CLASE 1: REPASO

¿Qué es el software?
En líneas generales, es conocimiento, del usuario, enlatado. El software deberá resolver una necesidad que no es ni más ni menos que un **requerimiento del usuario,** que será transformado en software a partir de convertir todas sus ideas y conocimiento en un modelo conceptual, que nos permita reducir la ambigüedad de la comunicación, y luego en líneas de código.

Procesos de negocio
Un **proceso de negocio** tiene personas que cumplen roles y realizan actividades en búsqueda de lograr un objetivo y generar valor al negocio en sí. Es importante comprender el proceso de negocio para luego elegir que actividades serán las que informaticemos con nuestro sistema. Durante el modelado del negocio:

- Se examina la estructura de la organización
- Se observan roles y las relaciones entre ellos,
- Se analiza el flujo de trabajo y los procesos principales.
- Se estudian entidades externas

Y se observan los siguientes elementos:

- Flujo de tareas
- Agentes
- Información
- Reglas de Negocio
    - Regulan el funcionamiento de la empresa
    - Describen restricciones e influyen en los RF

Una vez identificadas todas las actividades que serán parte del alcance de nuestro sistema**, c**omenzamos con el proceso de desarrollo. Pensando en el P.U, comenzamos con el análisis de los requisitos, luego el diseño, implementación y pruebas transformando esto en un proceso iterativo e incremental
Durante el análisis utilizamos CU para describir las funcionalidades que tendrá el sistema. Obtenemos un modelo de dominio estructural de la aplicación y un diagrama que nos muestra la interacción del actor con el sistema

CLASE 2: PROCESO DE DESARROLLO DE SOFTWARE

Software como producto de ingeniería:

- **¿Qué es?**
    - Es el producto del trabajo de los programadores y como producto, luego de su puesta en marcha necesita que se lo mantenga. Se sustenta en otros productos generados por ing. de sist. – arq. de sw - diseñadores
- **¿Quién lo hace?**
    - Ingenieros de Software / Ingenieros en Sistemas
    - Arquitectos de Software
    - Diseñadores
    - Analistas
- **¿Por qué es importante?**
    - Porque es un producto transversal a casi todas las actividades cotidianas tanto en el ámbito personal como organizacional
    - Porque la IS permite que cuando estos productos son complejos se puedan desarrollar con calidad
- **¿Cuáles son los pasos para obtenerlo?**
    - Los pasos estarán definidos por la metodología que se elija para llevar adelante el proyecto, esta metodología puede ser: tradicional, iterativa y evolutiva, ágil
- **¿Qué se obtiene como producto final?**
    - Lo que se obtiene va a depender del punto de vista. Desde el programador, a partir de programas y datos, una vez organizados constituyen “el sistema” y desde el usuario, se obtiene un valor agregado a los procesos y negocios de la organización

El software no se desgasta, pero se vuelve obsoleto. Cuando nace, comienza con una tasa de fallas que será necesaria mejorar para evitar la "mortalidad infantil". Con el paso del tiempo, a medida que el software comienza a sufrir cambios, las fallas comienzan a emerger hasta que se vuelven inmanejables, o el costo de mantenimiento es alto en comparación a adquirir o crear un software. Esto se conoce como entropía del software y nos revela la importancia de construir un software de calidad y reutilizable

¿Qué es un proceso del software?

Es una estructura para las actividades, acciones y tareas que se requieren para construir software de alta calidad. El flujo lineal del proceso de desarrollo de software está constituido por cinco actividades estructurales que se ejecutan en secuencia: **Comunicación -> Planeación -> Modelado -> Construcción -> Despliegue**

Consideraciones que debemos tener en cuenta antes de definir el proceso:

- Se debe hacer un esfuerzo por comprender el problema antes de desarrollar
- El diseño es una actividad crucial
- La calidad del software impacta en el producto o servicio que se brinda
- El software actual debe ser fácil de mantener

Ingeniería de Software

Es el establecimiento y uso de principios de la ingeniería para obtener un software que sea fiable y que funcione eficientemente sobre máquinas reales

Según Pressman: es un conjunto de proceso, métodos y herramientas apoyados en el compromiso de la calidad

- **Proceso:** define el marco de trabajo para un conjunto de áreas claves de proceso
  - Definición: se centra en el "Qué". Permite identificar los requisitos claves del sistema y del software.
  - Desarrollo: se centra en el "Cómo". Definición de estructuras de datos, arquitectura y detalles procedimentales. Se debe elegir un lenguaje.
  - Mantenimiento: se centra en el "Cambio". Asociado a la corrección de errores y adaptaciones requeridas
- **Métodos:** indican "como" construir técnicamente el software.
- **Herramientas:** proporcionan un enfoque automático o semiautomático para el proceso y los métodos

Modelos de proceso

El modelado es una técnica para tratar con la complejidad inherente a los sistemas. Los modelos se pueden utilizar para validar los requerimientos o funcionalidad esperada del sistema. El más utilizado es el **modelo prescriptivo**, que propone un conjunto de actividades, fundamentos y productos de trabajo que se requieren para desarrollar software de alta calidad. Las actividades que considera este modelo son: **Comunicación – Planeación – Modelado – Construcción – Desarrollo**. Y las organiza en base a un flujo de proceso que puede ser: **Lineal (Cascada, Desarrollo Rápido de Apps.) – Incremental – Evolutivo (Prototipos, D. espiral, D. concurrente)**

Modelo en cascada

Propone un conjunto de actividades secuenciales que deben terminar para dar comienzo a la siguiente:

- **Comunicación**: inicio del proyecto – recopilación de requisitos / **Planeación**: estimación – itinerario – seguimiento
- **Modelado**: Análisis y diseño / **Construcción**: código y prueba / **Despliegue**: entrega – soporte – retroalimentación

Problemas:

- No es común que los proyectos sigan un flujo secuencial
- Es difícil que el Cliente establezca todos los requerimientos de manera explícita
- Para obtener el software se debe esperar a que el proyecto este avanzado

Modelo basado en prototipos

Presenta una mejora en cuanto al problema, de establecer todos los requerimientos desde el principio, que tiene el modelo en cascada. Tiene como características y actividades:

- Comienza con la recolección de requisitos, el desarrollador y el cliente definen los objetivos globales e identifican requisitos conocidos y áreas que requieren más definición.
- De esto se crea un modelado y diseño rápido, que es construido como un prototipo.
- El prototipo se evalúa y se utiliza para refinar los requerimientos
- Un problema de este modelo es que, se suele confundir el prototipo con una versión final

Modelo incremental

Combina elementos del modelo lineal secuencial (cascada) con la filosofía iterativa de construcción de prototipos. Aplica secuencias lineales de forma escalonada y entrega un producto operacional al final de cada incremento

- El primer incremento es un producto esencial, se incorporan los requisitos básicos
- El producto esencial es evaluado por el cliente
- Los primeros incrementos son versiones incompletas del producto final

Proceso Unificado

Es un marco de trabajo del proceso para la IS OO, mediante la utilización de UML. Es un proceso de flujo iterativo e incremental, tomando lo mejor de cada uno de los modelos tradicionales. Todas las actividades (Recopilación de req. – AyD – Implementación y Test) aparecen en todas sus fases: **Gestación – Elaboración – Construcción – Transición**

- Dirigido por CU
- Iterativo e Incremental
- Centrado en la Arquitectura de Software

CLASE 3: ANÁLISIS Y DISEÑO DE UN MÓDULO DE SEGURIDAD Y SERVICIO

Arquitectura

La arquitectura de un software es su cimiento. Es lo que permitirá que ese software pueda crecer y mantenerse de la mejor manera. Al momento de elegir una arquitectura debemos considerar algunos criterios importantes. Entre ellos:

- **Mantenibilidad**: ¿Qué difícil es agregar o quitar nuevas funcionalidades?
- **Reúso**: Reutilizar componentes individuales en otros sistemas
- **Performance**: ¿Hay mucho tiempo de respuesta? ¿El comportamiento de los recursos es aceptable?
- **Claridad:** ¿Es posible darle feed back a los usuarios?
- **Tolerancia a fallos**

Login y Logout

Al iniciar sesión en un sistema, se debe crear una única sesión para el usuario hasta que cierre el sistema, no puede tener múltiples sesiones. Este problema lo resuelve el patrón Singleton, que se adapta para lograr que la sesión de un usuario sea única durante toda la vida de la misma, así también para que pueda ser accedida desde cualquier módulo del sistema

Gestión de perfiles

En la gestión de perfiles y permisos de usuario, se emplea el concepto de usuarios, patentes y familias. El concepto de **familia** es análogo al concepto de **grupo de permisos** (es un organizador lógico de patentes). Una **patente** es el permiso "atómico" que el sistema buscará validar para brindar acceso o no a un módulo del sistema. Un **usuario** tendrá asociados un conjunto de familias y/o patentes (este será su perfil). Para lograr esto se implementará el patrón Composite, que permite crear estructuras de objetos complejas a partir de una estructura clases basada en la herencia. Podremos crear objetos que representen hojas de un árbol (patentes) y ramas (familias), generando una estructura tipo árbol

Módulo de Encriptación

El concepto de encriptación representa transformar una cadena de caracteres en algo que el ser humano no pueda entender, basándose algún tipo de algoritmo. En base a esto, podemos definir los siguientes conceptos sobre criptografía:

- **Integridad**: El mensaje se lee tal cual se envió
- **Autenticidad**: mensaje enviado por quien dice que lo envió
- **Confidencialidad**: El mensaje solo lo lee quien es el destinatario

Tipos de encriptación

Criptografía simétrica

Solo utiliza una clave para cifrar y descifrar el mensaje, que tiene que conocer el emisor y el receptor. Este es el punto débil del sistema, la comunicación de las claves entre ambos sujetos, ya que resulta fácil interceptar una clave que se ha transmitido sin seguridad. Para utilizar este sistema de encriptación se deben considerar los siguientes temas:

- La clave es única (puede generar problemas de seguridad)
- Se necesita un canal seguro externo al sistema para compartir dicha clave

Criptografía asimétrica

Se basa en el uso de dos claves: la pública (que se podrá difundir a todas las personas que necesiten mandarte algo cifrado) y la privada (que no debe de ser revelada nunca). El método simétrico es inseguro y el asimétrico es lento

Criptografía por HASH

Son algoritmos que consiguen crear, a partir de una entrada (ya sea un texto, contraseña o archivo), una salida alfanumérica de longitud fija que representa un resumen de toda la información que se le ha dado (es decir, a partir de los datos de la entrada crea una cadena que solo puede volverse a crear con esos mismos datos). Permiten asegurar que no se ha modificado el mensaje en una transmisión, hacer ilegible una contraseña, etc.

Gestión de múltiples idiomas

La gestión de múltiples idiomas en un sistema implica adaptar la interfaz de usuario y su contenido para que sea accesible y comprensible para usuarios que manejen distintos idiomas. Aquí es donde el patrón Observer es útil. Este patrón establece una relación de dependencia entre objetos, de modo que cuando un objeto cambia de estado, todos los objetos dependientes son notificados y actualizados. Cuando un usuario cambia el idioma se notifica a todos los observadores dependientes sobre este cambio. A continuación, los observadores actualizan su contenido textual para reflejar el nuevo idioma seleccionado

CLASE 4: INTRODUCCION A LOS PATRONES DE DISEÑO

Principios SOLID

Single Responsability – Open/Closed – Liskov Substitution – Interface Segregation – Dependency Inversión

Introducción al concepto de patrones de diseño

¿Qué es un patrón?

Describe un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema, de tal modo que se pueda aplicar esta solución un millón de veces. Estos pueden ser adaptados y combinados de diferentes maneras

Nuestro diseño debe ser específico del problema que estamos manejando, pero también lo suficientemente general para adaptarse a futuros requisitos y problemas. Lograr que el diseño sea flexible y reutilizable suele ser difícil

Patrones de diseño en IS

Patrón de diseño: representación esquemática de una solución a un problema recurrente

Favorecen el reúso y la confianza en el software usando soluciones conocidas y probadas. Son descripciones de clases y objetos relacionados, particularizados para resolver un problema de diseño general. Están divididos en 3 categorías:

- **Patrones de creación:** centrados en la creación de los objetos
- **Patrones de estructura:** enfocados en la composición estática y en las estructuras de clases y objetos
- **Patrones de comportamiento:** enfocados en la interacción dinámica

Elementos esenciales de un patrón

- **Nombre:** describe el problema junto con sus soluciones y consecuencias
- **Problema:** describe cuándo usar el patrón.
- **Solución:** describe de qué forma se soluciona el problema planteado
- **Consecuencias:** ventajas y desventajas de aplicar el patrón
- **Contexto**: donde será posible aplicar el patrón

¿De qué forma resuelven los patrones los problemas de diseño?

- **Mediante el diseño de componentes genéricos**
  - Clases o paquetes que pueden ser extendidos, adaptados y reusados sin tener que modificar el código, para esto se utilizan dos técnicas: refactorizar y generalizar. Las clases abstractas e interfaces son de mucha ayuda
- **Cuando ponen a funcionar los mecanismos de reutilización.**
- **Al usar estructuras que relacionan tiempo de ejecución y compilación.**
- **Diseñar para el cambio**
  - Para diseñar un sistema que sea robusto a los nuevos requisitos hay que tener en cuenta cómo puede cambiar el sistema a lo largo de la vida. El rediseño afecta a muchas partes del sistema, por lo que los cambios no previstos siempre resultan costosos

Anti-patrones

Un patrón se convierte en un anti-patrón cuando su utilización conduce a una mala solución. Hay anti patrones de: gestión – diseño de sw – diseño OO – programación – proyectos

Patrones creacionales

Los patrones de creación de clases hacen uso de la herencia para cambiar la clase de la instancia a crear

- Abstraen el proceso de creación de instancias.
- Encapsulan el conocimiento sobre las clases concretas que usa el sistema.
- Ocultan cómo se crean y se asocian las instancias de estas clases.
- Dan mucha flexibilidad a **qué** es lo que se crea, **quién** lo crea y **cuándo**

Patrón Singleton

Este patrón tiene como propósito mantener una instancia única de un objeto, accesible desde toda la aplicación. Se aplica en un escenario donde debe haber una instancia de una clase, y esta debe ser accesible a los clientes desde un punto de acceso conocido. Dicha instancia debería ser extensible mediante herencia, y los clientes deberían ser capaces de usar una instancia extendida sin modificar su código.

Consecuencias de usar el patrón Singleton:

- Acceso controlado a la única instancia
- Espacio de nombres reducido
- Permite el refinamiento de operaciones - Permite un número variable de instancias.
- Más flexible que las operaciones de clase

Patrón Factory Method

Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clases instanciar. Permite a una clase delegar en sus subclases la creación de objetos. Su aplicabilidad es cuando una clase no puede prever la clase de objetos que debe crear.

Las clases delegan las responsabilidades en una de entre varias clases auxiliares, y queremos localizar qué subclase auxiliar concreta es la que se delega.

Consecuencias de usar el patrón Factory Method:

- Proporciona enganches para las subclases
- Conecta jerarquías de clases paralelas

Patrón Abstract Factory

Su propósito es solucionar la creación de diferentes familias de objetos. Está aconsejado cuando se prevé la inclusión de nuevas familias de productos, pero puede resultar contraproducente cuando se añaden nuevos productos o cambian los existentes ya que afectaría a todas las familias creadas. Proporciona una interfaz para crear familias de objetos relacionado o que dependen entre sí, sin especificar sus clases concretas. Debemos crear diferentes objetos, todos pertenecientes a la misma familia

Su aplicabilidad se ve cuando:

- Un sistema debe ser independiente de cómo se crean, componen y representan sus productos
- Un sistema debe ser configurado como una familia de entre varias
- Una familia de objetos relacionados está diseñada para ser usada conjuntamente (restricción)
- Proporciona una biblioteca de productos, relevando solo sus interfaces y no sus implementaciones

Consecuencias de usar el patrón Abstract Factory:

- Aísla las clases concretas
- Facilita el intercambio de familias de productos
- Promueve la consistencia entre productos
- Es difícil agregar nuevos tipos de productos

CLASE 5: PATRONES DE DISEÑO #2

Patrones de comportamiento

Están ligados con algoritmos y con la asignación de responsabilidades a objetos, describen la comunicación entre estos. Estos patrones están divididos en dos grupos:

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

Patrones estructurales

Se ocupan de combinar clases y objetos para formar estructuras más grandes y también están divididos 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

Patrón Composite

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 a los objetos individuales y a los compuestos. Compone objetos en estructuras de árbol para representar jerarquías de parte-todo

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 compuesta de manera uniforme.

Consecuencias de usar el patrón Composite:

- 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

Patrón Observer

Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen todos los objetos que dependan de él

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

Consecuencias de usar el patrón Observer:

- 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

CLASE 9: METRICAS

Medición: Proceso por el cual números o símbolos son asignados a atributos de entidades para describirlos de acuerdo con reglas establecidas. La elección de una medición para un atributo exige un grado de conocimiento del atributo

Las medidas de un atributo nos permiten mejorar nuestro conocimiento de la entidad a la que pertenece el atributo. Caracterizar o calificar con un atributo no medible a una entidad es una decisión arbitraria y muestra conocimiento incompleto. Podemos clasificar a la medición en:

- **Medición directa de un atributo:** no exige mediciones de otros atributos
- **Medición indirecta:** exige las mediciones previas de otros atributos y la especificación de ecuaciones que los relacionen

Razones para medir

- Caracterizamos para comprender mejor los procesos, productos, recursos, y entornos, y para establecer una base para las comparaciones con evaluaciones futuras
- Evaluamos para determinar el estado con respecto al diseño. Permite conocer cuando nuestros proyectos y procesos están perdiendo la pista
- Predecimos
    - Para poder planificar: Los valores observados pueden ser utilizados para predecir otros
    - Para establecer objetivos, proyecciones y estimaciones basadas en datos
    - Para analizar riesgos
- Medimos para mejorar la recolección de información cuantitativa que ayuda a identificar obstáculos

Medida, métrica e indicadores

- Una **medida** proporciona una indicación cuantitativa de la extensión, cantidad, capacidad o tamaño de algunos atributos de un proceso o producto
- La **medición** es el acto de determinar una medida. Aparece como resultado de la recopilación de uno o más aspectos de los datos. Este proceso posee las siguientes etapas:
    - **Formulación**: Es la obtención de medidas y métricas apropiadas para la representación del software que se desea desarrollar
    - **Colección**: Calculo de las métricas
    - **Interpretación**: Evaluación de los resultados
    - **Realimentación**: Recomendaciones obtenidas a través de la interpretación de las métricas obtenidas por el equipo de desarrollo
- **Métrica:** es una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado
- Un **indicador** es una métrica o combinación de ellas que brinda una visión profunda del proceso o proyecto o producto para realizar los ajustes necesarios en ellos para mejorar

Un ingeniero de software recopila medidas y desarrolla métricas para obtener indicadores

Indicadores y procesos – Métricas de proyecto

Se deberían recopilar métricas para que los indicadores del proceso y del producto puedan ser ciertos. Los indicadores de proyecto permiten al gestor de proyectos del software:

- Evaluar el estado del proyecto en curso
- Seguir la pista de los riesgos potenciales
- Detectar las áreas de problemas antes de que se conviertan en críticas
- Ajustar el flujo y las tareas del trabajo
- Evaluar la habilidad de un equipo del proyecto para controlar la calidad de los productos de trabajo de software

Integración al proceso de desarrollo de software

El proceso se sitúa en el centro de un triángulo que conecta tres factores con una profunda influencia en la calidad del software y en el rendimiento como organización