# PREGUNTAS TIPO PARCIAL

**¿Qué es el software y cuál es su importancia?**

**¿Quién lo Hace?**

**¿Como y que se obtiene?**

**¿Para qué se aplica la ingeniería de software? Describa sus etapas**

**¿Porque es costoso mantener un software?**

**¿Qué es un proceso de software?**

**¿Cuáles son los flujos de proceso? Descríbalos**

**¿Que son los patrones de proceso?**

**¿Qué es el CMM? que niveles tiene?**

**¿Qué es la ingeniería de requerimientos?**

**¿Qué es un requerimiento? ¿Cuáles son sus tipos?**

**Mencione dos maneras de elicitacion**

**¿En dónde se encuentras los problemas de requerimientos?**

# Procesos para la creación de software

## - Software como producto | Ingeniería de software

**Software:**

El software es el producto del trabajo de los programadores.

Es necesario mantenerlo luego de haberlo largado.

* Obsolescencia: Si los requerimientos cambian, lo que el software hace queda obsoleto, por lo que hay que mantenerlo (Etapa de mantenimiento)
* Etapa de mantenimiento: Intenta extender el tiempo de vida de del software.
* Ser productivo con un software debemos compararnos con lo que ganaríamos en una empresa.

**¿Quién lo hace?:**

* Ingenieros en software
* Arquitectos de software
* Diseñadores
* Analistas
* Programadores

**Importancia:**

* Toda empresa de hoy en día utiliza un mínimo de tecnología para sobrevivir, es su producto organizacional

**Como obtenerlo:**

Está definido por las metodologías

* Tradicionales, Iterativos y evolutivos, agiles
* Se utiliza una diferente en diferentes procesos del trabajo

**Se obtiene:**

* Para un programador:
  + Programas y datos
  + Se constituye un sistema
* Para el usuario:
  + La información
  + El valor agregado a los procesos y negocios de la organización.

**Software como sistema**

El programador no analiza el sistema, el analista es el que lo hace, y le pide al programador las cosas específicas que necesita en el sistema.

**Ingeniería de software:**

Se aplica para poder realizar bien el sistema, son décadas de experiencia, de las cuales se concluye como hacer las cosas bien.

- Una herramienta para obtener ese producto con unos estándares de calidad

Es un enfoque sistemático, disciplinado y cuantificado al desarrollo, operación y mantenimiento de software

Gráfico de embudo

Descripción generada automáticamenteProceso: define el marco de trabajo para un conjunto de áreas claves de proceso (ACPs).

Métodos: indican “como” construir técnicamente el software (Ej.: análisis de requisitos, diseño, construcción, pruebas, mantenimiento).

Herramientas: proporcionan un enfoque automático o semiautomático para el proceso y los métodos.

El **proceso** define fases:

* Definición: Se centra en el “Qué”. Permite identificar los requisitos
* Desarrollo: Se centra en el “Como”. Define las estructuras de datos, arquitectura y procedimientos.
* Mantenimiento: Se centra en el “Cambio”. Corrección de errores y adaptaciones necesarias.

BUENO + BARATO: LENTO

BUENO + RAPIDO: CARO

RAPIDO + BARATO: MALO

**El costo de mantenimiento siempre va creciendo.**

El costo de mantenimiento del producto no puede sobrepasar el costo de crear el producto desde cero.

* Como cualquier otro producto, el software sufre, no de un desgaste físico, sino que se vuelve obsoleto

**Software heredado:**

Comúnmente cuando lleguemos a una empresa, no vamos a empezar desde cero, vamos a tener 200 programas ya creados a los que nos vamos a tener que adaptar.

**Wasaps:**

Las aplicaciones webs, necesita uso intensivo de las redes de concurrencia a carga impredecible (pasa de 1000 ventas a 100000), rendimiento (tarda 5 segs en cargar, me voy), disponibilidad, es orientado a datos (Navegadores trabajan con esto), naturaleza sensible, evolución continua, inmediatez, seguridad, estética.

## - Software como proceso

**¿Qué es un proceso de software?**

El proceso de software define las actividades, acciones y tareas que se requieren para construir un software de alta calidad

**Flujos de proceso:**

* Flujo de proceso lineal: Ejecuta las cinco actividades en secuencia

Diagrama

Descripción generada automáticamente

* Flujo de proceso iterativo: Repite una o mas veces las actividades antes de pasar a la siguiente

Diagrama

Descripción generada automáticamente

* Flujo de proceso evolutivo: Funciona como un ciclo, vuelve a repetirse, y cada vez se completa más

Diagrama

Descripción generada automáticamente

* Flujo de proceso paralelo: El modelado de un aspecto del software se puede estar ejecutando a la vez que la construcción de otro aspecto del software

Diagrama

Descripción generada automáticamente

**Patrones de proceso:**

Un patrón de proceso describe un problema en la etapa de ingeniería de software y sugiere una o más soluciones.

Se compone de:

* Nombre – Fuerzas – Tipo – Contexto inicial – Problema – Solución – Contexto Resultante – Patrones relacionados – Usos conocidos

**Evaluación y mejora del proceso:**

CMM – Modelo de madurez de la capacidad

1. Inicial: El proceso de software es caótico y tiene pocos procesos definidos.

2. Repetible: Se usan procesos administrativos básicos para el seguimiento de costos y tiempos.

3. Definido: El proceso de software está documentado, estandarizado e integrado en un proceso para la organización.

4. Administrado: Se decide la calidad del producto. Tanto el proceso como el producto de software se controlan.

5. Optimizado: Existe una retroalimentación que permite una mejora continua del proceso.

Tareas principales:

* Comunicación
* Planeamiento
* Moderado
* Construcción
* Despliegue

Entregable: Producto con la idea de.

## - Requerimientos e ingeniería de R

**Ingeniería requerimientos:**

* Es imposible que alguien sepa hacer todo, es necesario contratar gente que sepa del tema, si tengo que hacer un programa de contabilidad, y no se contabilidad, llamo a alguien que sepa del tema, sino no puedo hacer ni los requerimientos.

La ingeniería de requisitos separa el “Que” del “Como”, se utiliza para saber que es lo que el cliente necesita, emplea diferentes métodos para conseguir los diferentes requisitos. Es la parte más importante del proceso de desarrollo de software, si no se hace una buena ingeniería de requisitos, el resto nos servirá de nada.

Se dedica mucho tiempo para el análisis del problema, tiene que estar todo bien definido desde el principio.

Debe tener una evolución, ir creciendo en complejidad, tiene que ser realista en los pasos de los requerimientos.

Primero se puede hacer un prototipo, para poder estar de acuerdo en todos los puntos.

Cambiar algo cuando el sistema ya se encuentra en etapas finales, es sumamente costoso, por lo que hay que tener bien en claro todo desde el principio.

**Requerimiento:**

* Situación o capacidad que se necesita para alcanzar un objetivo
* Pueden ser: Necesidades, deseos o expectativas.
  + Necesidades: Características necesarias para resolver el problema
  + Deseos: Capacidad del sistema que no son imprescindibles, pero si están mejor.
  + Expectativas: Características del sistema esperadas, capaz no explicitas, que se pueden perder fácilmente, como aspectos de la interfaz o de funcionalidad.
* Se deben tener bien en claro estas diferencias, para no confundir como necesario algo que era una boludez.

**Tipos de requerimientos:**

* Funcionales: Que necesita la empresa para cumplir el requisito empresarial.
  + No todo requerimiento es un caso de uso.
* No funcionales: Restricciones al sistema, requerimientos técnicos.
* Empresariales: Razón por la que creo el sistema, lo que la empresa necesita. restricciones del ambiente.
* Requerimientos de la empresa: Propósitos de la empresa (Quiere ser más eficiente), que tenga ciertos protocolos.
* Requerimientos no funcionales: Que el sistema sea “Rápido”, que sea “Fiable”, Que no se detenga, etc.
* Requerimientos funcionales: Que es lo que el sistema hace. Que procesos son los que genera la información que el usuario necesita.

**Entrevistas:**

* **Objetivos de las entrevistas:**
* 1. Registrar información fácilmente utilizable en los procesos de la RE
* 2. Descubrir información proveniente del entrevistado precisa y eficientemente
* 3. Asegurar al entrevistado que su comprensión del tópico ha sido explorado, atendido y valorado
* 4. La entrevista se planifica, que preguntas hacer, cuanto tiempo va a llevar
* Procesos de entrevistas:

Derivación de escenarios:

Se utiliza el LEL como técnica, interpretar escenarios (Palabras) para una mejor interpretación. (Sujeto, objeto, acción)

**BrainStorming:**

Introducido en 1930 para generar ideas

Todos los miembros del grupo contribuyen a una lista de problemas a resolver o soluciones a un problema

Ayuda a discutir muchas ideas en un corto período de tiempo

**Procesos de RE**

* Elisitación de requerimientos
* Especificación de requerimientos
* Validación de requerimientos

**Donde están los problemas:**

* Falta de input del usuario
* Requerimientos incompletos
* Cambio de los requerimientos
* Expectativas poco realistas
* Falta de claridad

Las partes interesadas muchas veces no conocen lo que desean obtener, o se expresan con sus propios términos, o distintas partes tienen requerimientos diferentes

# Modelo de seguridad

La arquitectura nos permite construir nuestro sistema de manera sólida y segura.

Al construir un sistema **multiusuario**, aparece un **manejo concurrente de datos**, tenemos que garantizar que un usuario no modifique un dato más actualizado, y mantener una **integridad y consistencia de los datos.**

**Concurrencia:**

Propiedad de los sistemas en donde los procesos de computo se hacen simultáneamente

**Arquitectura:**

La arquitectura de un software es su simiento, permite que el software crezca y se mantenga.

Hay muchos tipos de arquitecturas y patrones para ayudarnos en esto, hay que elegir la opción adecuada.

## - LogIn, Logout

Se utiliza el patron singleton para que la sesión de un usuario sea unica

## - Gestión de perfiles

## - Bitácora

Registro de que actividades de acciones se llevaron adelante en el sistema

* Se guardan los datos asociados. Un string con los datos del cliente, etc.
* Usuario, rango de fecha, tipo de acción.
* Usuario auditor

## - Encriptación

* Se hace antes de entrar al sistema, si valida abre el sistema.
* Dato extra para verificar la veracidad de los datos. Dato de redundancia
* Se obtiene con un cálculo entre los datos guardados.
* Verificador horizontal para verificar los datos de un row
* Verificador vertical para asegurar que no ingresan o borran un row de datos

## - Multi-idioma

Las interfaces se pueden ver en distintos idiomas

* Idioma base, cargar idiomas diferentes,
* En la tabla SQL seria:

|  |  |  |  |
| --- | --- | --- | --- |
| 01 | Botón | Button | Buttoni |

* Cuando inicio sesión, elijo el idioma, voy a la tabla SQL para saber que leyenda se debe mostrar.

## - Backup

Backup:

Poder hacer una copia de la base de datos, del mdvs y del log

# Patrones de diseño

## - Patrones y desarrollo de software

**¿Qué es un patrón?**

Cada patrón describe un problema que ocurre seguido en nuestro entorno, y también describe su solución. Por lo que se puede aplicar las veces que quiera.

Son soluciones conocidas y probadas. Son descripciones de clases y las relaciones de objetos.

**Elementos:**

Nombre, problema, solución, consecuencias, contexto.

**Patrones creacionales**

**Centrados en la creación de objetos**

Dan flexibilidad a que es lo que se crea, quien lo crea y cuando

**Patrones de comportamiento**

**Están enfocados en la interacción dinámica**

Los patrones de comportamiento están ligados con algoritmos y con la asignación de responsabilidades a objetos, describen la comunicación entre estos (los objetos).

Podemos identificar a estos en dos grupos:

* Los basados en clases que hacen uso de la herencia para distribuir el comportamiento.
* Los basados en objetos que hace uso de la composición de objetos en vez de la herencia.

**Patrones estructurales**

**Enfocados en la composición estática y en las estructuras de clases y objetos**

Los patrones estructurales se ocupan de combinar clases y objetos para formar estructuras más grandes. Al igual que con los de comportamiento, también podemos identificar a estos en dos grupos:

* Los basados en clases hacen uso de la herencia para componer interfaces o implementaciones.
* Los basados en objetos describen formas de componer objetos para obtener una nueva funcionalidad.

## - Singleton

El propósito es que una clase solo tenga una instancia, y proporciona un punto de acceso global a ella

\*Una y solo una\*

Tabla

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Necesito un constructor privado, para que nadie lo llame.

Necesito un objeto de la clase, también privado, para crearlo.

Tengo el método getInstance, público, que solo cuando el objeto anterior no exista, lo va a crear.

## - Composite

**Patrón Composite**

El patrón composite se encuentra situado dentro de los patrones estructurales. Su finalidad es poder construir objetos complejos en base a objetos más simples. Esto es posible gracias a la recursividad y a la estructura en forma de árbol.

Permite que los clientes traten de manera uniforme los objetos individuales y a los compuestos y a los objetos en estructuras de árbol para representar jerarquías de parte-todo.

Imagen en blanco y negro de un reloj

Descripción generada automáticamente con confianza baja

Aplicabilidad:

* Cuando se quiera representar jerarquías de objetos todo-parte.
* Cuando se quiere que los clientes sean capaces de obviar las diferencias entre composiciones de objetos y los objetos individuales. Estos tratarán a todos los objetos de la estructura de manera uniforme.

Estructura:

Diagrama

Descripción generada automáticamente

Consecuencias:

* Define jerarquías de clases formadas por objetos primitivos y compuestos.
* El cliente trata las estructuras uniformemente.
* Facilita añadir nuevos tipos de componentes.
* Puede hacer que un diseño sea demasiado general.

Ejemplo:

Imaginemos que nos solicitan desarrollar un sistema para poder gestionar los diferentes menús en un restaurante, ya que dependiendo el momento del día el mismo cambia completamente.

Tengo una clase “General”, en donde declaro (abstracta) los métodos necesarios para las clases que la heredan, no importa que no los usen.

La clase general tiene una lista de los que la integran (Sus nodos y sus hojas) y un método para agregar un hijo (Nodo y hoja)

Después tenemos la clase nodo, y la clase hoja, que heredan de la clase general. Por lo que son el mismo componente

Diagrama

Descripción generada automáticamente

Podemos observar un menú principal que va a contener los diferentes menús que van a ir variando a lo largo del día.

Podríamos plantear la siguiente solución:

Diagrama

Descripción generada automáticamente

## - Observer

**Patrón Observer**

El patrón observer define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y se actualicen automáticamente todos los objetos que dependan de él. A dicho patrón también lo podemos encontrar como Dependents o Publish-Suscribe.

Aplicabilidad:

* Cuando una abstracción tiene dos aspectos y uno depende del otro.
* Cuando un cambio de un objeto requiere cambiar otros, y no sabemos cuántos objetos necesitan cambiarse.
* Cuando un objeto debería ser capaz de notificar a otros sin hacer suposiciones sobre quienes son dichos objetos.

Estructura:

Diagrama

Descripción generada automáticamente

Consecuencias:

* El patrón permite modificar los sujetos y observadores de forma independiente. Es posible reutilizar objetos sin reutilizar sus observadores, y viceversa. Esto permite añadir observadores sin modificar el sujeto u otros observadores.
* Acoplamiento abstracto entre sujeto y observador.
* Capacidad de comunicación mediante difusión.
* Actualizaciones inesperadas.

Ejemplo:

Imaginemos que se nos solicita desarrollar un sistema el cual permita crear publicaciones al usuario y que permita “suscribir” a otros a las mismas. El sistema deberá dar aviso a los usuarios suscritos cada vez que haya una nueva publicación.

Dicho patrón se verá reflejado al momento de desarrollar el módulo de idioma en el sistema desarrollado en la materia.

Se tiene un sujeto y un observador, el observado (el sujeto) mantiene una lista de todos los observadores que dependen de el.

El sujeto tiene 3 operadores principales: Suscribir, de-suscribir y notificar. Ante un cambio de estado, va a mandar un mensaje de notificaciones, que recorrerá toda la lista de observadores y le enviará mensaje para que se actualice.

Se tiene un observador concreto que puede obtener de la notificación el objeto o mensaje.