? Defina línea base y ejemplifique

- **Gestión de Configuración del Software (GCS):** Es un **CONJUNTO DE PRACTICAS Y PROCESOS** para identificar, controlar y rastrear los cambios en los elementos de un software (código, documentos, datos) a lo largo de su vida útil. **Esto es esencial para mantener la integridad y la calidad del software a medida que evoluciona a lo largo del tiempo.**
- UTILIDAD El GCS se usa para identificar y definir los elementos en el sistema, controlando el cambio de estos elementos a lo largo de su ciclo de vida, registrando y reportando el estado de los elementos y las solicitudes de cambio, y verificando que los elementos estén completos y que sean los correctos. En resumen, identifica el cambio, controla este, garantiza que el cambio se implemente adecuadamente e informa del cambio a todos aquellos que puedan estar afectados. Es una actividad de autoprotección que se aplica durante el proceso del software.
- **Línea Base (Baseline):** Es un punto de referencia en el desarrollo del software que ha sido revisado y acordado formalmente. Una vez establecida, cualquier cambio sobre ella debe seguir un proceso formal de control.
  Una línea base es un punto de referencia fundamental en el desarrollo del software. Esta línea base se establece cuando uno o más Elementos de Configuración de Software (ECS) han sido completados y aprobados. Una vez que estos elementos se convierten en una línea base, cualquier modificación o cambio posterior en el documento se somete a la gestión y control del GCS. En otras palabras, una línea base en GCS es una versión específica de un sistema o componente de software

que se toma como referencia para desarrollos y cambios futuros. La línea base representa un estado estable y conocido del software en un punto determinado en el tiempo. Cualquier modificación realizada después de establecer una línea base se registra y rastrea meticulosamente. Esto permite comparar los cambios con la versión de la línea base y mantener un historial claro de las alteraciones efectuadas. Esta práctica es esencial para asegurar que el software pueda evolucionar de manera controlada y que los cambios sean gestionados de manera efectiva y coherente a lo largo del tiempo.

- **Ejemplo:** Una "Especificación de Requisitos" aprobada por el cliente es una línea base. Si a mitad del proyecto el cliente quiere agregar un botón nuevo, no se cambia el documento así nomás; se debe pedir un cambio formal porque ya se había "cerrado" esa versión base para empezar a programar.

Ubicación: Páginas 137, 138 y 139 del PDF de contenido

Una rta mia; Una linea base es un punto donde el desarrollo de software ya fue aprobado, todo esta revisado y cumple con los requisitos **CORRECCION → te falta la característica más importante que diferencia a una Línea Base de una simple "versión final".**
Tu definición es **parcialmente correcta** porque efectivamente es un punto donde los elementos "han sido completados y aprobados", pero la clave técnica es lo que pasa **después** de ese punto.

Una Línea Base es:

1. Un **punto de referencia** estable.
2. Se crea cuando uno o más **Elementos de Configuración (ECS)** (código, documentos, datos) están **aprobados**.
3. **LO MÁS IMPORTANTE:** A partir de este momento, cualquier cambio **debe pasar por un control formal** (Gestión de Configuración de Software o GCS)

No es solo que "está aprobado", sino que **se "congela"** y ya no se puede modificar libremente; para cambiarlo hay que pedir permiso formal y registrarlo.

6. Rejuvenecimiento.

- **Concepto:** Es una parte del mantenimiento que busca mejorar la calidad global de un sistema que ya existe para evitar que se degrade o falle por su "envejecimiento". Sirve para alargar la vida útil del sistema.

Tipos de Rejuvenecimiento:

1. **Re-Documentación:** Analizar el código existeente para generar manuales o explicaciones de cómo funciona.
2. **Re-Estructuración:** Cambiar el código internamente para que sea más ordenado y fácil de manejar, sin cambiar lo que hace. A veces acompañado de una re-documentación.

3. **Ingeniería Inversa:** A partir del código fuente, intentar recrear los documentos de diseño que se perdieron o nunca se hicieron.
4. **Re-Ingeniería:** Es lo más completo; se usa ingeniería inversa para entender el sistema y luego se crea un código nuevo mejor estructurado.

Ubicación: Páginas 144 y 145 del PDF de contenido

7. **Calcule el camino crítico**

Tarea	Duración	Dependencia
A	30 min	-
B	10 min	A
C	120 min	B
D	15 min	A
E	80 min	A
F	20 min	A
G	45 min	C, I
H	25 min	G
I	60 min	D
J	25 min	B, I
K	50 min	D
L	30 min	D
M	5 min	J, K, L
N	100 min	E
O	10 min	F
P	10 min	C, H, M, N, O

8. Tipos de planificación organizacional vistos. // organigrama de equipos genericos

- Existen tres tipos principales de estructuras organizativas para los equipos de software:

1. **Descentralizado Democrático (DD):** No tiene un jefe permanente. Las decisiones se toman por consenso y la comunicación es horizontal entre todos. **Es bueno para problemas complejos y a largo plazo, EL PDF DICE adecuada para tareas raidas y problemas sencillos// CHEQUEAR ESTO, CUAL ES LA CORRECTA?**. Se nombran coordinadores de tareas a corto plazo y se sustituyen por otros cuando cambia la tarea. La resolución de problemas es compartida.
2. **Descentralizado Controlado (DC):** Tiene un líder definido que coordina tareas y líderes secundarios para subtareas. La resolución de problemas es grupal, pero la implementación se divide. La comunicación es tanto horizontal como vertical. Adecuada para problemas complejos
3. **Centralizado Controlado (CC):** Tiene un jefe que toma las decisiones y coordina todo. La comunicación es vertical (del jefe hacia abajo). Es eficaz para abordar problemas difíciles.

Los proyectos muy grandes se gestionan mejor mediante equipos con estructuras CC o DC ya que permiten formar subgrupos fácilmente.

La duración del tiempo de trabajo influye en la moral del equipo. Los equipos de tipo DD son más adecuados para equipos que permanecen juntos durante periodos largos.

Ubicación: Página 50.

DESCENTRALIZADO DEMOCRÁTICO (DD)	DESCENTRALIZADO CONTROLADO (DC)	CENTRALIZADO CONTROLADO (CC)
No tiene un jefe permanente, se nombran coordinadores de tareas a corto plazo y se sustituyen por otro cuando cambia la tarea.	Tiene un jefe definido que coordina tareas específicas y jefes secundarios para subtareas.	El jefe del equipo se encarga de la resolución de problemas a alto nivel y la coordinación interna del equipo.
La resolución de problemas es una actividad compartida.	La resolución de problemas es una actividad del grupo, pero la implementación de soluciones se reparte entre subgrupos por el jefe del equipo.	Las decisiones y enfoques adoptados son tomados por el jefe.
Las decisiones se toman por consenso.	Las decisiones y enfoques adoptados se toman por conceso.	La resolución de problemas es dirigida por el jefe.
La comunicación entre los miembros del equipo es horizontal.	La comunicación entre los subgrupos e individuos es horizontal y vertical.	La comunicación entre el jefe y los miembros del equipo es vertical.
Adecuada para tareas rápidas y problemas sencillos.	Adecuada para problemas complejos.	Eficaz para abordar problemas difíciles.

9. Característica de todos los riesgos.

- Todos los riesgos comparten dos características fundamentales:

1. **Incertidumbre (probabilidad):** El riesgo puede ocurrir o no; es decir, hay una probabilidad (nunca del 100%, porque si no sería un problema real y no un riesgo).
2. **Pérdida (impacto):** Si el riesgo se convierte en realidad, siempre traerá consecuencias negativas o pérdidas para el proyecto.

Ubicación: Página 51.

10. Definición de recuperabilidad (principios de diseño).

En el contexto del diseño (específicamente de interfaces y usabilidad), la recuperabilidad se refiere a la capacidad del sistema para permitir al usuario corregir sus errores.

- **Explicación fácil:** Si te equivocas al hacer clic o borrar algo, el sistema debe ofrecerte una forma de "deshacer" esa acción o recuperarte del fallo sin perder tu trabajo. Un diseño es recuperable si es tolerante a los errores del usuario.

Ubicación: Páginas 98 (mencionado como tolerancia a errores en usabilidad) y 112 (como tipo de defecto).

11. Diferencia entre modelo repositorio y modelo cliente-servidor.

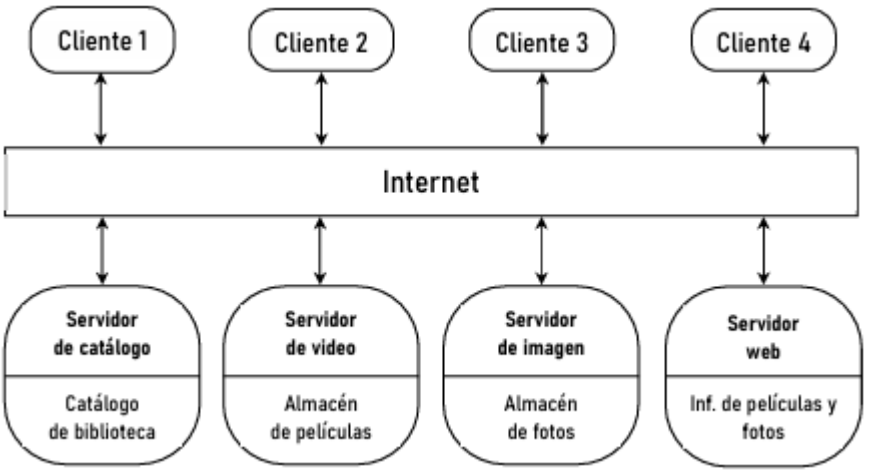
- Modelo Repositorio: Todos los subsistemas comparten una base de datos central única. Los datos se gestionan en un solo lugar y los módulos acceden a él. Es eficiente para compartir muchos datos, pero si el repositorio falla, todo el sistema cae.
  Ventajas:
  - Eficiencia: Compartir grandes cantidades de datos sin necesidad de transmitir información entre subsistemas.
  - Independencia funcional: Los subsistemas productores de datos no necesitan conocer el uso futuro de los datos.
  - Centralización: Actividades de backup, protección y control de acceso gestionadas de manera centralizada.
  - Representación visual: Modelo de datos compartido ofrece una representación estructurada y visual.
  - Integración sencilla: Facilita la incorporación de nuevas herramientas compatibles con el modelo de datos.

  Desventajas:
  - Limitación de flexibilidad: Subsistemas deben ajustarse a los modelos del repositorio, lo que puede afectar su adaptabilidad.
  - Complejidad de evolución: La acumulación de datos en el repositorio puede dificultar la evolución o migración del sistema, generando problemas de consistencia y errores.
  - Uniformidad de políticas: Dificultad para acomodar variaciones en protección y políticas de seguridad de diferentes subsistemas.
  - Dificultad en distribución: Problemas de redundancia, inconsistencias y mantenimiento al distribuir el repositorio en múltiples máquinas.

Aunque el patrón de repositorio puede ser beneficioso en términos de centralización y compartición de datos, también introduce desafíos en cuanto a flexibilidad, adaptabilidad y mantenibilidad a largo plazo.

- Modelo Cliente-Servidor: El sistema se divide en proveedores de servicios (servidores) y consumidores (clientes). Los datos y funciones están distribuidos. Es más flexible y escalable, pero la comunicación depende de la red.
- Ventajas:
    - Escalabilidad: El sistema puede escalarse de manera eficiente mediante la adición de nuevos servidores para manejar la carga de trabajo creciente.
    - Reutilización: Los servicios proporcionados por los servidores pueden ser reutilizados por múltiples clientes.
    - Interoperabilidad: Los clientes y servidores pueden estar en diferentes plataformas o lenguajes de programación. Las interfaces adaptan servicios.
- Desventajas:
    - Complejidad de Implementación: Configurar y mantener la infraestructura de servidores y la comunicación puede ser complejo.
    - Dependencia de la Red: La comunicación constante entre clientes y servidores significa que la calidad y disponibilidad de la red son críticas para el funcionamiento del sistema.
    - Puntos Únicos de Fallo: Si un servidor falla, los clientes que dependen de ese servidor pueden quedar afectados.

En resumen, el patrón Cliente-Servidor es poderoso para dividir responsabilidades y permitir escalabilidad y reutilización, pero también introduce desafíos relacionados con la complejidad, la dependencia de la red y la seguridad.



- Diferencia clave: En el repositorio todo gira en torno a los datos centrales; en cliente-servidor, todo gira en torno a los servicios distribuidos.

Ubicación: Páginas 68, 69 y 70.

12. Describa el problema de las 4 P.

- Se refiere a los 4 elementos fundamentales que deben gestionarse en un proyecto de software para que sea exitoso:

1. **Personal:** Es el factor más importante. Se debe gestionar a las personas, sus roles y habilidades.
2. **Producto:** Se debe definir claramente qué se va a construir (alcance y objetivos) antes de empezar.

3. **Proceso:** Se debe elegir un marco de trabajo (metodología) adecuado para que el equipo pueda cumplir con el plan.
4. **Proyecto:** Se debe planificar y controlar el trabajo para evitar el caos y asegurar la calidad.

Ubicación: página 16.

13. ¿Cuáles son los elementos claves de la gestión de proyectos?

- Para asegurar el éxito de un proyecto en tiempo, costo y calidad, se deben integrar los siguientes elementos:

1. Métricas.
2. Estimaciones.
3. Planificación temporal (calendario).
4. Planificación organizativa (equipo).
5. Análisis de riesgos.
6. Seguimiento y control.

Ubicación: Página 17.

MEDIDA	MEDICIÓN
Es una indicación cuantitativa de alguna propiedad o atributo, como tiempo, recursos, líneas de código, etc. Se utiliza para cuantificar la magnitud de un aspecto particular de un sistema, componente o proceso.	Es el proceso de determinar una medida a través de cálculos y análisis. Implica el acto de evaluar y registrar valores cuantitativos para obtener información sobre un atributo específico.
MÉTRICA	**INDICADOR**
Es un conjunto de reglas, fórmulas o criterios que se aplican a una medida para obtener información específica sobre el objeto que se está evaluando. Las métricas son herramientas que te obtener información más significativa a partir de las medidas.	Un indicador es un elemento de información que se forma a partir una combinación de métricas y establece una pauta de cuál es la medida correcta para un atributo. Esto permite al gestor del proyecto ajustar el producto, proceso o proyecto en función de la información brindada por el indicador

La medida es un objeto intangible, la medición es una acción. El resultado de una medición, la medida, da un valor cuantitativo concreto tomado en un punto particular. La métrica, en cambio, da un valor cuantitativo en base al análisis y recopilación de varias medidas de un atributo específico. De igual modo, el indicador también se obtiene en base a una recopilación de resultados de métrica.

14. Describa el modelo MOI.

- Es un modelo de liderazgo para mejorar la gestión del equipo de software. Sus siglas significan:

- **M (Motivación):** La habilidad del líder para incentivar al personal técnico a que dé su máximo rendimiento.

- **O (Organización):** La capacidad para estructurar procesos que permitan transformar el trabajo en un producto final.
- **I (Innovación):** La capacidad de fomentar nuevas ideas y creatividad en el equipo.

Ubicación: Página 48.

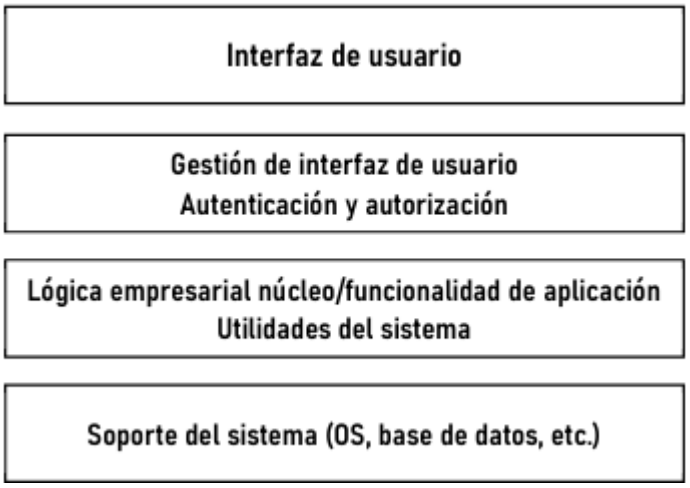
15. ¿Qué define el diseño arquitectónico? Describa los tipos de organización del sistema.

- **Definición:** El diseño arquitectónico define la relación entre los componentes estructurales principales del software. Es como el plano general de la casa antes de diseñar cada habitación. Establece la relación entre los componentes estructurales del software y selecciona los estilos arquitectónicos, patrones de diseño y otras decisiones clave para la estructura general del sistema.

Tipos de organización:

- **Repositorio:** Datos centrales compartidos.
- **Cliente-Servidor:** Servicios distribuidos.
- **Capas (Abstract Machine):** El sistema se organiza en capas jerárquicas, donde cada capa presta servicio a la superior. Normalmente se utiliza cuando se busca una separación clara de responsabilidades y una estructura modular, o en sistemas con múltiples niveles de abstracción, donde cada capa realiza una función específica y se comunica con las capas vecinas. Ejemplos de aplicación de este patrón incluyen sistemas de gestión de contenido, plataformas de aprendizaje en línea, aplicaciones empresariales y herramientas de procesamiento de imágenes, entre otros.
    - Ventajas:
        - Desarrollo incremental: Permite añadir capas gradualmente, facilitando el desarrollo iterativo.
        - Portabilidad y resistencia al cambio: Mantener las interfaces entre capas permite cambiar o reemplazar capas sin afectar al sistema completo. Incluso en caso de no poder modificar una interface, se puede generar una capa de adaptación para permitir el acoplamiento.
        - Adaptabilidad multiplataforma: Las capas internas son las únicas dependientes de la plataforma, permitiendo crear capas de interfaz específicas para adaptar otras plataformas.
        - Separación de responsabilidades: Cada capa tiene una función clara e independiente, facilitando la comprensión y el mantenimiento del sistema.
    - Desventajas
        - Estructuración compleja: Planificar qué hace cada capa, dónde ubicarla y cómo será su interfaz puede resultar complejo en la práctica.
        - Impacto en rendimiento y mantenibilidad: Las capas internas a menudo brindan servicios necesarios para todas las capas. Sus cambios o errores pueden desencadenar fallas a otras unidades.
        - Rutas de acceso complejas: Los servicios del usuario pueden pasar por múltiples capas adyacentes, pudiendo ocurrir problemas con el flujo y procesamiento de la información.

- Interpretación repetida de servicios: En sistemas con muchas capas, un servicio solicitado por la capa superior puede tener múltiples interpretaciones en capas diferentes, lo que reduce la eficiencia y fluidez de respuesta.



En resumen, el patrón de arquitectura en capas brinda una estructura modular y una separación clara de responsabilidades, pero también introduce desafíos en términos de diseño, rendimiento y complejidad de las interacciones entre capas.

Ubicación: Páginas 58, 67, 68, 70.

16. Enumere y describa los principios de diseño. // consultar creo que esta incompleta

Para lograr un buen diseño se deben seguir principios como:

- Modularidad: Dividir el sistema en partes pequeñas (módulos) independientes.
- Cohesión: Que cada módulo haga una sola cosa y la haga bien.
- Acoplamiento: Que los módulos dependan lo menos posible unos de otros.
- Abstracción: Ocultar los detalles complejos y mostrar solo lo necesario.
- Refinamiento: Ir de lo general a lo detallado paso a paso.

Ubicación: Páginas 62, 63, 64.

17. ¿Qué es un proyecto?

Es un esfuerzo planificado y controlado para crear un producto de software. Implica comprender los factores de éxito, gestionar riesgos, definir objetivos claros, asignar recursos y monitorear el progreso para cumplir con los plazos y la calidad.

Ubicacion pagina 16.
Proyecto → Un proyecto es un esfuerzo temporal y progresivo con un inicio y fin definidos, que se lleva a cabo para crear un producto, servicio o resultado único. Además, puede considerarse como la subdivisión de un problema inicial en tareas más pequeñas y sencillas, lo que permite manejar la complejidad del software que se desea desarrollar.

- Temporal → Un proyecto tiene un punto de inicio claro, marcado por la fase inicial de planificación y organización, y un fin definido, que se alcanza cuando los objetivos del proyecto han sido logrados satisfactoriamente o cuando se reconoce que no se pueden alcanzar.
- Resultado → Estos resultados pueden ser productos, servicios, resultados únicos o logros específicos que se crean para satisfacer necesidades particulares.
- Elaboración gradual →La metodología de desarrollo gradual es común en los proyectos. Esto implica dividir el proyecto en fases o etapas, desarrollando y mejorando incrementos sucesivos del producto a lo largo del tiempo.

18. En planificación temporal, ¿qué tareas se deben realizar para hacer el seguimiento y control de proyecto?

- Para controlar que el proyecto no se desvíe del plan (tiempo y costo), se deben realizar tareas como:
    - Monitorear el progreso regularmente, comparando lo real vs. lo planificado.
    - Gestionar el alcance para evitar cambios descontrolados.
    - Analizar riesgos y tomar acciones correctivas si hay retrasos.
    - Utilizar métricas de estado y uso del tiempo.

Ubicación: Páginas 16, 17, 19.

19. ¿Cuáles son los tipos (áreas) de diseño de software? Describa.

El diseño de software se encarga de definir cómo se organizarán los componentes, módulos y funciones del software, así como las interacciones entre ellos, para lograr los objetivos planteados. El diseño de software es esencial, ya que representa la primera actividad técnica en el proceso de desarrollo. Se considera el núcleo técnico de la ingeniería de software, donde los problemas se convierten en soluciones.

El diseño se divide en cuatro áreas principales:

1. **Diseño de datos** → Transforma los objetos de datos en estructuras de datos (como bases de datos).
2. **Diseño arquitectónico** → Define la estructura general y la relación entre los componentes principales.
3. **Diseño de interfaz** → Describe cómo se comunica el software con el usuario y con otros sistemas.
4. **Diseño de componentes (o procedimental)** → Transforma la arquitectura en una descripción detallada de cómo funciona cada módulo por dentro.

Ubicación: Páginas 57, 58, 59.

20. ¿Qué es verificación? ¿Qué es validación? De todos los tipos de pruebas vistos, ¿cuál dirías que se está verificando y cuál que se está validando?

- **Verificación:** ¿Estamos construyendo el producto correctamente? Se enfoca en si el software cumple con las especificaciones técnicas y de diseño. (Ej: Pruebas unitarias, de integración, de caja blanca).

- **Validación:** ¿Estamos construyendo el producto correcto? Se enfoca en si el software cumple con lo que el cliente realmente necesita y quiere. (Ej: Pruebas de aceptación, de caja negra, validación con el cliente).

Ubicación: Página 120.

21. ¿Qué es el rejuvenecimiento del software? Describa los tipos.

Es mejorar la calidad de un sistema existente para alargar su vida útil. Tipos:

- **Re-Documentación:** Crear documentación nueva del código para entenderlo mejor.
- **Re-Estructuración:** Mejorar la estructura interna del código sin cambiar su función.
- **Ingeniería Inversa:** Analizar el código para recuperar el diseño original.
- **Re-Ingeniería:** Rediseñar y reescribir partes del sistema para modernizarlo.

Ubicación: Páginas 144, 145.

22. La empresa de pastas HOGAREÑAS S.A desea hacer un estudio del tiempo de demora en hacer una lasaña, para esto ha identificado las siguientes actividades en minutos. Encontrar el camino crítico, calculando tiempos tempranos y tiempos tardíos. Calcular el tiempo final.

ACTIVIDADES		PREDECESOR	DURACIÓN
A	Comprar queso		30
B	Rayar queso	A	5
C	Batir huevos		2
D	Mezclar huevos y queso	C, B	3
E	Picar cebollas y hongos		7
F	Cocinar la salsa de tomate	E	25
G	Hervir agua		15
H	Hervir la pasta de lasaña	G, F	10
I	Enjuagar la pasta de lasaña	H	2
J	Unir los ingredientes	I, F, D, B	10
K	Precalentar horno		15
L	Hornear la lasaña	J, K	30

23. ¿Cómo se clasifican los riesgos?

Un riesgo se define como un evento no deseado que conlleva consecuencias negativas. Componentes:

- Incertidumbre (probabilidad): Representa la probabilidad de que ocurra un evento riesgoso. Nunca llega al 100%, ya que ese sería un problema presente.

- Pérdida (impacto): Indica el impacto y grado de consecuencias negativas que el evento riesgoso podría causar. Puede variar en intensidad, desde leve hasta grave.

Se pueden clasificar en tres categorías generales:

1. **Riesgos del proyecto** Amenazan el plan del proyecto (presupuesto, calendario, recursos).
2. **Riesgos técnicos** Amenazan la calidad y la implementación técnica (tecnología nueva, dificultad de diseño).
3. **Riesgos del negocio** Amenazan la viabilidad del producto en el mercado (nadie lo quiere, pierde dinero).

Ubicación: Página 51 (implícito en la descripción general de riesgos).

24. ¿Cómo puede ser la organización del sistema? (Diseño arquitectónico)

.Esta pregunta es similar a la 15.

- El diseño arquitectónico define la relación entre los componentes estructurales principales del software. Es como el plano general de la casa antes de diseñar cada habitación. Establece la relación entre los componentes estructurales del software y selecciona los estilos arquitectónicos, patrones de diseño y otras decisiones clave para la estructura general del sistema.

La organización puede ser:

- Centralizada (Repositorio).
- Distribuida (Cliente-Servidor).
- Jerárquica (Capas).

Ubicación: Página 67.

25. Explique el PCMM (modelo de capacitación y motivación del personal).

Nota importante: En el PDF proporcionado *no* aparece explícitamente el término "PCMM" (People Capability Maturity Model). Sin embargo, el documento trata este tema bajo el "Modelo MOI de Liderazgo" (página 48), que se enfoca en Motivación, Organización e Innovación para gestionar el capital humano, que es el activo más valioso.

El líder de equipo es una persona que desempeña un papel fundamental en la gestión y dirección de un grupo de individuos que trabajan juntos hacia un objetivo común. Tiene la responsabilidad de guiar, coordinar, motivar y supervisar a los miembros del equipo para asegurarse de que trabajen de manera efectiva y alcancen los resultados deseados. La organización del equipo de software es crucial para maximizar las habilidades y capacidades de cada miembro.
Modelo MOI de Liderazgo:

- Motivación al personal: Habilidad para incentivar al personal técnico a dar su máximo rendimiento.

- Organización del equipo: Habilidad para diseñar procesos que conduzcan al producto final.
- Innovación: Capacidad para fomentar la creatividad y la generación de ideas del personal.

26. Enumere y describa las técnicas de estimación que conozca.

Sirven para predecir tiempo, costos y recursos:

- **Juicio Experto:** Consultar a personas con experiencia para que den su opinión.
- **Técnica Delphi:** Un grupo de expertos estima de forma anónima y luego discuten las diferencias hasta llegar a un consenso.
- **Divicion de Trabajo Jerarquica:** Esta técnica implica la consulta a personas en diferentes niveles jerárquicos de la organización. Comienza consultando a quienes están en niveles más bajos y avanza hacia arriba en la jerarquía. Cada nivel proporciona su estimación basada en su comprensión del proyecto y de cómo se verá afectado por sus roles.
- **Planning Poker:** Técnica ágil donde el equipo usa cartas con números (serie de Fibonacci) para estimar el esfuerzo de las tareas en conjunto.
- **Puntos de función / LDC:** Estimaciones basadas en fórmulas matemáticas usando el tamaño o funcionalidad del software.

Ubicación: Páginas 33, 34.

27. Defina y diferencia pruebas de caja blanca y de caja negra. De un ejemplo de cada una.

- Caja Blanca (Cristal): Se conoce el código interno. Se prueban los caminos lógicos, bucles y condiciones internas del programa.
    - *Ejemplo:* Probar que un `if` en el código funcione tanto para el "verdadero" como para el "falso".

- Caja Negra (Opaca): No se mira el código. Se prueba la funcionalidad basándose en qué entra y qué debe salir según los requisitos.
    - *Ejemplo:* Ingresar un usuario y contraseña válidos en un login y verificar que el sistema permita el acceso, sin importar cómo lo haga por dentro.

Ubicación: Páginas 113, 118.

28. ¿Qué consideraciones se deberían tener en cuenta a la hora de desarrollar una interfaz de usuario?

Se debe priorizar la **experiencia del usuario (UX)**:

- Dar el control al usuario (que pueda deshacer acciones).
- Reducir la carga de memoria (que no tenga que recordar cosas de una pantalla a otra).
- Hacer la interfaz consistente (que los botones y menús sean siempre iguales).

- Adaptarse a la diversidad de usuarios (principiantes y expertos).

Para desarrollar una interfaz de usuario (UI) efectiva, se deben considerar diversos aspectos técnicos, humanos y de diseño que aseguren una comunicación fluida entre la persona y la máquina.

1. Las Tres Reglas Doradas de Theo Mandel

Estas reglas son fundamentales para enfocar el diseño en las necesidades del usuario:

- **Dar el control al usuario**: El sistema debe adaptarse al usuario y permitirle tomar decisiones, como deshacer acciones o interrumpir procesos.
- **Reducir la carga de memoria**: La interfaz debe ser intuitiva para que el usuario no tenga que recordar información compleja. Se recomienda usar valores por defecto y metáforas visuales de la realidad.
- **Lograr consistencia**: Toda la aplicación (o familia de aplicaciones) debe mantener el mismo estilo visual y reglas de diseño para evitar confusiones.

2. Principios de Diseño y Usabilidad

- **Enfoque en la experiencia (UX)**: Es esencial comprender qué necesita y qué espera el usuario antes de empezar a diseñar.
- **Simplicidad e Inclusión**: La interfaz debe ser fácil de entender para distintos niveles de habilidad (desde novatos hasta expertos) y permitir un acceso rápido al contenido.
- **Adaptabilidad**: El diseño debe funcionar en diferentes dispositivos (PCs, celulares, tablets) y configuraciones de hardware.

3. Aspectos Visuales y de Soporte

- **Uso del color**: No se debe abusar de los colores (máximo 4 o 5 por ventana) y siempre se debe considerar la accesibilidad para personas con dificultades visuales, como el daltonismo.
- **Prevención de errores**: Es mejor guiar al usuario para que no cometa errores (por ejemplo, usando menús de selección en lugar de campos de texto libre) que simplemente mostrar un mensaje de error después.
- **Ayuda y retroalimentación**: El sistema debe informar siempre su estado actual y ofrecer ayudas contextuales si el usuario se pierde.

Proceso Sugerido para el Diseño

El documento propone un ciclo iterativo para asegurar la calidad de la interfaz:

1. **Analizar** las actividades del usuario.
2. Crear **prototipos en papel** para evaluar ideas rápidamente.
3. **Iterar** con usuarios finales para recibir comentarios.
4. Desarrollar un **diseño dinámico** (simulaciones en software).
5. Crear el **prototipo funcional** definitivo.

Ubicación: Página 86, 87, 95, 96 y 97

29. Defina métricas del proceso y del producto/objetivas y subjetivas. ¿Qué son las métricas post mortem y cuáles las tempranas?

- **Métricas del Proceso:** Miden cómo trabajamos (eficiencia, calidad del flujo de trabajo) para mejorar a largo plazo.
- **Métricas del Producto:** Miden las características del software en sí (calidad, fiabilidad, complejidad) en un momento dado.
- **Métricas Post Mortem:** Se analizan *después* de terminar el proyecto para aprender de los errores y éxitos (lecciones aprendidas).
- **Métricas Tempranas:** Se usan al *inicio* (durante el diseño o análisis) para predecir problemas antes de programar.

Ubicación: Páginas 19, 20, 22.

30.

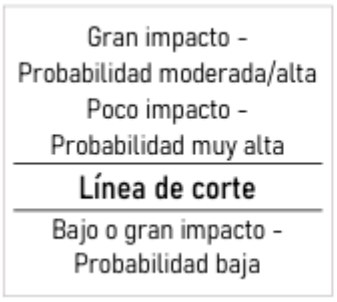
. **Desarrolle un PERT de las siguientes actividades**

Descripción	Tarea	Precedencia	Duración esperada en días
Realizar entrevistas	A	Ninguna	4
Aplicar cuestionarios	B	A	4
Leer informes de la organización	C	Ninguna	8
Analizar el flujo de datos	D	B, C	3
Introducir prototipos	E	B, C	8
Observar las reacciones hacia el prototipo	F	E	5
Realizar el análisis de costo/ beneficio	G	D	8
preparar la propuesta	H	F, G	2
Presentar la propuesta	I	H	2

31. Enumere las actividades del análisis de riesgo.

1. **Identificación:** Hacer una lista de todos los posibles riesgos (brainstorming).

2. **Análisis/Estimación/ Linea de Corte Y Boehm:** Determinar la probabilidad de que ocurran y el impacto (daño) que causarían. (La lista de riesgos se ordena considerando su impacto y probabilidad, y se coloca una línea de corte para establecer cuáles riesgos serán atendidos. Mientras Boehm sugiere supervisar los 10 riesgos más altos, ese número debe ser manejable y adaptarse al proyecto. Los riesgos por encima de la línea reciben atención primaria, mientras que los debajo se reevaluarán en supervisiones y tendrán una prioridad secundaria.)



Riesgo	Categoría	Probabilidad	Impacto
AAA	Proyecto	80%	2
FDS	Producto	80%	3
DFA	Negocio	75%	3
GFE	Proyecto	60%	2
GTR	Proyecto	50%	2
UID	Negocio	40%	1
HWR	Producto	20%	2

3.
4. **Planificación (gestión):** Decidir qué hacer si ocurren (minimizar, evitar, aceptar).
5. **Supervisión (Control):** Vigilar los riesgos durante todo el proyecto.

Ubicación: Páginas 53, 54.

32. Describa los fundamentos del diseño. ¿Qué es la cohesión funcional? Describa los grados de cohesión.

- Fundamentos: Modularidad, abstracción, refinamiento.
- Cohesión: Medida de cuán relacionadas están las tareas *dentro* de un solo módulo. Se busca que sea alta.
- Cohesión Funcional (La mejor): El módulo hace una sola cosa específica y nada más.
- Grados de Cohesión (de peor a mejor):
    - Coincidental (tareas sin relación).
    - Lógica (tareas similares, ej. "todas las entradas").
    - Temporal (se ejecutan al mismo tiempo).
    - Procedimental (parte de una secuencia).
    - Comunicacional (usan los mismos datos).
    - Secuencial (la salida de uno es entrada de otro).
    - Funcional (una sola tarea definida).

Ubicación: Página 63.

33. Enumere las técnicas de validación que conozca.

La validación asegura que el software cumple con los requisitos del cliente.

- **Pruebas de aceptación:** El usuario prueba el sistema final. Dentro de estas pruebas estan las:
    - **Pruebas alfa:** El usuario prueba en el entorno del desarrollador.

- **Pruebas beta:** El usuario prueba en su propio entorno real sin el desarrollador presente. En ocasiones, la versión beta se libera en el mercado para ser probada.

ALFA	BETA
Realizadas por el cliente.	Realizadas por los usuarios finales del software.
Ejecutadas en el lugar de desarrollo.	Llevadas a cabo en los lugares de trabajo de los clientes o en ambientes de producción. En ocasiones la versión beta se libera en el mercado para ser probada.
Entorno controlado: Se usa el software de forma natural, similar a como lo utilizaría un usuario en situaciones normales o reales, pero de manera controlada.	Entorno no controlado: la prueba beta es una aplicación en vivo del software, es decir, que este es presentado a los usuarios finales en un entorno real o de producción sin poder controlar su uso.
El desarrollador participa como observador.	El desarrollador normalmente no está presente.
El desarrollador es quien registrando errores y problemas de uso.	Los clientes registran los problemas encontrados durante la prueba beta y los informan regularmente al desarrollador.
Es posible añadir funciones y realizar ajustes en función tras las observaciones.	Se realizan ajustes según los comentarios de usuarios y clientes, y una vez no se necesiten más cambios en el software, se lanza la versión final.
No es la versión final del software ni la última etapa de prueba	Las pruebas beta constituyen la última fase de las pruebas y emplean técnicas de caja negra (pruebas de la interfaz).

- **Revisión con el cliente:** Mostrar prototipos o versiones preliminares.

Ubicación: Página 131 (Pruebas Alfa/Beta), 120 (validacion).

34. ¿Qué tipos de mantenimiento conoce? Defina ingeniería inversa y reingeniería.

La validación asegura que el software cumple con los requisitos del cliente.
El mantenimiento es la etapa final del ciclo de vida (después de la entrega) y se clasifica en cuatro tipos según su propósito y frecuencia de uso:
1. Tipos de Mantenimiento:
- Mantenimiento Perfectivo (60%): Es el más común. Se centra en mejorar el sistema, optimizar su eficiencia y agregar nuevas funcionalidades que el cliente necesite.
- Mantenimiento Adaptativo (18%): Su objetivo es ajustar el software para que siga funcionando cuando cambia su entorno, como nuevas plataformas, sistemas operativos o hardware, o para integrarlo con otros sistemas.
- Mantenimiento Correctivo (17%): Se dedica a diagnosticar y solucionar errores o "bugs" encontrados en el software, ya sean problemas de lógica, rendimiento o documentación.

- Mantenimiento Preventivo (5%): Es el menos frecuente. Consiste en realizar modificaciones anticipadas (antes de que ocurra un problema) para facilitar el mantenimiento futuro y evitar fallos.

2. Definiciones: Ingeniería Inversa y Reingeniería
Estos conceptos pertenecen a las técnicas de "Rejuvenecimiento del Software" :

- Ingeniería Inversa: Se utiliza cuando se tiene un sistema antiguo ("legado") del cual no existe documentación para entenderlo. El proceso parte del código fuente hacia atrás para generar documentos de diseño, especificaciones o manuales que expliquen cómo funciona el sistema. En resumen: Es analizar el código para entender qué hace y documentarlo.
- Reingeniería: Es una extensión o paso siguiente de la ingeniería inversa. Primero se aplica ingeniería inversa para entender el sistema y obtener su diseño; luego, con esa información, se genera un nuevo código fuente mejor estructurado y de mayor calidad, pero sin alterar la funcionalidad original del programa. En resumen: Es entender el sistema viejo (ingeniería inversa) para reconstruirlo mejor (nuevo código) haciendo exactamente lo mismo.

Ubicación: Página 142 y 145

35. Describa el concepto de cohesión y sus grados.

- **Fundamentos:** Modularidad, abstracción, refinamiento
- **Cohesión:** Medida de cuán relacionadas están las tareas *dentro* de un solo módulo. Se busca que sea **alta.**
- **Cohesión funcional (La mejor):** El módulo hace una sola cosa específica y nada más.
- **Grados de cohesión (de peor a mejor):**
    - Coincidental (tareas sin relación).
    - Lógica (tareas similares, ej. "todas las entradas").
    - Temporal (se ejecutan al mismo tiempo).
    - Procedimental (parte de una secuencia).
    - Comunicacional (usan los mismos datos).
    - Secuencial (la salida de uno es entrada de otro).
    - Funcional (una sola tarea definida).

Ubicación: Página 63.

36. Defina las estrategias de integración.

Son las formas de unir los módulos para probarlos juntos:

- **Big Bang:** Unir todo de golpe (no recomendado, difícil encontrar errores).
- **Descendente (Top-Down):** Empezar por el módulo principal e ir bajando. Se usan "stubs" (módulos falsos) para simular los de abajo.
- **Ascendente (Bottom-Up):** Empezar por los módulos de más abajo (los más detallados) e ir subiendo. Se usan "drivers" (controladores) para probarlos.

Ubicación: Páginas 125, 126.

37. ¿Qué es la barrera del mantenimiento?

Es el punto crítico económico y técnico donde ya no conviene seguir manteniendo un sistema viejo porque es demasiado costoso, riesgoso o difícil. En este punto, es más rentable tirar el sistema viejo y construir uno nuevo desde cero que seguir arreglándolo.

- Ubicación: Página 141.

Finales Ingeniería de Software II

Una empresa de fletes de La Plata ha hecho convenios con otras empresas más pequeñas en el resto del país para optimizar los viajes y lograr mejores beneficios. Para eso necesita gestionar la información de los viajes de manera óptima. En una primera etapa piensan trabajar con 20 empresas con proyección a ir creciendo cada semestre. Los requerimientos se presentan muy variantes (cambiantes) tanto funcionales como no funcionales. Se solicita que la interfaz sea intuitiva y pensada para evitar errores y poder informar claramente cómo actuar en cada caso.

[(1) Mencione y describa dos principios de Nielsen que apunten al feedback y la prevención de errores, que son dos de los problemas detectados en la interfaz].

Rta: Página 94
Teniendo en cuenta que la interfaz debe ser intuitiva y evitar errores, los dos principios (heurísticas) clave son:

1. Visibilidad del estado del sistema (Feedback): El sistema debe mantener siempre informado al usuario de lo que está ocurriendo mediante retroalimentación adecuada en un tiempo razonable.
    - Ejemplo en el caso: Si el operador asigna un viaje, el sistema debe mostrar un mensaje verde que diga "Viaje asignado con éxito" o un ícono de "cargando" mientras busca la ubicación, para que no crea que se trabó.
2. Prevención de errores: Es mejor diseñar un sistema que prevenga el error antes que uno que solo dé buenos mensajes de error.
    - Ejemplo en el caso: En el formulario de "solicitar flete", en lugar de dejar que el usuario escriba la fecha manualmente (y pueda poner una fecha pasada por error), poner un calendario desplegable que tenga bloqueados los días anteriores a hoy.

Tiene que soportar una fuerte demanda de acceso y con un rendimiento óptimo las 24 horas del día. Para ello deciden hacer un sitio donde presentar sus servicios y que la gente pueda solicitarlos sin importar donde se encuentren. El software por desarrollar parece ser de

tamaño medio, pero se torna complejo por la organización de rutas de reparto y geolocalización de cada entrega que solicitaron para ser vistas en el sistema, esto va a hacer que el equipo esté mucho tiempo trabajando junto abordando nuevas tareas y capacitaciones para desarrollar el producto. Deciden solicitar el desarrollo a SoftIT una empresa con amplia experiencia en Ingeniería de Software. En la empresa SoftIT aceptaron el trabajo, y, dada las características del desarrollo decidieron Identificar y definir los elementos en el sistema, para controlar sus cambios a lo largo de todo el desarrollo.
[(2) a) ¿A qué estaría haciendo referencia? b) ¿Cuáles serían los elementos que controlar? Dé al menos 2 ejemplos.] *Leandro, líder del proyecto piensa en la mejor constitución del equipo que lo desarrollará.*

Rta: Pagina 138 y 139
a) Hace referencia a la Gestión de Configuración del Software (GCS). Es el conjunto de actividades diseñadas para controlar los cambios, identificando los productos de trabajo que es probable que cambien y auditando que las modificaciones se hagan correctamente.

b) Elementos a controlar: Se llaman Items de Configuración de Software (ICS) o Configuration Items. Son la unidad básica de lo que se va a controlar.

Ejemplo 1: El Documento de Especificación de Requisitos (SRS), ya que el texto dice que los requerimientos son "muy variantes", este documento cambiará mucho.

Ejemplo 2: El Código Fuente de los módulos de geolocalización.

[(3) Enumere al menos 3 factores a considerar cuando se planifica una estructura de equipo]. *Bruno (socio de Leandro), identifica los riesgos a cubrir. Le preocupa la decisión y luego de ordenarlos, trazó una línea de corte.*

Rta: Página: 50.
Respuesta: Debe considerar la naturaleza del proyecto para armar el equipo. Tres factores clave son:

1. La dificultad del problema: El texto menciona que hay "geolocalización" y "rutas de reparto", lo cual añade complejidad técnica.
2. El tamaño del programa: Se menciona que el software es de "tamaño medio", lo que influye en cuánta gente se necesita.
3. La estabilidad de los requisitos: El caso dice explícitamente que los requisitos son "muy variantes/cambiantes", lo que suele requerir equipos más flexibles o ágiles. (Otros factores válidos: tiempo que el equipo permanecerá unido, grado de modularización).

[(4) a) Defina qué es la línea de corte.
b) Teniendo en cuenta la decisión del inciso anterior, enumere 3 riesgos de este proyecto, cada uno con su estrategia]. *Leandro sabe de la importancia de tener todo debidamente probado con lo cual sugiere diferentes técnicas de caja blanca y caja negra, y va realizando las pruebas de integración.*

Rta: Página: 55.

a. **Línea de corte:** Es una técnica (usada en la tabla de Boehm) donde se ordenan los riesgos según su probabilidad e impacto. Se traza una línea horizontal; los riesgos que quedan por encima de la línea son los que se deben gestionar activamente (con planes de contingencia), y los que quedan por debajo solo se supervisan por si suben de categoría.
b. 3 Riesgos del proyecto y sus estrategias:

1. Riesgo: Alta inestabilidad de requisitos (el cliente cambia de idea a cada rato).
    a. Estrategia: Mitigación. Usar un modelo incremental y hacer prototipos rápidos de la interfaz para congelar requisitos por etapas.

2. Riesgo: Falta de experiencia técnica en geolocalización (se menciona que necesitan "capacitaciones").
    a. Estrategia: Mitigación. Contratar una capacitación externa inmediata o asignar tiempo extra en el cronograma para la curva de aprendizaje (Riesgo de personal/técnico).

3. Riesgo: Problemas de rendimiento con la "fuerte demanda 24hs".
    a. Estrategia: Supervisión. Realizar pruebas de carga y estrés tempranas para vigilar si el servidor aguanta la concurrencia proyectada.

[(6) Indique qué integración utilizaría, por qué y describa brevemente la misma]. La empresa de fletes decide comprar un nuevo vehículo para sumar a la flota:

Rta: Página: 126 y 127.

Recomendaría la Integración Sándwich (o Híbrida).

Porque el caso menciona que la interfaz es crítica ("intuitiva", "feedback") y los requisitos cambian mucho: esto pide una integración Descendente (Top-Down) para probar la interfaz y el flujo principal cuanto antes.

Pero también hay módulos complejos de bajo nivel como la "geolocalización" y "rutas": esto pide una integración Ascendente (Bottom-Up) para asegurar que los cálculos complejos funcionen bien antes de unirlos.

Descripción breve: La integración Sándwich combina ambas. Se prueban los niveles superiores (interfaz) con "stubs" (módulos falsos) y simultáneamente los niveles inferiores (geolocalización) con "drivers" (controladores), encontrándose en el medio del sistema.

	Tarea	Precedida por	Duración
A	Hacer un estudio de factibilidad	-	3
B	Lista de los modelos posibles	A	1
C	Investigar todos los modelos posibles	B	3
D	Entrevista con el mecanismo	C	1
E	Obtener precios en las agencias	C	2
F	Compilar los datos adecuados	B,E	1
G	Escoger los tres modelos mejores	F	2
H	Conducción de prueba con los 3 modelos	G	3
I	Pedir datos de garantía y de financiamiento	G	3
J	Escoger un vehículo	H,I	1
K	Comprar en automóvil	F,G,J	2

(7)Calcular las fechas más tempranas, más tarifas, realizar el grafico y encontrar el camino critico] Video donde lo explica: 8_Clase 5 Planificación Temporal - 2022

1. Describe 4 criterios de diseño de software.

(Página 59) Para crear un buen diseño de software, se deben seguir ciertos criterios técnicos fundamentales:

1. **Modularidad:** El sistema debe dividirse en partes más pequeñas (subsistemas o componentes) que sean independientes y tengan una función clara. Esto hace que sea más fácil de entender y arreglar.
2. **Interfaces simplificadas:** Las conexiones entre los módulos deben ser lo más simples posible. Si un módulo necesita hablar con otro, debe hacerlo de forma clara y sin enredos complejos.
3. **Independencia funcional:** Cada componente debe encargarse de una sola cosa específica. Si un módulo hace de todo, es un mal diseño.
4. **Representaciones múltiples:** El diseño debe poder verse desde distintos ángulos (como un plano de datos, un plano de estructura, un plano de interfaz) para entenderlo por completo.

2. Menciona los paradigmas de equipo y describe 1.

(Página 50) El texto presenta tres estructuras u organigramas de equipo genéricos (paradigmas de organización):

1. Democrático Descentralizado (DD)
2. Descentralizado Controlado (DC)
3. Centralizado Controlado (CC)

- Descripción del Centralizado Controlado (CC):

    - Cómo funciona: Hay un jefe absoluto que toma todas las decisiones y coordina todo el trabajo. La comunicación es vertical (del jefe hacia abajo).
    - Cuándo es útil: Es ideal para resolver problemas sencillos o tareas que requieren mucha rapidez, ya que no se pierde tiempo discutiendo decisiones.

3. Menciona 4 principios de Nielsen y describe 1.

(Páginas 99 y 100) Los principios de usabilidad de Jacob Nielsen para mejorar la interfaz de usuario son:

1. Diálogo simple y natural.
2. Lenguaje de usuario (hablar como la persona, no como la máquina).
3. Minimizar el uso de la memoria del usuario.
4. Consistencia.
5. Feedback (Retroalimentación).

- Descripción de "Feedback":
    - Qué es: El sistema siempre debe informar al usuario sobre lo que está pasando.
    - Ejemplo: Si el usuario hace clic en "Guardar", el sistema debe mostrar un mensaje de "Guardando..." o un ícono de carga. Si no muestra nada, el usuario pensará que se trabó y volverá a hacer clic, generando errores.

4. 2 ventajas y 2 desventajas del patrón repositorio.

(Páginas 68 y 69) Este patrón organiza el sistema alrededor de una base de datos central donde todos los subsistemas guardan y leen información.

- **Ventajas:**
    1. **Eficiencia en datos grandes:** Es excelente para compartir grandes volúmenes de información sin tener que estar pasándolos de un subsistema a otro.
    2. **Centralización del control:** Es más fácil hacer copias de seguridad (backup) y gestionar la seguridad porque todo está en un solo lugar.
- **Desventajas:**
    1. **Punto único de fallo:** Si el repositorio central se rompe, todo el sistema deja de funcionar.
    2. **Dificultad de evolución:** Si querés cambiar la estructura de la base de datos, tenés que actualizar todos los subsistemas que la usan, lo cual es muy costoso y difícil.

5. Cuándo usar juicio experto.

(Página 33) El juicio experto es una técnica de estimación que se basa en consultar a personas con experiencia. Se debe usar cuando:

- **No hay datos históricos:** Es un proyecto nuevo y no tenemos registros anteriores para comparar.

- **Proyectos de alto impacto:** Cuando hay mucho en juego y se necesita la validación de alguien que sepa mucho del tema antes de avanzar.
- **Nuevos dominios:** Cuando el equipo técnico nunca trabajó con esa tecnología o negocio específico y necesita la guía de un especialista.

1. ¿Qué es y para qué sirve un indicador?
Página 18
Definición: Un indicador es un elemento de información que se construye combinando distintas métricas.
Para qué sirve: Sirve para evaluar el estado actual de un proceso o proyecto y compararlo con un objetivo predefinido. Básicamente, te da una "señal" de si vas bien o mal. Permite a los gestores tomar decisiones informadas para ajustar el producto, el proceso o el proyecto.
Ejemplo: Imaginate que las "métricas" son los ingredientes (cantidad de errores encontrados, horas trabajadas), y el "indicador" es la receta final que te dice "La calidad está bajando" o "Estamos atrasados".

2. ¿Cuáles son los requerimientos no funcionales que se ven afectados por la arquitectura?
Página 66
La arquitectura del software tiene un impacto directo sobre la calidad del sistema, afectando principalmente a estos requerimientos no funcionales:
- Rendimiento: Si la arquitectura tiene demasiadas comunicaciones entre componentes, el sistema puede volverse lento. Se optimiza usando componentes de grano grueso (más grandes y menos numerosos).
- Seguridad: Una arquitectura en capas permite poner los activos más críticos en las capas internas para protegerlos mejor.
- Protección (Safety): Se suele usar un solo subsistema crítico para monitorear la seguridad, reduciendo costos de validación.
- Disponibilidad: Se logra con componentes redundantes (duplicados) para que si uno falla, otro tome su lugar sin detener el sistema.
- Mantenibilidad: Se favorece usando componentes pequeños y autocontenidos que sean fáciles de cambiar sin romper todo lo demás.

3. ¿Qué es un riesgo? ¿Cuáles son los ítems de más alto riesgo según Bohem?

Página 51, 52 y 55
- Definición de Riesgo: Es un evento no deseado que tiene dos características clave: Incertidumbre (puede pasar o no, probabilidad) y Pérdida (si pasa, trae consecuencias negativas, impacto).
- Ítems de alto riesgo (Top 10 de Boehm): Boehm sugiere monitorear los riesgos más graves. Algunos de los principales factores que menciona el documento (pág. 52) son:

1. Falta de compromiso de los usuarios finales.
2. No entender completamente los requisitos.
3. Alcance o requisitos del proyecto inestables (cambian mucho).
4. Falta de habilidades adecuadas en el equipo.
5. Expectativas no realistas.

4. ¿Cuáles son las consideraciones que se deben tener en cuenta a la hora de diseñar una interfaz?

Página 86, 87 y 95

El objetivo principal es crear una comunicación eficiente entre el humano y la máquina. Las consideraciones clave son:

- Adaptación al usuario: La tecnología debe adaptarse a la persona, no al revés. Hay que considerar quién va a usar el sistema (edad, conocimientos técnicos).
- Prevención de errores: Diseñar para que el usuario no se equivoque, ya que una interfaz confusa genera rechazo.
- Las 3 Reglas de Oro de Theo Mandel:
    1. Dar el control al usuario: Que pueda deshacer acciones y sentir que maneja el sistema.
    2. Reducir la carga de memoria: No obligar al usuario a recordar cosas de una pantalla a otra.
    3. Hacer la interfaz consistente: Que los botones y menús siempre funcionen igual en todas partes.
- Inclusividad: Permitir el uso de diversos periféricos (teclado, mouse, pantallas táctiles).

5. ¿Qué es la Gestión de Configuración? Definir línea base. Ejemplifique.

Páginas 7 y 139

- Gestión de Configuración del Software (GCS): Es el conjunto de actividades para identificar, controlar y rastrear los cambios en el software a lo largo de su vida. Asegura que sepamos qué versión estamos usando y qué cambios se hicieron.
- Línea Base (Baseline): Es un punto de referencia aprobado formalmente. Una vez que se establece una línea base (por ejemplo, "Versión 1.0 de Requisitos"), cualquier cambio futuro debe pasar por un proceso formal de aprobación. Es como poner un "seguro" a lo que ya está hecho para poder seguir avanzando sobre terreno firme.
- Ejemplo: Imaginate que terminás de escribir el "Plan del Proyecto" y el cliente lo firma. Ese documento firmado es ahora una Línea Base. Si la semana que viene querés cambiar una fecha, no podés simplemente borrarla; tenés que hacer una solicitud de cambio formal porque ya estaba aprobado.

Característica	SRS (Especificación)	GCS (Gestión de Configuración)
¿Qué es?	Un **documento** de requisitos.	Un **proceso** de control y organización.
Objetivo	Definir el producto a construir.	Controlar los cambios y versiones de todo.
Pregunta clave	¿Qué tiene que hacer el sistema?	¿Dónde está la última versión y qué cambió?
Ejemplo	"El sistema tendrá un botón de pago".	"Se guardó el cambio del botón en la rama 'main'".

6. Rejuvenecimiento.
Páginas 144 y 145

- Concepto: Es el proceso de mejorar un sistema existente para alargar su vida útil y evitar que se degrade. Es como hacerle un "lifting" al software.
- Tipos de Rejuvenecimiento:

1. Re-Documentación: Crear documentación nueva porque la vieja se perdió o no existe. Ayuda a entender qué hace el código.
2. Re-Estructuración: Ordenar el código internamente (hacerlo más limpio) sin cambiar lo que hace por fuera.
3. Ingeniería Inversa: Analizar el código para intentar recuperar el diseño original (ir de atrás para adelante).
4. Re-Ingeniería: Es lo más completo. Se analiza el sistema viejo, se rediseña y se vuelve a construir una versión mejorada.

Punto de comparación	Re-documentación	Ingeniería Inversa
Nivel de Abstracción	Se queda en el mismo nivel (Código -> Documento descriptivo).	Sube de nivel (Código -> Modelos/Diagramas de diseño).
Herramientas	Principalmente manual (vos escribiendo).	Principalmente automatizada (software que analiza código).
Objetivo	Que alguien sepa **usar** o **leer** el sistema.	Que alguien sepa **cómo está diseñado** el sistema internamente.

7. Calcule el camino crítico.

1) Tipos de estructuras de equipo (Planificación Organizativa)
Existen tres estructuras o modelos genéricos de equipo:

- Descentralizado Democrático (DD):

    - Definición: No existe un jefe permanente; se nombran coordinadores para tareas a corto plazo que luego son rotados. Las decisiones y la resolución de problemas se realizan por consenso y la comunicación es horizontal.
    - Ideal para: Equipos que permanecerán unidos por largos períodos y para problemas difíciles.
    - **Por qué para problemas difíciles:** Al no haber un jefe único, todas las ideas valen lo mismo. Esto fomenta la "lluvia de ideas" y el análisis profundo desde muchos ángulos, lo que es necesario cuando la solución no es obvia y requiere mucha creatividad o investigación.
    - **Por qué larga duración:** Como las decisiones se toman por consenso, el proceso es más lento. Esto solo funciona si el equipo tiene tiempo para conocerse, generar confianza y mantenerse unido durante mucho tiempo

- Descentralizado Controlado (DC):
    - Definición: Hay un jefe definido que coordina tareas específicas y jefes secundarios para subtareas. La resolución de problemas es grupal, pero la implementación se reparte en subgrupos. La comunicación es tanto horizontal como vertical.
    - Ideal para: Proyectos grandes que requieren dividirse en subgrupos y para abordar problemas complejos.
    - **Por qué para problemas complejos:** La complejidad se diferencia de la "dificultad" en que un problema complejo se puede **dividir en partes**. Esta estructura permite tener un jefe principal y líderes de subgrupos, lo que facilita atacar cada parte del problema por separado pero de forma coordinada.
    - **Por qué proyectos grandes:** Es la estructura que mejor escala. Podés formar subgrupos fácilmente (por ejemplo: un grupo para la base de datos, otro para la interfaz) y mantener la comunicación tanto horizontal (entre pares) como vertical (con los jefes)

- Centralizado Controlado (CC):
    - Definición: El jefe del equipo se encarga de la resolución de problemas a alto nivel y de toda la coordinación interna. Las decisiones y el enfoque son tomados únicamente por el líder. La comunicación es puramente vertical.
    - Ideal para: Tareas rápidas y problemas sencillos
    - **Porque para problemas sencillos:** Si la solución ya se conoce o el camino es claro, no hace falta perder tiempo en debates grupales o reuniones de consenso. El jefe sabe qué hay que hacer y da las órdenes directas.
    - **Porque para tareas rápidas:** Al ser la comunicación puramente vertical (de arriba hacia abajo), la toma de decisiones es inmediata. Esto permite avanzar a una velocidad que las otras estructuras más "charladas" no pueden alcanzar.

DD: Muchos cerebros pensando juntos → **Calidad máxima** en soluciones nuevas.
DC: Muchos grupos trabajando en paralelo → **Gestión de la complejidad** y el tamaño.
CC: Una sola voz mandando → **Velocidad máxima** en lo que ya se sabe hacer.

Ubicación: Página 50

2) Características de todos los riesgos

Página: 51.
Todo riesgo tiene dos características fundamentales:

- **Incertidumbre:** El riesgo puede ocurrir o no (probabilidad < 100%).
- **Pérdida:** Si el riesgo se convierte en realidad, traerá consecuencias negativas (daño o costo).

3) Definir recuperabilidad (principio de diseño)

Página: 112 (referido en pruebas de recuperación).
Aunque el PDF lo menciona principalmente en la etapa de pruebas ("Prueba de recuperación tras fallos"), como principio de diseño se refiere a la capacidad del sistema para reponerse de fallos y restablecer el funcionamiento correcto y los datos afectados en un tiempo aceptable.
En el contexto de los principios de diseño de interfaz y sistema, es fundamental para que el usuario no sienta miedo a equivocarse.

Según el material de la materia, aquí tenés la definición técnica y sus variantes:

Concepto de Recuperabilidad
Es la capacidad del sistema para permitir al usuario corregir una acción una vez que ha reconocido que se cometió un error. El objetivo es que el sistema sea "tolerante" a las equivocaciones humanas o fallas técnicas.

Existen dos formas principales de recuperabilidad:

1. Recuperación Indirecta (Recuperación de Errores): Ocurre cuando el sistema ayuda al usuario a salir de una situación de error.

Ejemplo: Un cartel de confirmación que pregunta "¿Está seguro de que desea borrar este archivo?" antes de realizar la acción, o un sistema de autoguardado que recupera los datos tras un corte de luz.

2. Recuperación Directa (Capacidad de deshacer): Es cuando el usuario puede volver exactamente al estado anterior a la acción errónea.

Ejemplo: El comando Undo (Ctrl+Z). El usuario realiza la acción, ve que no era lo que quería, y la revierte de forma inmediata.

¿Por qué es importante en el diseño?
Reduce la ansiedad: El usuario se siente más libre de explorar la herramienta si sabe que puede volver atrás.

Aumenta la eficiencia: Es mucho más rápido deshacer un cambio que tener que volver a cargar o escribir todo desde cero.

Seguridad: Evita la pérdida definitiva de información importante.

Tip para el examen: A veces se confunde con la Resiliencia. Recordá que la Recuperabilidad está más enfocada en la interacción (que el usuario pueda arreglar el error), mientras que la resiliencia es la capacidad del sistema de seguir funcionando "como sea" a pesar de los fallos.

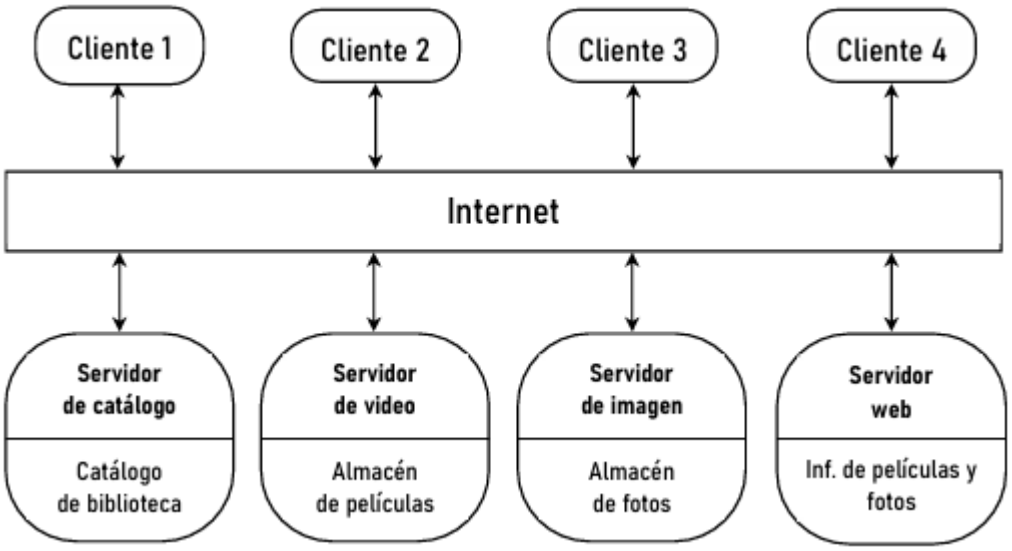
4) Diferencias entre modelo repositorio y modelo cliente-servidor

Páginas: 68 y 70.
Respuesta:

- Repositorio: Centraliza todos los datos en un solo lugar compartido. Los subsistemas interactúan a través de esta base de datos central. Es eficiente para datos grandes pero si falla el centro, falla todo.

Aunque el patrón de repositorio puede ser beneficioso en términos de centralización y compartición de datos, también introduce desafíos en cuanto a flexibilidad, adaptabilidad y mantenibilidad a largo plazo.

- Cliente-Servidor: Distribuye la carga. Hay proveedores de servicios (servidores) y consumidores (clientes). Es más fácil de escalar (agregar más servidores) y distribuir en una red.



- El patrón Cliente-Servidor es poderoso para dividir responsabilidades y permitir escalabilidad y reutilización, pero también introduce desafíos relacionados con la complejidad, la dependencia de la red y la seguridad.

5)Qué es GCS. Línea Base, ejemplifique

Páginas: 137 y 139.
Respuesta:

- GCS (Gestión de Configuración del Software): Es el proceso para identificar, controlar y gestionar los cambios en el software a lo largo de su vida.
- Línea Base: Es un punto de referencia aprobado formalmente (una "foto" del proyecto en un momento dado).
- Ejemplo: La "Especificación de Requisitos" firmada por el cliente. A partir de esa firma, cualquier cambio requiere un trámite formal.

6) Describa rejuvenecimiento.

Página: 145.
Respuesta: Es el proceso de mejorar un sistema existente para alargar su vida útil y calidad. Incluye técnicas como:

- Re-documentación: Crear manuales nuevos del código.
- Re-estructuración: Ordenar el código internamente.
- Ingeniería Inversa: Analizar el código para recuperar el diseño.
- Re-ingeniería: Rediseñar y reconstruir el sistema.

1) Describa el problema de las “4 p”.

Página: 16
El problema es que muchas veces los gestores se enfocan demasiado en cuestiones técnicas y descuidan la gestión equilibrada de los cuatro elementos clave. Si alguno de estos factores se ignora o se gestiona mal, el proyecto tiende al fracaso. :

1. Personal (la gente),

2. Producto (qué se hace),
3. Proceso (cómo se hace) y
4. Proyecto (la planificación).

El descuido del personal es la causa más común de fracaso.

Es un problema porque el gestor de software debe hacer un "malabarismo" constante entre ellas. Por ejemplo:

- Si el Producto cambia (alcance), afecta al Proyecto (tiempos) y al Proceso (tareas).
- Si el Personal tiene un conflicto, el Proyecto se atrasa.

2)¿Cuáles son los elementos claves de la gestión de proyectos?

Página: 16.
Respuesta: Son las 4 P mencionadas arriba: Personal, Producto, Proceso y Proyecto.

Los elementos clave son:

1. El Personal y su Organización
No solo es tener a la gente, sino cómo se estructura. El gestor debe definir:
- Roles y responsabilidades: Quién hace qué.
- Estructura del equipo: Elegir si será un equipo democrático (DD), controlado (DC) o centralizado (CC).
- Mecanismos de comunicación: Cómo se va a compartir la información para evitar malentendidos.

2. El Enfoque en el Problema (El Producto)
Un gestor no puede planificar si no entiende qué está construyendo.
- Definición del Alcance: Establecer los límites del proyecto (qué entra y qué queda afuera).
- Análisis de Requerimientos: Asegurar que el SRS sea sólido antes de empezar a asignar tareas.

3. El Proceso de Software
Es la elección de la metodología de trabajo.
- El gestor debe decidir qué modelo de ciclo de vida (Incremental, Cascada, Espiral, Ágil) se adapta mejor al tipo de producto y al cliente. Esto define las etapas y los hitos del proyecto.

4. La Planificación y el Control (El Proyecto)
Es el seguimiento diario para que el proyecto no se descarrile.
- Estimación: Calcular cuánto tiempo y dinero va a llevar (usando métricas como Puntos Función o COCOMO).
- Calendarización: Crear el cronograma (usando herramientas como diagramas de Gantt o PERT).
- Gestión de Riesgos: Identificar qué puede salir mal y tener un plan B.

Resumen para responder rápido:

Si te preguntan cuáles son, podés decir que son la organización del personal, la definición del producto/alcance, la selección del proceso adecuado y el control del cronograma y riesgos.

3)Describa el modelo MOI.
Página: 48.
Respuesta: Es un modelo de liderazgo para gestionar equipos:

- M (Motivación): Incentivar al personal para que rinda al máximo.
- O (Organización): Estructurar el equipo y los procesos para lograr el producto.
- I (Innovación/Ideas): Fomentar la creatividad y nuevas soluciones.

4) ¿Qué define el diseño arquitectónico? Describa los tipos de organización del sistema.
Páginas: 58 y 67.
Respuesta: Define la relación entre los principales elementos estructurales del software. Los tipos de organización son:

- Repositorio (datos centralizados).
- Cliente-Servidor (servicios distribuidos).
- Máquina Abstracta o Capas (organización jerárquica donde cada capa sirve a la superior).

El diseño arquitectónico define la estructura de alto nivel del software. Su objetivo es identificar los componentes de software principales, sus propiedades externas (interfaces) y las relaciones o comunicaciones entre ellos.
Se puede decir que define:

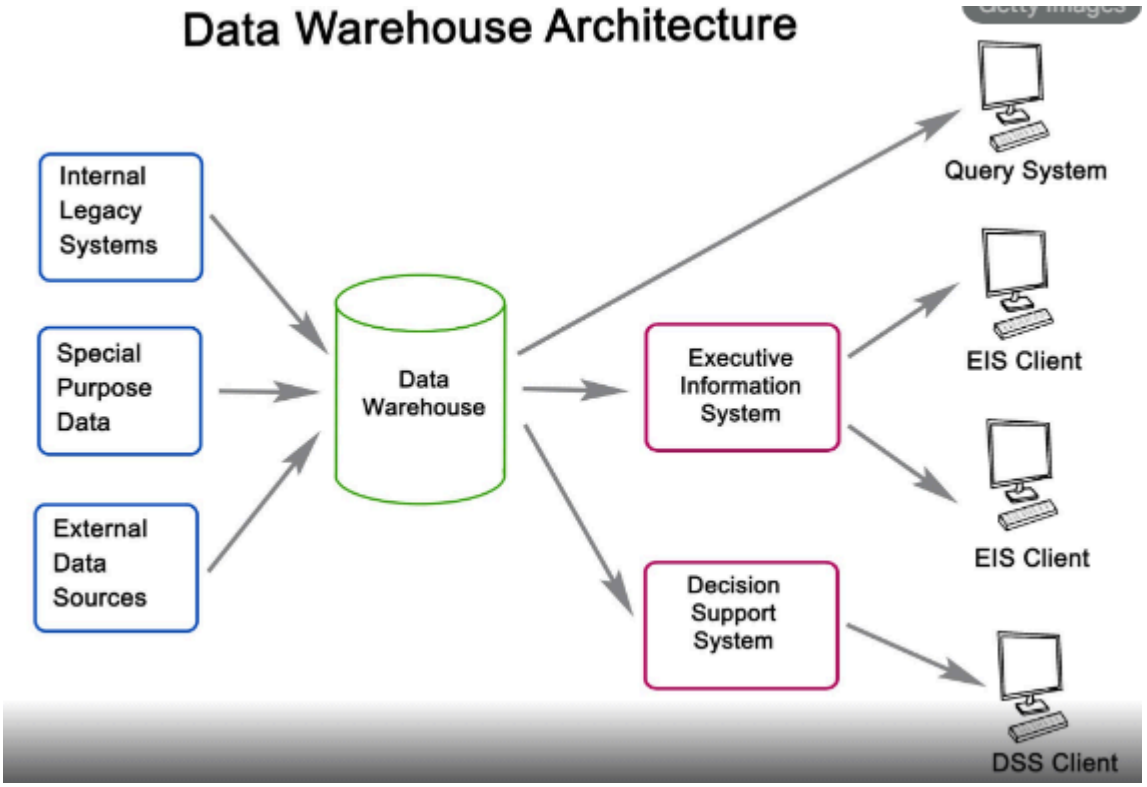
- Los componentes: Los "bloques" que forman el sistema.
- Los conectores: Cómo se comunican esos bloques.
- Las restricciones: Las reglas que debe seguir el diseño (ej: seguridad, rendimiento).

---

2. Tipos de organización del sistema
La organización del sistema (o estilos arquitectónicos) determina cómo se distribuyen los datos y el procesamiento. Según el material de la cátedra, existen tres tipos principales:

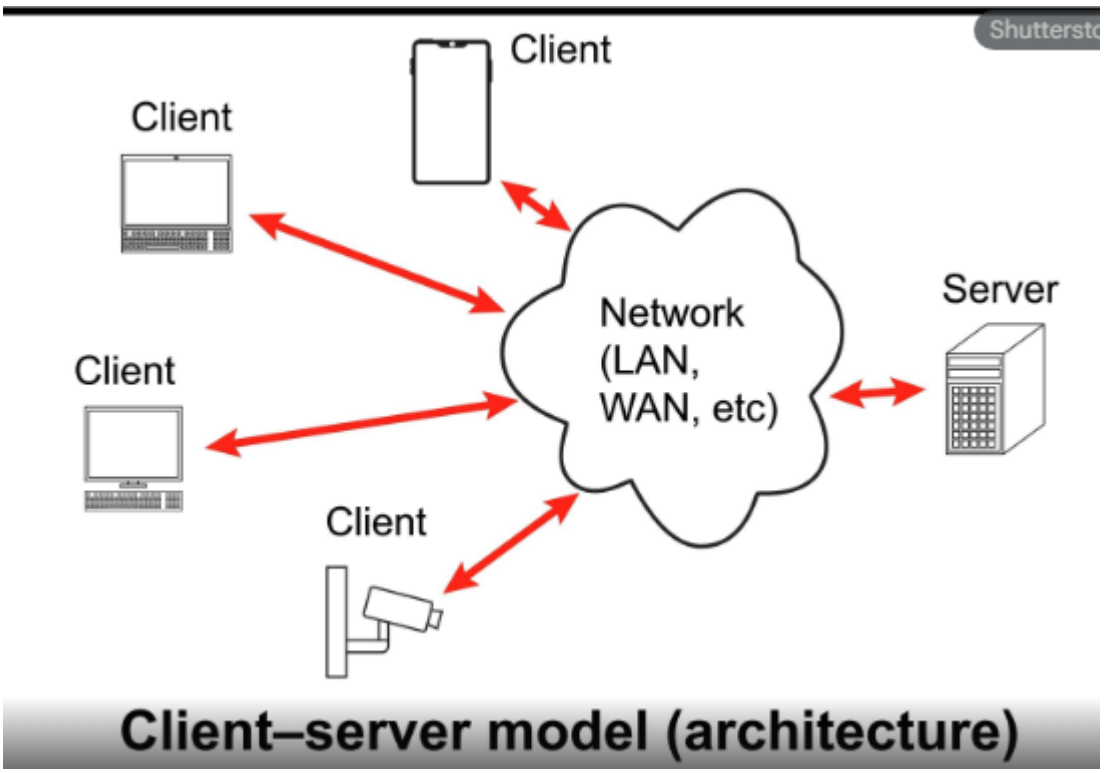
A. Arquitectura de Repositorio (Modelo de Datos Compartido): En este estilo, todos los componentes del sistema acceden a una base de datos central o repositorio. Es ideal cuando se manejan grandes volúmenes de datos que muchos sub-sistemas deben compartir.

- Cómo funciona: Los componentes son independientes; no se comunican entre sí, sino que todos "leen y escriben" en el mismo lugar central.
- Ventaja: Es eficiente para compartir datos.
- Desventaja: El repositorio es un "cuello de botella"; si falla la base de datos central, se cae todo el sistema.



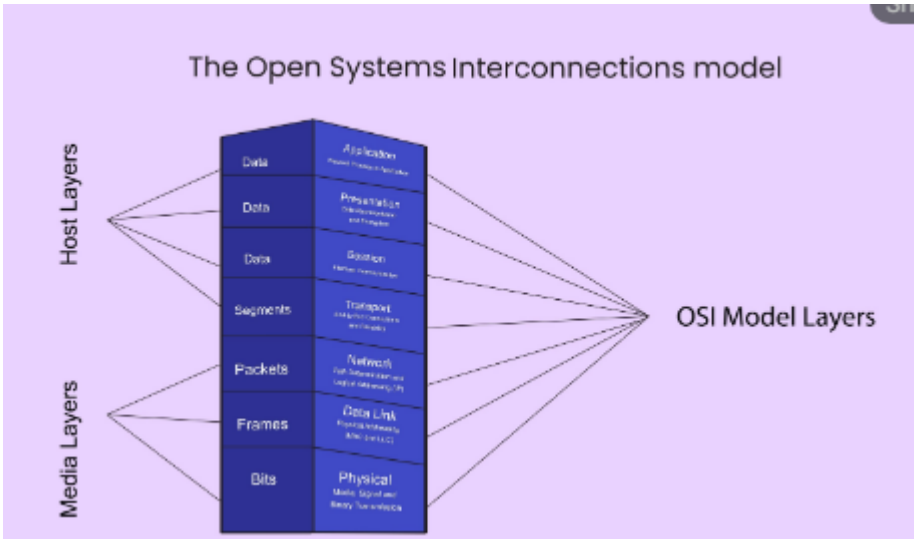
B. Modelo de Cliente-Servidor (Servicios Distribuidos): Divide el sistema en dos roles claros: los proveedores de servicios (servidores) y los usuarios de esos servicios (clientes).

- Cómo funciona: El servidor ofrece funcionalidades (ej: base de datos, impresión) y los clientes las solicitan a través de una red.
- Ventaja: Es muy escalable. Podés agregar más clientes sin afectar al servidor, y el procesamiento está distribuido.
- Desventaja: Depende totalmente de la red. Si no hay conexión, el cliente no puede hacer nada.



C. Modelo de Máquina Abstracta (Arquitectura en Capas): Organiza el sistema en una serie de capas, donde cada una ofrece servicios a la capa superior y utiliza los servicios de la capa inferior.

- Cómo funciona: Es como una cebolla. Por ejemplo, la capa superior es la Interfaz de Usuario, la del medio es la Lógica de Negocio y la inferior es la Base de Datos.
- Ventaja: Es muy fácil de mantener. Si querés cambiar la base de datos, solo tocás esa capa sin romper la interfaz.
- Desventaja: Puede ser menos eficiente, ya que un pedido tiene que "atravesar" todas las capas para obtener una respuesta.



- **Diseño Arquitectónico:** Es el "mapa" o plano general del sistema.
- **Repositorio:** Todos a un mismo baúl central.
- **Cliente-Servidor:** Uno pide, otro da (distribuido).
- **Máquina Abstracta (Capas):** Una estructura jerárquica (una capa sobre otra).

5) Enumere y describa los principios de diseño.

Página: 59.
Respuesta:

- Modularidad: Dividir en partes pequeñas e independientes.
- Abstracción: Ocultar detalles complejos.
- Refinamiento: Ir de lo general a lo detallado.
- Cohesión: Que cada módulo haga una sola cosa.
- Acoplamiento: Que los módulos dependan poco entre sí.

Los principios de diseño son reglas generales que guían al diseñador para crear sistemas de alta calidad, fáciles de usar y mantener. estos principios se centran en la interacción entre el usuario y el sistema.

1. Anticipación
El sistema debe prever las necesidades del usuario de modo que la información y las herramientas necesarias estén disponibles en el momento preciso.
Objetivo: Evitar que el usuario tenga que buscar funciones o datos en menús escondidos cuando es obvio que los va a necesitar.

2. Autonomía
El usuario debe sentir que tiene el control del sistema. El software debe ser un entorno donde el usuario tome las decisiones, no donde el sistema lo obligue a seguir pasos rígidos sin libertad.
Objetivo: Proporcionar un ambiente flexible donde el usuario inicie las acciones.

3. Consistencia
El sistema debe comportarse de la misma manera en situaciones similares. Se divide en:
Consistencia interna: El diseño (colores, botones, términos) debe ser igual en todas las pantallas de la app.
Consistencia externa: El sistema debe seguir las convenciones ya establecidas en el mundo del software (ej: que el ícono de un diskette siempre signifique "guardar").

4. Eficiencia del Usuario
El diseño debe estar orientado a maximizar la productividad del usuario, no la del motor del software. No importa si el código es complejo, lo importante es que el usuario haga su tarea rápido y con pocos clics.

5. Relevancia (Enfoque)
El sistema debe mostrar solo la información relevante para la tarea actual, evitando saturar al usuario con datos innecesarios que lo distraigan del objetivo principal.

6. Protección del Trabajo del Usuario

El sistema debe garantizar que el trabajo del usuario nunca se pierda por un error.
Ejemplo: Guardado automático, pedidos de confirmación antes de cerrar sin guardar o recuperación tras un fallo.

7. Legibilidad
La información presentada (textos, gráficos, íconos) debe ser fácil de leer y comprender visualmente. Esto incluye el buen uso de contrastes, tipografías y espacios en blanco.

8. Navegación Fácil
El usuario debe saber siempre dónde está, de dónde viene y hacia dónde puede ir. El diseño debe minimizar la carga cognitiva necesaria para moverse por las distintas pantallas.

9. Recuperabilidad
Es la capacidad de permitir al usuario corregir un error.
Recuperación Directa: El comando "Deshacer" (Undo).
Recuperación Indirecta: Mensajes de advertencia o confirmación.

6) Defina y describa GCS.

Páginas: 137 y 139.
Respuesta:

- GCS (Gestión de Configuración del Software): Es el proceso para identificar, controlar y gestionar los cambios en el software a lo largo de su vida.
- Línea Base: Es un punto de referencia aprobado formalmente (una "foto" del proyecto en un momento dado).
- Ejemplo: La "Especificación de Requisitos" firmada por el cliente. A partir de esa firma, cualquier cambio requiere un trámite formal.

7)Ejercicio de PERT: Hallar tiempos tempranos - tardíos - caminos críticos y duración total del proyecto .
Páginas: 40-42.
Método: Para hallar los tiempos, debés dibujar la red de tareas.

- Tiempos Tempranos: Se calculan de inicio a fin (sumando duraciones).
- Tiempos Tardíos: Se calculan de fin a inicio (restando duraciones).
- Camino Crítico: Es la ruta donde el Tiempo Temprano es igual al Tiempo Tardío (holgura = 0). Determina la duración total del proyecto.

1. ¿Qué es un proyecto? Describa el problema de las 4 “P”.
Página: 16.
Respuesta: Un proyecto es un esfuerzo planificado y controlado para crear un producto, gestionando recursos y riesgos. (Para las 4P).

1. El Personal y su Organización
No solo es tener a la gente, sino cómo se estructura. El gestor debe definir:

Roles y responsabilidades: Quién hace qué.

Estructura del equipo: Elegir si será un equipo democrático (DD), controlado (DC) o centralizado (CC).

Mecanismos de comunicación: Cómo se va a compartir la información para evitar malentendidos.

2. El Enfoque en el Problema (El Producto)
Un gestor no puede planificar si no entiende qué está construyendo.

Definición del Alcance: Establecer los límites del proyecto (qué entra y qué queda afuera).

Análisis de Requerimientos: Asegurar que el SRS sea sólido antes de empezar a asignar tareas.

3. El Proceso de Software
Es la elección de la metodología de trabajo.

El gestor debe decidir qué modelo de ciclo de vida (Incremental, Cascada, Espiral, Ágil) se adapta mejor al tipo de producto y al cliente. Esto define las etapas y los hitos del proyecto.

4. La Planificación y el Control (El Proyecto)
Es el seguimiento diario para que el proyecto no se descarrile.

Estimación: Calcular cuánto tiempo y dinero va a llevar (usando métricas como Puntos Función o COCOMO).

Calendarización: Crear el cronograma (usando herramientas como diagramas de Gantt o PERT).

Gestión de Riesgos: Identificar qué puede salir mal y tener un plan B.

Resumen para responder rápido:
Si te preguntan cuáles son, podés decir que son la organización del personal, la definición del producto/alcance, la selección del proceso adecuado y el control del cronograma y riesgos.

2. En Planificación temporal, ¿Qué tareas se deben realizar para hacer el seguimiento y control del proyecto?

Páginas: 17 y 19.
Respuesta: Se debe monitorear el cronograma regularmente comparando la fecha real vs. planificada, gestionar el alcance para evitar cambios descontrolados, y analizar riesgos. Se usan métricas de "Estado" y "Uso del tiempo"

Las tareas o acciones que se deben realizar para el control, particularmente ante la detección de retrasos, son las siguientes:

- Revisar el impacto sobre la fecha de entrega: Se debe analizar si el retraso ocurre en una tarea crítica (lo que afectaría todo el proyecto) o en una con cierta flexibilidad (margen de tiempo).
- Reasignar recursos: Implica mover recursos a las tareas retrasadas para acelerarlas. Sin embargo, el apunte advierte que esto debe hacerse con cuidado, ya que "aumentar el número de personas no siempre resulta en un aumento directo de la productividad" y puede causar confusión.
- Reordenar tareas: Consiste en reorganizar la secuencia de actividades para minimizar el impacto del retraso. Esta acción debe evaluarse bien para no comprometer la calidad.
- Modificar la entrega: Si los retrasos son inevitables y graves, se puede considerar ajustar la fecha de entrega final o realizar entregas parciales, lo cual suele requerir negociación con los involucrados (stakeholders).

Para la gestión adecuada (página 16), el control implica la monitorización del progreso y la toma de medidas para mantener el proyecto alineado con sus objetivos.

3. ¿Cuáles son los tipos (áreas) de Diseño de Software? Describa.

Página: 58.
Respuesta:

- Diseño de Datos: Estructuras de datos y bases de datos.
- Diseño Arquitectónico: Estructura general del sistema.
- Diseño de Interfaz: Comunicación usuario-sistema.
- Diseño de Componentes: Detalle procedimental de cada módulo.

4. ¿Qué es Verificación? ¿Qué es Validación? De todos los tipos de pruebas vistos, ¿Cuál dirías que se está verificando y cual que se está validando al usar la prueba?

Página: 120.
Respuesta:

- Verificación: ¿Estamos construyendo el producto correctamente? (Cumplir especificaciones técnicas). Qué significa: Es el proceso de comprobar si el software cumple con los requisitos especificados (el SRS) y los estándares de diseño. Se centra en la calidad técnica y en asegurar que no haya "bugs" o desvíos de lo que se escribió en los documentos. Foco: El proceso de desarrollo.
- Validación: ¿Estamos construyendo el producto correcto? (Cumplir necesidades del cliente). Qué significa: Es el proceso de comprobar si el software realmente satisface las necesidades y expectativas del cliente. Puede que el software no tenga errores técnicos (esté verificado), pero que no le sirva al usuario para lo que quería (no está validado). Foco: El usuario final y su satisfacción.
- En pruebas: Una prueba de Caja Blanca suele verificar (lógica interna), y una de Caja Negra suele validar (requisitos funcionales).

Basándonos en los tipos de pruebas vistos en la materia, la división se hace según quién realiza la prueba y contra qué se compara:

A. Pruebas de verificación (enfoque técnico)
Acá estamos comprobando que el código hace lo que dicen los planos (diseño y especificaciones).

- Pruebas de unidad: Verifican que cada pequeña pieza de código (función o método) funcione bien por separado.
- Pruebas de integración: Verifican que las piezas, al unirse, se comuniquen bien (que no fallen las interfaces).
- Pruebas de sistema: Verifican que todo el sistema completo cumpla con los Requerimientos Funcionales y No Funcionales del SRS. (Nota: Aunque se prueba todo el sistema, sigue siendo verificación porque se compara contra el documento técnico SRS, no necesariamente con el usuario real).

B. Pruebas de validación (Enfoque de Usuario)
Acá estamos comprobando que el sistema le sirve a la persona real.

- Pruebas de aceptación (UAT): Es la validación por excelencia. El cliente usa el sistema para decidir si lo acepta o no.
- Pruebas Alpha y Beta:
    - Alpha: Usuario prueba en el entorno del desarrollador.
    - Beta: Usuario prueba en su propio entorno real.
    - En ambos casos, el usuario está validando si el producto le es útil en la vida real.

Resumen:
Si chequeás contra el papel/documento (SRS, Diseño) → Estás Verificando (Pruebas de Unidad, Integración, Sistema).

Si chequeás contra la necesidad del usuario (Realidad) → Estás Validando (Pruebas de Aceptación/Alpha/Beta).

5. Describa el Proceso de GCS.

Página: 143 (Gestión de cambios).
Respuesta: Implica: 1) Identificar los ítems a controlar, 2) Controlar las versiones, 3) Controlar los cambios (solicitud, evaluación, aprobación), 4) Auditorías de configuración, 5) Informes de estado.

6. ¿Qué es el rejuvenecimiento del software? Describa los tipos.

Es el proceso de modificar un sistema heredado (viejo) para que siga siendo útil y rentable. El objetivo no es necesariamente cambiar lo que el sistema hace, sino mejorar su estructura interna para que sea más fácil de mantener, entender y evolucionar.
Es como hacerle un "lifting" o mantenimiento profundo al software para alargar su vida útil.

Los Tipos de Rejuvenecimiento. → Tipos principales que forman parte de este proceso:

1. Re-documentación: Es la creación de nueva documentación para un sistema que no la tiene, que está desactualizada o es incomprensible.
   Objetivo: Hacer que el código sea legible y entendible para los programadores.
   Diferencia clave: Se hace mediante análisis estático del código (leyéndolo) y no cambia ni una sola línea de código, solo genera papeles/archivos de texto.
2. Re-estructuración: Es el proceso de reorganizar la estructura lógica del sistema sin cambiar su comportamiento externo.
   Objetivo: Convertir el "código espagueti" (desordenado) en un código limpio y modular. Ejemplo: Cambiar nombres de variables confusas, simplificar condicionales complejos (IF anidados) o modularizar funciones gigantes.
3. Ingeniería Inversa: Es el proceso de analizar el código fuente para identificar los componentes del sistema y sus relaciones, con el fin de crear representaciones en un nivel más alto de abstracción (diseño).
   Objetivo: Recuperar el diseño perdido. Ir del Código al Diseño. Clave: No altera el sistema, solo busca entenderlo y modelarlo.
4. Reingeniería (El proceso completo): Es la modificación completa del sistema. Implica tomar el sistema viejo, entenderlo y volver a implementarlo en una nueva forma. El ciclo: Normalmente incluye hacer Ingeniería Inversa (para entender el viejo) → Re-estructuración (para mejorarlo) → Ingeniería hacia adelante (para construir el nuevo).
   Objetivo: Migrar a nuevas tecnologías o arquitecturas manteniendo la funcionalidad del negocio.

   Resumen rápido para estudiar:
   1. Re-documentar → Escribir el manual que falta.
   2. Re-estructurar → Ordenar el código sucio.
   3. Ingeniería inversa → Sacar el plano a partir de la obra construida.
   4. Reingeniería → Renovar todo el sistema usando lo anterior.

7. La empresa de pastas HOGAREÑAS S.A. desea hacer un estudio del tiempo que demora en hacer una lasaña, para esto ha identificado las siguientes actividades (en minutos). Encontrar el camino crítico, calculando tiempos tempranos y tiempos tardíos. Calcular el tiempo final.

1. ¿Qué es un riesgo? ¿Cómo se clasifican?

Página: 51.
Un riesgo se define como un evento no deseado que conlleva consecuencias negativas.

- Riesgos del Proyecto: Afectan la planificación (costos, plazos).
- Riesgos Técnicos: Afectan la calidad y diseño del software.
- Riesgos del Negocio: Afectan la viabilidad comercial del producto.

2. ¿Cómo puede ser la organización del sistema? (diseño arquitectónico).

Los estilos arquitectónicos o modelos de diseño definen cómo se estructuran y comunican los componentes del software. Existen tres formas principales de organizar un sistema:
1. Modelo de Repositorio (Datos Compartidos)

En este modelo, todos los subsistemas o componentes comparten una gran estructura de datos central (base de datos o repositorio).

- Cómo funciona: Los componentes no se comunican directamente entre sí, sino que interactúan únicamente con el repositorio central. Cuando un subsistema modifica un dato, los demás pueden verlo.
- Ventajas: Es eficiente para compartir grandes volúmenes de datos; los componentes pueden ser independientes (si agregás uno nuevo, no rompe a los otros).
- Desventajas: El repositorio es un punto único de fallo (si se cae, se cae todo) y puede convertirse en un cuello de botella de rendimiento.

2. Modelo Cliente-Servidor (Distribución)
El sistema se organiza como un conjunto de servicios y servidores asociados, y unos clientes que usan esos servicios.

- Cómo funciona:
    - Servidores: Ofrecen servicios (impresión, base de datos, cálculo).
    - Clientes: Usuarios o programas que solicitan esos servicios.
    - Red: Es el medio que los conecta.
- Ventajas: Es escalable (podés agregar muchos servidores) y flexible (los servidores pueden estar distribuidos geográficamente).
- Desventajas: El intercambio de datos a través de la red puede ser lento y cada servicio es un punto de fallo potencial.

3. Modelo de Máquina Abstracta (Capas)
El sistema se organiza en una jerarquía de capas, donde cada capa ofrece servicios a la capa superior y utiliza servicios de la capa inferior.

- Cómo funciona: Es como una cebolla.
    - Capa Base: Manejo de hardware o base de datos.
    - Capas Intermedias: Lógica del negocio, utilidades.
    - Capa Superior: Interfaz de usuario.
- Regla de oro: Una capa solo "habla" con sus vecinas inmediatas.
- Ventajas: Es fácil de mantener y cambiar (si cambiás la base de datos, solo tocás la capa inferior, no la interfaz).
- Desventajas: Puede perder eficiencia porque cada pedido tiene que atravesar múltiples capas.

3. ¿En qué se diferencian la verificación a la validación? Enumere las pruebas de integración que conozca.

- Verificación: ¿Estamos construyendo el producto correctamente? (Cumplir especificaciones técnicas). Qué significa: Es el proceso de comprobar si el software cumple con los requisitos especificados (el SRS) y los estándares de diseño. Se centra en la calidad técnica y en asegurar que no haya "bugs" o desvíos de lo que se escribió en los documentos. Foco: El proceso de desarrollo.
- Validación: ¿Estamos construyendo el producto correcto? (Cumplir necesidades del cliente). Qué significa: Es el proceso de comprobar si el software realmente satisface las necesidades y expectativas del cliente. Puede que el software no tenga errores técnicos (esté verificado), pero que no le sirva al usuario para lo que quería (no está validado). Foco: El usuario final y su satisfacción.

Pruebas de Integración:

- **Descendente (Top-Down):** De arriba hacia abajo (usa stubs).
- **Ascendente (Bottom-Up):** De abajo hacia arriba (usa drivers).
- **Sándwich:** Combina ambas.
- **Big Bang:** Todo junto (no recomendada).

Las pruebas de integracion: son la fase de pruebas que ocurre justo después de probar cada pieza por separado (Pruebas Unitarias). En esta etapa, se toman esos módulos individuales y se ensamblan (se integran) para probarlos como un grupo.

Su objetivo principal no es buscar errores dentro del código de una función (eso ya lo hizo la prueba unitaria), sino detectar errores en la interfaz y la comunicación entre los componentes.
Ejemplo: El módulo A calcula un precio y se lo pasa al módulo B para que lo guarde.
La prueba unitaria dice que A calcula bien.
La prueba unitaria dice que B guarda bien.
La Prueba de Integración chequea si A le está pasando el dato en el formato correcto a B.
Si A manda texto y B espera números, la integración falla.

Enumeración de las Pruebas de Integración (Estrategias)
Las estrategias de integración se dividen principalmente en No Incremental (todo de golpe) e Incremental (paso a paso).

1. Integración "Big Bang" (No Incremental)

Se combinan todos los componentes a la vez y se prueba todo el programa de golpe.
Desventaja: Si hay un error, es muy difícil saber qué módulo lo causó porque está todo mezclado. Es un caos para sistemas grandes.

2. Integración Descendente (Top-Down)

Se empieza probando el módulo principal (la interfaz o menú principal) y se va bajando hacia los módulos auxiliares.
Herramienta clave: Se usan Stubs (falsos módulos inferiores) para simular las partes que aún no están integradas.
Ventaja: Se ve la funcionalidad principal rápido.

3. Integración Ascendente (Bottom-Up)

Se empieza probando los módulos más bajos (acceso a datos, cálculos básicos) y se va subiendo hacia la interfaz.
Herramienta clave: Se usan Drivers (controladores) para simular que alguien está llamando a esas funciones.
Ventaja: Asegura que la base y el procesamiento de datos sean sólidos antes de ponerles la interfaz encima.

4. Integración Sándwich (o Híbrida)

Combina las dos anteriores. Se integran las capas superiores (Top-Down) y las inferiores (Bottom-Up) simultáneamente, encontrándose en el medio.

Resumen:
Las pruebas son: Big Bang, Top-Down (usa Stubs), Bottom-Up (usa Drivers) y Sándwich. Sirven para ver si los módulos "se hablan" bien entre sí.

4. ¿Qué tipos de diseño de software conoce?

El diseño de software se divide en **4 áreas o tipos principales**. Cada una se enfoca en una capa distinta de la construcción del sistema:

1. Diseño de Datos →Transforma el modelo de dominio (la información que obtuviste en el análisis) en las estructuras de datos, objetos y relaciones que va a usar el programa.
Objetivo: Definir cómo se van a almacenar y gestionar los datos dentro del sistema.

2. Diseño arquitectónico → Define la relación entre los principales elementos estructurales del software. Aca es donde se eligen los estilos (como Cliente-Servidor o Capas) y los patrones de diseño.
Objetivo: Dar la estructura general y el "esqueleto" del sistema.

3. Diseño a Nivel de Componentes → Transforma los elementos de la arquitectura en una descripción detallada de cada módulo o función. Utiliza modelos de clases, flujos y comportamientos para especificar la lógica interna.
Objetivo: Definir exactamente "cómo funciona" cada pieza por dentro.

4. Diseño de Interfaz → Describe cómo se comunica el software, tanto consigo mismo (internamente), con otros sistemas y, lo más importante, con las personas.
Objetivo: Definir la interacción del usuario y la presentación de la información.

Adicional: Diseño UX / UI
Más adelante en el apunte (páginas 86 y 89), se profundiza en el diseño de interfaz dividiéndolo en dos enfoques modernos:

Diseño UI (Interfaz de Usuario): Se centra en lo visual (botones, colores, tipografía) y en la interacción directa con la pantalla.

Diseño UX (Experiencia de Usuario): Abarca la totalidad de la experiencia (emocional, funcional), buscando que el producto sea útil, usable y deseable.

Resumen:
Si te preguntan los tipos generales de diseño, mencioná las 4 áreas: Datos, Arquitectura, Componentes e Interfaz. Si te preguntan específicamente sobre la interacción con el usuario, podés ampliar con UI y UX.

5. Cómo es el proceso de GCS.

Páginas: 137 y 139.
Respuesta:

- GCS (Gestión de Configuración del Software): Es el proceso para identificar, controlar y gestionar los cambios en el software a lo largo de su vida.
- Línea base: Es un punto de referencia aprobado formalmente (una "foto" del proyecto en un momento dado).
- Ejemplo: La "Especificación de Requisitos" firmada por el cliente. A partir de esa firma, cualquier cambio requiere un trámite formal.

6. Dada la siguiente tabla cuál o cuáles son el/los camino/s críticos? ¿Cuándo terminará el proyecto? Si llueve durante el periodo de excavación, ¿cuánto tiempo se retrasa el proyecto?

7. Explique el P-CMM (modelo de capacitación y motivación del personal)
Página: 48.
El PDF no utiliza explícitamente las siglas "P-CMM" (People Capability Maturity Model), pero cubre este concepto bajo la Planificación Organizativa y el modelo MOI (Motivación, Organización, Innovación), destacando que el personal es el activo más importante y se debe gestionar su capacidad y motivación para el éxito del proyecto.

1. Enumere y describa las técnicas de estimación que conozca.
Página: 33-34.

- Juicio experto Consultar a especialistas.
- Delphi: Expertos estiman anónimamente y discuten hasta consensuar.
- Planning Poker: Estimación ágil con cartas (Fibonacci) en equipo.
- Puntos de función / LDC: Basadas en fórmulas matemáticas sobre el tamaño del software.

1. Juicio Experto (Expert Judgment) Es una de las técnicas más utilizadas. Consiste en consultar a una o más personas con amplia experiencia en proyectos similares y en el dominio de la aplicación para que den su estimación.
   Ventaja: Es rápido y aprovecha la intuición humana.
   Desventaja: Es subjetivo. Si el experto tiene un sesgo o está de mal humor, la estimación falla.
   Variante formal: El Método Delphi. Aca, un grupo de expertos realiza estimaciones anónimas, se comparan, se discuten las diferencias y se vuelve a estimar hasta llegar a un consenso, evitando que la opinión de uno domine al resto.

2. Estimación por Analogía Se basa en comparar el proyecto actual con uno anterior muy similar (histórico) del cual ya conocemos los datos reales de esfuerzo y costo.
   Cómo funciona: Se dice "Este proyecto es como el Proyecto X, pero un 20% más grande", y se ajustan los valores.
   Requisito: Tener una base de datos confiable de proyectos pasados.

3. Estimación por Descomposición (Bottom-Up) Consiste en dividir el proyecto en piezas más pequeñas (funciones, tareas o módulos) utilizando la WBS (Work Breakdown Structure).
   Cómo funciona: Se estima el esfuerzo de cada pequeña parte por separado (que es más fácil de visualizar) y luego se suman todas para obtener el total.
   Ventaja: Es más precisa porque entra en detalle.

Desventaja: Lleva mucho tiempo realizarla.

4. Modelos Empíricos / Algorítmicos: Utilizan fórmulas matemáticas derivadas de datos históricos para predecir el esfuerzo. Relacionan el tamaño del software (Líneas de Código o Puntos de Función) con el esfuerzo.
   El principal exponente: COCOMO II (Constructive Cost Model).
   Cómo funciona: Se introduce el tamaño estimado y se aplican "conductores de costo" (multiplicadores que dependen de la complejidad, experiencia del equipo, herramientas, etc.) para obtener la cantidad de meses-persona.

5. Planning Poker (Técnica Ágil): Es una técnica basada en el consenso del equipo, muy usada en metodologías ágiles (Scrum).
   Cómo funciona: Cada miembro del equipo tiene un mazo de cartas con números (generalmente la serie de Fibonacci: 1, 2, 3, 5, 8, 13...).
   Se lee una historia de usuario.
   Todos muestran su carta al mismo tiempo (para no influenciarse mutuamente).
   Si hay diferencias grandes (uno tiró un 2 y otro un 13), discuten por qué y vuelven a votar hasta llegar a un acuerdo.
   Ventaja: Involucra a quienes realmente van a hacer el trabajo y descubre malentendidos sobre los requerimientos.

6. Ley de Parkinson y "Ganar para Perder" (Pricing to Win) A veces mencionadas como "anti-patrones" o técnicas no recomendadas, pero que existen en la realidad:
   Ley de Parkinson: El trabajo se expande hasta llenar el tiempo disponible. Se estima según el plazo límite ("tenemos 3 meses, así que tardaremos 3 meses").
   Pricing to Win: Se estima lo que el cliente quiere escuchar o lo necesario para ganar el contrato, independientemente de la realidad técnica.

Resumen:
Si tenés que elegir las más importantes para desarrollar, enfocate en: Juicio Experto, COCOMO (Algorítmica) y Planning Poker, ya que cubren los tres enfoques: experiencia humana, matemática y consenso de equipo.

pagina 33:

- Juicio Experto → En esta técnica, se consulta a varios expertos en el campo relevante para obtener sus opiniones y estimaciones. Los expertos estiman, comparan y discuten entre sí para llegar a una solución conjunta.
- Técnica Delphi→ La técnica Delphi involucra a un grupo de expertos que estiman de manera anónima. En cada ronda, se recopilan las estimaciones individuales y se proporcionan a todos los participantes sin revelar la identidad de los demás. Luego, los expertos vuelven a estimar, teniendo en cuenta las opiniones de los demás. Este proceso se repite en sucesivas rondas hasta que se alcanza un consenso. La anonimidad permite evitar la influencia de la opinión de otros participantes.
- División de Trabajo Jerárquica → Esta técnica implica la consulta a personas en diferentes niveles jerárquicos de la organización. Comienza consultando a quienes están en niveles más bajos y avanza hacia arriba en la jerarquía. Cada nivel proporciona su estimación basada en su comprensión del proyecto y de cómo se verá afectado por sus roles.

2. Que es la gestión de configuración. Defina línea de base. Ejemplifique.

Páginas: 137 y 139.
Respuesta:

- GCS (Gestión de Configuración del Software): Es el proceso para identificar, controlar y gestionar los cambios en el software a lo largo de su vida.
- Línea base: Es un punto de referencia aprobado formalmente (una "foto" del proyecto en un momento dado).
- Ejemplo: La "Especificación de Requisitos" firmada por el cliente. A partir de esa firma, cualquier cambio requiere un trámite formal.

3. Describa el rejuvenecimiento.

Es el proceso de modificar un sistema heredado (viejo) para que siga siendo útil y rentable. El objetivo no es necesariamente cambiar lo que el sistema hace, sino mejorar su estructura interna para que sea más fácil de mantener, entender y evolucionar.
Es como hacerle un "lifting" o mantenimiento profundo al software para alargar su vida útil.

Los Tipos de Rejuvenecimiento. → Tipos principales que forman parte de este proceso:

5. Re-documentación: Es la creación de nueva documentación para un sistema que no la tiene, que está desactualizada o es incomprensible.
   Objetivo: Hacer que el código sea legible y entendible para los programadores.
   Diferencia clave: Se hace mediante análisis estático del código (leyéndolo) y no cambia ni una sola línea de código, solo genera papeles/archivos de texto.
6. Re-estructuración: Es el proceso de reorganizar la estructura lógica del sistema sin cambiar su comportamiento externo.
   Objetivo: Convertir el "código espagueti" (desordenado) en un código limpio y modular. Ejemplo: Cambiar nombres de variables confusas, simplificar condicionales complejos (IF anidados) o modularizar funciones gigantes.
7. Ingeniería Inversa: Es el proceso de analizar el código fuente para identificar los componentes del sistema y sus relaciones, con el fin de crear representaciones en un nivel más alto de abstracción (diseño).
   Objetivo: Recuperar el diseño perdido. Ir del Código al Diseño. Clave: No altera el sistema, solo busca entenderlo y modelarlo.
8. Reingeniería (El proceso completo): Es la modificación completa del sistema. Implica tomar el sistema viejo, entenderlo y volver a implementarlo en una nueva forma. El ciclo: Normalmente incluye hacer Ingeniería Inversa (para entender el viejo) → Re-estructuración (para mejorarlo) → Ingeniería hacia adelante (para construir el nuevo).
   Objetivo: Migrar a nuevas tecnologías o arquitecturas manteniendo la funcionalidad del negocio.

   Resumen rápido para estudiar:
   5. Re-documentar → Escribir el manual que falta.
   6. Re-estructurar → Ordenar el código sucio.
   7. Ingeniería inversa → Sacar el plano a partir de la obra construida.

8. Reingeniería → Renovar todo el sistema usando lo anterior.

4. Una empresa de software (bla bla bla) lleva un proyecto con requisitos muy estrictos (bla bla) pero no tiene el personal necesario para cierto proyecto. Analice la siguiente situación: Es imposible encontrar el personal con las habilidades adecuadas. a) Analice el riesgo. b) Planifique (Estrategias de tratamientos de riesgos).

Caso de Riesgo (Falta de personal).

- Página: 53-54.
- a) Análisis: Es un Riesgo del Proyecto (recursos) con impacto "Serio" o "Catastrófico" y probabilidad alta (ya dice que es "imposible" encontrar).
- b) Estrategia (Planificación):
    - Mitigación: Capacitar al personal actual (aunque tome tiempo) o contratar consultores externos temporales.
    - Evitación: Reducir el alcance del proyecto para que no requiera esas habilidades específicas tan complejas.

a) Analice el riesgo
El análisis de riesgos implica estimar la probabilidad y el impacto para determinar la exposición.

1. Identificación del Riesgo:

- Nombre: Déficit de personal calificado (Staff Shortage).
- Tipo: Es un Riesgo de Proyecto (afecta la planificación y tiempos) y un Riesgo de Producto (afecta la calidad si lo hace gente inexperta).

2. Estimación (Valoración):

- Probabilidad: Muy Alta / Certeza. (El enunciado dice "Es imposible encontrar el personal", por lo que la probabilidad es prácticamente del 100%).
- Impacto: Catastrófico. (Al tener "requisitos muy estrictos", si no hay gente que sepa cumplirlos, el proyecto va directo al fracaso o a la cancelación).
- Exposición al Riesgo (RE): Como la fórmula es $RE = Probabilidad \times Impacto$, la exposición es máxima. Es un riesgo de prioridad crítica.

3. Consecuencia:

- Imposibilidad de cumplir con los plazos.
- Entregable de baja calidad (bugs, mala arquitectura).
- Pérdida de reputación de la empresa.b)

b) Planifique (Estrategias de tratamiento de riesgos)
Dado que no se puede "contratar" (porque es imposible encontrar gente), el plan debe enfocarse en alternativas.

1. Estrategia de Mitigación (reducción): Busca reducir la probabilidad o el impacto antes de que sea un desastre.

- Capacitación intensiva (Mentoring): Si no podés traer expertos de afuera, tenés que crearlos adentro. Tomar al personal actual más capaz y pagarles cursos avanzados o asignarles un mentor para que aprendan la tecnología requerida rápidamente.

- Rediseño técnico: Cambiar la tecnología del proyecto por una que el personal actual sí conozca, siempre que se puedan mantener los requisitos estrictos.

2. Estrategia de Transferencia (Compartir): Pasar el riesgo (o parte de él) a un tercero.
- Subcontratación (Outsourcing) de nicho: Quizás la empresa no puede contratar empleados fijos, pero sí puede pagarle a una consultora especializada que ya tenga ese equipo armado para que haga esa parte difícil.
- Compra en lugar de construcción (COTS): En lugar de programar el módulo difícil, comprar una solución de software ya hecha (Commercial Off-The-Shelf) que cumpla con los requisitos.

3. Estrategia de Evasión (evitar): eliminar la causa del riesgo modificando el proyecto.
- Reducción de Alcance: Negociar con el cliente para eliminar los requisitos "estrictos" que obligan a usar esa tecnología o habilidad específica que nadie tiene.
- Rechazo del proyecto: Si el riesgo es demasiado alto y no hay solución técnica, la decisión más sana comercialmente puede ser no iniciar el proyecto ("No-Go decision").

4. Estrategia de Aceptación (Retención)Ajuste de Cronograma: Asumir que el equipo actual tardará el doble o triple en aprender y hacer el trabajo. Se acepta el riesgo y se extiende la fecha de entrega y el presupuesto.

Resumen:Ante la imposibilidad de contratar, tu plan debe ser: Capacitar a los tuyos (Mitigar), Comprar el software hecho o contratar consultora (Transferir) o Cambiar los requisitos (Evitar).

5. Qué consideraciones se deberían tener en cuenta a la hora de desarrollar una interfaz de usuario.

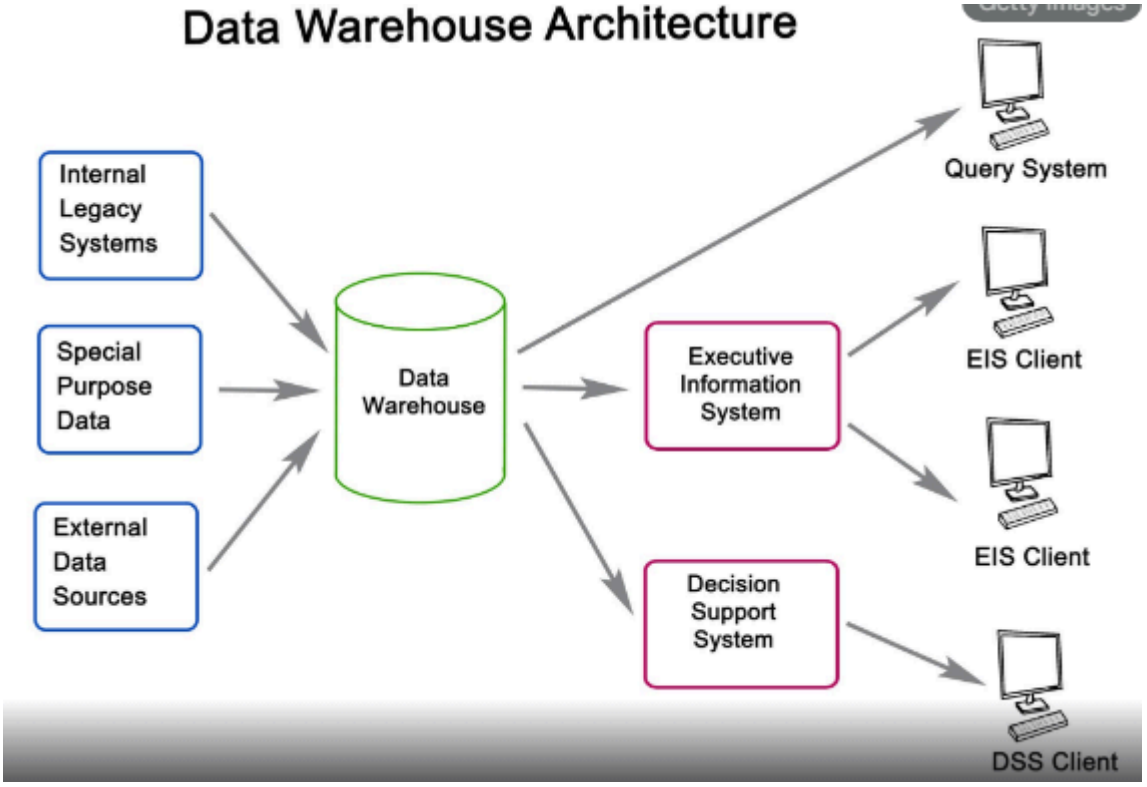
Páginas: 86 y 95.
Respuesta:
- Conocer al usuario (perfil).
- Consistencia (mismos colores y botones).
- Feedback (informar qué pasa).
- Minimizar carga de memoria (no obligar a recordar).
- Prevención de errores.

6. Diferencie Modelo de Repositorio y Modelo Cliente/Servidor.

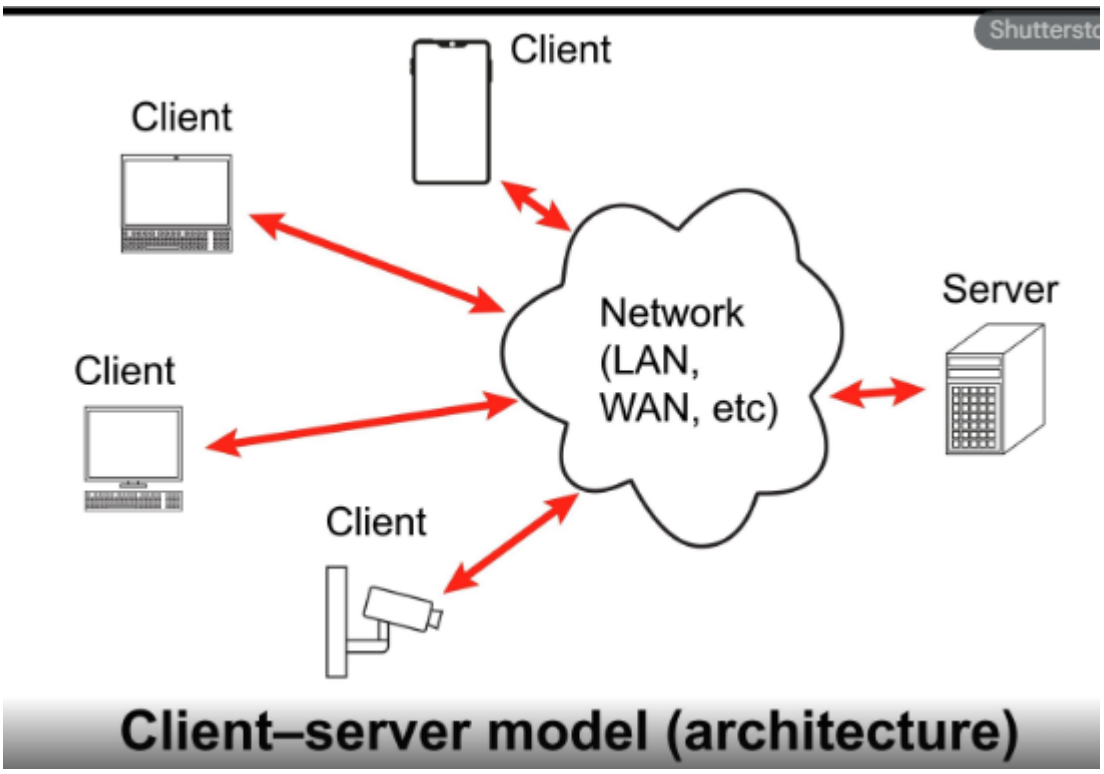
Tipos de organización del sistema
La organización del sistema (o estilos arquitectónicos) determina cómo se distribuyen los datos y el procesamiento. Según el material de la cátedra, existen tres tipos principales:

- Arquitectura de Repositorio (Modelo de Datos Compartido): En este estilo, todos los componentes del sistema acceden a una base de datos central o repositorio. Es ideal cuando se manejan grandes volúmenes de datos que muchos sub-sistemas deben compartir.

- Cómo funciona: Los componentes son independientes; no se comunican entre sí, sino que todos "leen y escriben" en el mismo lugar central.
- Ventaja: Es eficiente para compartir datos.
- Desventaja: El repositorio es un "cuello de botella"; si falla la base de datos central, se cae todo el sistema.



- Modelo de Cliente-Servidor (Servicios Distribuidos): Divide el sistema en dos roles claros: los proveedores de servicios (servidores) y los usuarios de esos servicios (clientes).
- Cómo funciona: El servidor ofrece funcionalidades (ej: base de datos, impresión) y los clientes las solicitan a través de una red.
- Ventaja: Es muy escalable. Podés agregar más clientes sin afectar al servidor, y el procesamiento está distribuido.
- Desventaja: Depende totalmente de la red. Si no hay conexión, el cliente no puede hacer nada.



- **Diseño Arquitectónico:** Es el "mapa" o plano general del sistema.
- **Repositorio:** Todos a un mismo baúl central.
- **Cliente-Servidor:** Uno pide, otro da (distribuido).

La diferencia fundamental está en cómo se comunican los componentes y dónde reside el foco:

- Modelo de Repositorio (Centrado en DATOS):
  - Cómo funciona: Todos los subsistemas comparten una base de datos central. No se hablan entre ellos directamente; se comunican leyendo y escribiendo en ese repositorio compartido.
  - Clave: Es eficiente para compartir grandes volúmenes de datos, pero si cae la base central, muere todo el sistema.
- Modelo Cliente/Servidor (Centrado en PROCESAMIENTO/SERVICIOS):
  - Cómo funciona: El sistema se distribuye en Servidores (que ofrecen servicios) y Clientes (que los piden). La comunicación es directa a través de una red.
  - Clave: Es fácil de distribuir y escalar (agregar más servidores), pero el intercambio de datos por la red es más lento que acceder a una memoria compartida.

Resumen:

- Repositorio = Todos contra la base de datos.
- Cliente/Servidor = Pedidos y respuestas a través de la red.

7. Un diagrama de Pert (Fácil).

1. ¿Qué es la planificación?¿Cuáles son los objetivos de la planificación temporal y organizacional?.

Página: 16-17.
Respuesta: Es el proceso de definir objetivos, recursos y tareas.

- Objetivo Temporal: Controlar el tiempo y cumplir plazos.
- Objetivo Organizacional: Gestionar el equipo humano y roles.

2. ¿Qué es el diseño? Defina acoplamiento y cohesión y sus diferentes grados.

Página: 57, 63, 64.
Respuesta: El diseño es la representación técnica del software antes de construirlo.

- Cohesión: Qué tan relacionadas están las tareas dentro de un módulo. (Se busca Alta). Grados: Funcional (mejor), Secuencial, Comunicacional, Procedimental, Temporal, Lógica, Coincidental (peor).
- Acoplamiento: Qué tanto depende un módulo de otro. (Se busca Bajo).

3. ¿Qué es la GCS?

Páginas: 137 y 139.
Respuesta:

- GCS (Gestión de Configuración del Software): Es el proceso para identificar, controlar y gestionar los cambios en el software a lo largo de su vida.
- Línea base: Es un punto de referencia aprobado formalmente (una "foto" del proyecto en un momento dado).
- Ejemplo: La "Especificación de Requisitos" firmada por el cliente. A partir de esa firma, cualquier cambio requiere un trámite formal.

4. ¿Qué son los riesgos?¿Cómo se clasifican?. Ejemplifique.

Un riesgo se define como un evento no deseado que conlleva consecuencias negativas.
Componentes:

- Incertidumbre (probabilidad): Representa la probabilidad de que ocurra un evento riesgoso. Nunca llega al 100%, ya que ese sería un problema presente.
- Pérdida (impacto): Indica el impacto y grado de consecuencias negativas que el evento riesgoso podría causar. Puede variar en intensidad, desde leve hasta grave.

Se pueden clasificar en tres categorías generales:

4. **Riesgos del proyecto** Amenazan el plan del proyecto (presupuesto, calendario, recursos).
5. **Riesgos técnicos** Amenazan la calidad y la implementación técnica (tecnología nueva, dificultad de diseño).
6. **Riesgos del negocio** Amenazan la viabilidad del producto en el mercado (nadie lo quiere, pierde dinero).

Ubicación: Página 51 (implícito en la descripción general de riesgos).

Ejemplo: Riesgo Técnico -> "La tecnología de base de datos nueva no soporta la cantidad de usuarios esperada".

5. Defina estimación y los tipos de estimación que conoce.

Página: 33.
Respuesta: Es predecir valores futuros (tiempo, costo, esfuerzo) con datos incompletos.
Tipos: Juicio Experto, Basada en Algoritmos (COCOMO), Basada en Analogías.

6. ¿Qué es el mantenimiento? Defina el rejuvenecimiento.
Página: 141 y 144.
Respuesta:

- Mantenimiento: Es la etapa final del ciclo de vida donde se corrigen errores, se adapta o mejora el software después de entregado.
- Rejuvenecimiento: es un desafío dentro del mantenimiento, que busca mejorar la calidad global de un sistema existente. Su objetivo es prevenir la degradación del sistema y evitar fallos relacionados con el envejecimiento, permitiendo aumentar su vida útil y continuar con el mantenimiento del software.

7. Calcular camino crítico (PERT + CMP)

Nº Afirmación V/F

1. Los elementos de la configuración del software son solo los programas y los datos.
2. Una línea base es un concepto de GCS que nos ayuda a controlar los cambios.
3. El análisis del riesgo no implica dejar de lado la gestión de riesgo, ya que podría llevar a obtener una deuda técnica.
4. El concepto de evaluación de un riesgo no implica que siguiendo esta estrategia, la probabilidad de que el riesgo se materialice sea cero.
5. Una tarea en la planificación temporal se describe por 3 elementos: precursor, duración y fecha de entrega.
6. La estrategia de desarrollo en cascada es un proceso secuencial que se hace en etapas.
7. La investigación de usuario es una de las fases del desarrollo de la experiencia de usuario (UX).
8. La mantenibilidad y el rendimiento impactan en el diseño de la arquitectura de software.
9. Las métricas GQM son base de datos que se usan en el ejemplo de diseño de la arquitectura de software en capas.
10. Las métricas estándar se relacionan con las características de calidad del software.
11. No es recomendable usar una métrica que “mida” cierto objetivo.
12. Si se realiza una prueba y el software ha fallado significa que no hace lo que especifican los requerimientos.
13. Un defecto por omisión en el software resulta cuando algún aspecto del código es incorrecto.
14. La prueba de caja negra también denominada prueba de comportamiento, se centra en los requisitos funcionales del software analizando las entradas y salidas.

15. La complejidad del software es una métrica que proporciona una medición cuantitativa del comportamiento de la calidad del software.
16. Las técnicas de estrategias de pruebas son parte de la Verificación y Validación incluidas en el aseguramiento de la calidad del software.
17. Las pruebas de regresión de software solo se pueden hacer en forma manual.