# UNIDAD I

## MAPAS CONCEPTUALES

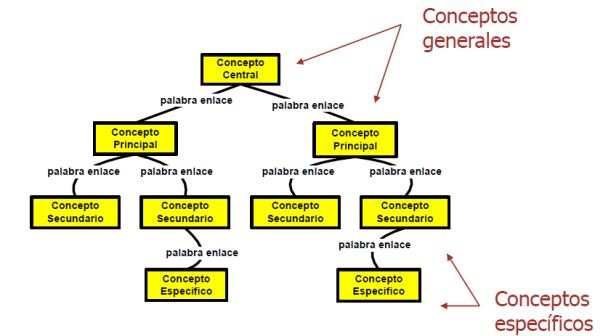
Un mapa conceptual es un gráfico compuesto por términos conceptuales relacionados por palabras enlace para formar proposiciones en una estructura jerárquica. Es una manera muy concreta de poder comenzar a desarrollar software.

Algunos objetivos:

* Desarrollar la capacidad reflexiva de los alumnos
* Aprender a aprender

### Jerarquía

La estructura cognitiva del ser humano está organizada jerárquicamente. Características en las que se apoya el aprendizaje memorístico. Esta organización se basa en proposiciones y conceptos menos generales y más específicos subordinados a las proposiciones y conceptos más generales e inclusivos.



### Mapas conceptuales

Los mapas conceptuales se forman de conceptos, palabras enlace y proposiciones.

#### Concepto

* Regularidad en los acontecimientos o en los objetos que se designa mediante algún término.
* Representado por una o más palabras para designar cierta imagen que se produce en la mente del individuo.

**Palabra enlace**

* Palabras que sirven para unir conceptos y señalar el tipo de relación existente entre ellos.

#### Proposiciones

* Dos o más términos conceptuales (conceptos) unidos por una frase (palabras enlaces) para formar una unidad con significado lógico.
* Es la unidad semántica más pequeña que tiene valor de verdad, se afirma o niega algo de un concepto.

# UNIDAD II

## Sistemas de Información

**Dato vs. Información**

Los **datos** son símbolos que describen algo. Una estructura de datos es una representación de la relación lógica que existe entre los elementos de datos individuales.

* Los usuarios requieren INFORMACIÓN para la toma de decisiones.
* Los diseñadores deben tener la libertad de proponer distintas estructuras de DATOS que brinden la INFORMACIÓN al usuario.
* Las estructuras de DATOS deben ser eficientes.

En cuanto a la **Información**,

* Conjunto de datos que permiten aclarar algo sobre aquello que es desconocido.
* Conjunto de estímulos que desencadenan el comportamiento. Los datos, en cambio, son símbolos que describen a un objeto.

**Información = Datos procesados por el usuario para la toma de decisiones**

### Atributos de la información

**Objetivos**:la información debe tener un objetivo o fin en el momento de ser trasmitida sino será solamente datos o ruido.

**Forma de representación**: sensorial al ser humano, vista, el oído, el tacto, el gusto y el olfato; en las organizaciones, el formato es visual y auditivo.

**Redundancia**: es exceso de información.

**Frecuencia**: cantidad de veces, en una unidad de tiempo, que ésta se transmite. Repercute en su valor.

**Costo**: es el precio de conseguirla y es un factor limitante en su obtención. Está íntimamente vinculada con el valor de información.

**Valor**: representa la utilidad que ésta tiene para la toma de decisiones.

**Densidad**: representa el “volumen” de información. Un informe escrito tendrá menor densidad que un gráfico o una tabla.

**Consultas y actualizaciones de datos**

* Las **consultas** no modifican los contenidos de las estructuras de datos, solamente inspeccionan su contenido.
* Las **actualizaciones**, esto es ingresar, modificar o eliminar uno o más datos, sí producen modificaciones en su contenido.
* El tipo de estructura condiciona estas operaciones.

### Sistemas

* Conjunto de componentes (subsistemas) que se interrelacionan con el fin de lograr un objetivo.
* El concepto de sistema/subsistema/suprasistema es relativo al observador.
* Las propiedades de los sistemas son también aplicables a los sistemas de información.
* El conocimiento de esas propiedades será de utilidad en el diseño y construcción de sistemas de información.

Ejemplo de definición de "Sistemas"

Un grupo de cuerpos que interactúan entre sí bajo las influencias de fuerzas relacionadas. Por ejemplo, el sistema gravitacional.

### Características generales de los sistemas

* Los sistemas tienen objetivos, metas o fines hacia los cuales pretende llegar.
* El ambiente, en donde está inmerso el sistema, es todo lo que está fuera de él; engloba lo que está fuera del control del sistema.
* El ambiente actúa sobre el sistema cuando le provee insumos y cuando recibe de él sus salidas.
* Los sistemas tienen límites que los separan de su medio ambiente.
* Frontera distinción entre sistema abierto y cerrado.
* Los límites se definen con relativa facilidad en los sistemas naturales y físicos, pero son muy difíciles de delinear en los sistemas sociales, tales como los organizacionales.

### Conceptos básicos de los sistemas

**Adaptabilidad**: es la propiedad de los sistemas de aprender y modificar algunas características de acuerdo a las modificaciones del contexto.

**Caja** **negra**: representa a los sistemas que no se sabe, o no interesa en ese momento, qué elementos lo componen, pero sí se sabe que a determinadas entradas corresponden determinadas salidas.

**Mantenibilidad**: constantemente en funcionamiento.

**Entradas**: ingresos al sistema, estos pueden ser recursos materiales, humanos o información.

**Proceso**: transforma una entrada en salida.

**Salidas**: los resultados que se obtienen de procesar las entradas.

**Alimentación delantera**: forma de control de los sistemas, se realiza a la entrada del mismo.

**Sistemas abiertos**: son aquellos que se vinculan con el medio ambiente.

**Sistemas cerrados**: son aquellos que no interactúan con el medio ambiente.

**Frontera**: línea que separa al sistema de su entorno y que define lo que le pertenece y lo que queda fuera de él.

**Subsistemas**: conjunto de elementos y relaciones que responden a estructuras y funciones especializadas dentro de un sistema mayor.

### Sistemas de Información

Sistemas específicos que se utilizan para asistir a la toma de decisiones.

Comprender el funcionamiento de un sistema puede ser una tarea muy compleja. Por lo tanto, se realiza, la descomposición funcional, esto es, la división de un sistema en partes.

#### Descomposición funcional

La **cohesión** es la relación interna que existe entre los componentes de un sistema, esta ligazón alta.

El **acoplamiento** es la relación que existe entre los procesos después de la descomposición y es recomendable baja.

### Tipos de sistemas de información

**Información Gerencial:**

Sistemas de procesamiento que no toman decisiones por sí mismo. Auxilian a los gerentes y otros profesionales de una organización a tomar decisiones. (Control administrativo).

**Apoyo a la toma de decisiones**:

Utilizados por los gerentes para evaluar y analizar la gestión de la organización. Estos sistemas ofrecen indicaciones o sugerencias más amplias y generales. (Administración estratégica) **Procesamiento de datos**:

Sistemas de control operativo, son los que se utilizan para procesar datos. Por ejemplo, en una casa de comidas rápida, se utilizan para tomar pedidos y emitir facturas.



# UNIDAD III

## Uso de modelos

Métodos de análisis se basan en la construcción de un conjunto de modelos del sistema que se pretende desarrollar.

**Ventajas del uso de modelos:**

* Centrarse en determinadas características del sistema, dejando de lado las menos significativas.
* Enfocar las discusiones con el usuario en los aspectos más importantes.
* Realizar cambios y correcciones en los requisitos a un bajo costo y sin riesgo.
* Verificar que el analista haya entendido correctamente las necesidades del usuario.

### Diagrama de flujos de datos (DFD)

El diagrama de flujos de datos es una técnica gráfica que representa el flujo de datos y las transformaciones que se aplican a ellos, pero no cuándo se realizan o en qué secuencia.

### Procesos

Los procesos son transformadores de flujos. Muestran una parte del sistema que transforma datos de entrada en datos de salida. Estos evitan mostrar detalles procedimentales.

#### Entidades externas

Representan elementos externos al sistema: personas, organizaciones u otros sistemas que interactúan con él. Están fuera de los límites del sistema y proporcionan datos que serán transformados por el sistema o consumen los datos que fueron transformados por el sistema.

#### Flujo de datos

Representan datos o estructuras de datos que fluyen a través del sistema. La flecha indica el sentido de flujo. Estos conectan procesos entre sí, con entidades externas o con almacenamientos de datos.

#### Almacén de datos

Describen los datos persistentes que serán utilizados por el sistema. Permiten guardar, temporalmente, datos que luego podrán ser procesados por el mismo proceso que los creó o por otro distinto.

* Si un flujo de datos ingresa a un almacenamiento se interpreta como una modificación del contenido de su estructura.
* Si un flujo de datos parte desde un almacenamiento a un proceso expresa la lectura de uno o más elementos, no hay modificación del contenido de su estructura de datos.

#### Diagrama de contexto

* Recibir entradas.
* Procesarlas de acuerdo con determinadas funciones. ➢ Generar las salidas.

#### Procesos primitivos

* Los diagramas de flujo de datos no muestran detalles procedimentales.
* Los diagramas de flujo de datos de menor nivel, más altos en la jerarquía, pasan por un proceso de descomposición en el cual, los procesos se describen mediante un nuevo diagrama de flujo de datos que define, más detalladamente, las funciones que realiza y los flujos que maneja. Esto se realiza con el fin de que un proceso pueda ser descrito de forma sencilla y no ambigua.
* Los nombres de los procesos deben ser explícitos en relación a lo que hace el proceso.

#### Diccionario de datos

* Listado organizado que contiene detalles de los modelos del sistema. ➢ Definiciones precisas y rigurosas.
* Define el significado de los flujos y almacenamientos, de los componentes de la especificación de proceso y de los elementos del modelo entidad interrelación.

#### Dato elemental

Es la mínima unidad indivisible, es el componente con mayor nivel de detalle en las estructuras de datos del sistema.

* Nombre
* Descripción
* Longitud
* Tipo
* Dominio

#### Estructura de datos

Los datos elementales se agrupan en estructuras para describir componentes del sistema. Estas estructuras se construyen a partir de un conjunto de relaciones entre los componentes.

**Especificación de procesos**

Se encarga de definir, de forma clara y no ambigua, las funciones y restricciones del sistema, de forma tal de evitar problemas en las etapas de diseño y codificación.

#### Árbol de decisión

Diagrama que muestra en forma secuencial condiciones y acciones, presenta qué condiciones se consideran en primer lugar, cuál en segundo y así sucesivamente hasta que al final se despliega una acción determinada.

**Tabla de decisión**

La tabla de decisión se utiliza para especificar procesos complejos.

#### Pseudocódigo

Lenguaje similar a los convencionales, no tiene una sintaxis tan rígida. Describe qué es lo que sucede en cada burbuja primitiva del diagrama de flujo de datos, sin exigir decisiones prematuras. Su propósito es definir lo que debe hacerse para transformar entradas en salidas.

### Estructuras

**De secuencia**: una entrada y una salida dentro de la cual se encuentran una serie de acciones cuya ejecución es lineal y en el orden en que aparecen.

**De decisión**: una entrada y una salida, pero dentro de la misma se realiza una acción de entre varias, según una condición preestablecida.

**De repetición**: una entrada y una salida dentro la cual se repite una acción, que generalmente es una estructura de secuencia.

## Diagrama de entidad relación

Un diagrama entidad relación (DER) es un tipo de diagrama de flujo que ilustra como las “entidades” (personas, objetos o conceptos) se relacionan entre si dentro de un sistema.

Se caracteriza por utilizar una serie de símbolos y reglas para representar los datos y sus relaciones.

**Entidades fuertes y débiles**

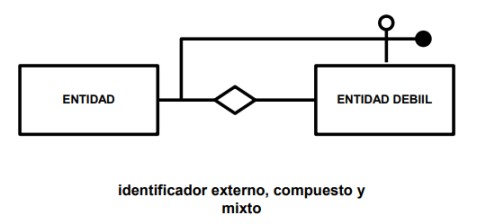
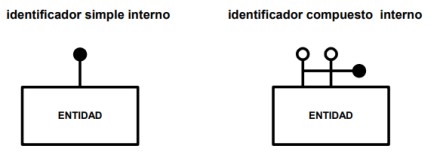
Una entidad fuerte tiene existencia propia. Una entidad débil su existencia depende de otra entidad.

### Atributos

Las entidades poseen atributos que las describen o identifican. Estos pueden ser:

* Descriptivos
* Simples / Compuestos. ➢ Derivables.

### Identificadores

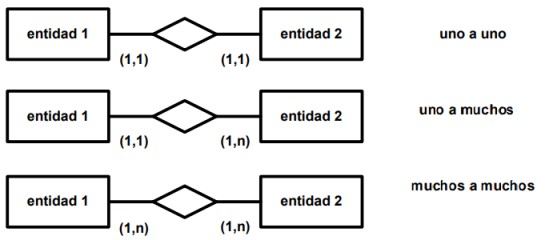


### Interrelaciones

Las interrelaciones pueden tener atributos los cuales se denominan **atributos descriptivos.**

### Multiplicidad de relaciones

Para analizar la multiplicidad, parto de un objeto de la entidad y pregunto con cuántos objetos de la otra puede conectarse.



**Casos de uso y requisitos**

**¿Qué es el software?**

**INGENIERÍA DE REQUISITOS**

La Ingeniería de Requisitos direcciona el proceso de elicitación (inducir), definición, modelado, análisis, especificación y validación de los requisitos de un sistema y de su software, basado en un enfoque sistemático, separando el "qué" del "cómo" del diseño.

**EL PROCESO DE INGENIERÍA DE REQUISITOS**

El Proceso de Desarrollo de Software está formado por un conjunto de actividades organizadas y secuenciadas para transformar los requisitos del usuario en un sistema/software.

Este proceso se caracteriza por:

* Estar dirigido por casos de uso
* Estar centrado en la arquitectura
* Ser iterativo e incremental

Los casos de uso representan los requisitos funcionales y guían el diseño, la implementación y la prueba. Basándose en los casos de uso, los desarrollares crean modelos de diseño e implementación.

Se dice que el proceso está centrado en la arquitectura porque es una vista de diseño con las características más importantes, dejando los detalles de lado.

**¿QUÉ ES UN REQUISITO?**

Flujo de trabajo fundamental cuyo propósito esencial es orientar el desarrollo hacia el sistema correcto.

Un requerimiento es una condición o capacidad a la que el sistema (siendo construido) debe conformar.

**¿QUÉ ES UN REQUISITO DE SOFTWARE?**

Una capacidad del software necesaria por el usuario para resolver un problema o alcanzar un objetivo, que sirva para la toma de decisiones.

Una capacidad del software que debe ser reunida o poseída por un sistema o componente del sistema para satisfacer un contrato, especificación, estándar, u otra documentación formal.

Es una descripción completa del comportamiento del sistema que se va a desarrollar.

**ESTRUCTURA DEL DOCUMENTO DE REQUERIMIENTOS (ERS):**

❑ Definición de Requerimientos No-funcionales: Definir las limitantes del sistema y del proceso de desarrollo.

❑ Evolución del Sistema: Definir las suposiciones fundamentales en las cuales el sistema se basa y se anticipan los cambios.

❑ Especificación de Requerimientos: Especificación detallada de los requerimientos funcionales del sistema.

❑ Apéndices: Descripción de la plataforma de Hardware del Sistema. Requerimientos de la base de Datos (quizá como un modelo ER)

❑ Índice.

**CARACTERÍSTICAS DE UNA BUENA ERS**

Las características de una buena ERS son definidas por el estándar IEEE 830-1998. Una buena ERS debe ser:

1. Completa: Todos los requerimientos deben estar reflejados en ella y todas las referencias deben estar definidas.
2. Consistente: Debe ser coherente con los propios requerimientos y también con otros documentos de especificación.
3. Inequívoca: La redacción debe ser clara de modo que no se pueda mal interpretar.
4. Correcta: El software debe cumplir con los requisitos de la especificación.
5. Trazable: Se refiere a la posibilidad de verificar la historia, ubicación o aplicación de un ítem a través de su identificación almacenada y documentada.
6. Priorizable: Los requisitos deben poder organizarse jerárquicamente según su relevancia para el negocio y clasificándolos en esenciales, condicionales y opcionales.
7. Modificable: Aunque todo requerimiento es modificable, se refiere a que debe ser fácilmente modificable.
8. Verificable: Debe existir un método finito sin costo para poder probarlo.

**POSIBLES ACTORES EN LOS REQUISITOS DE INGENIERÍA**

Usuario Final: Son personas que utilizarán el sistema en desarrollo. Quienes utilicen las interfaces y manuales de usuario.

Usuario Líder: Son las personas que comprenden el ambiente del sistema y el dominio del problema. Proporcionan información al equipo técnico (detalles de los requerimientos, interfaces, etc.).

Personal de mantenimiento: Son quienes administran los cambios y mejoran los procesos.

Analistas y Programadores: Son los responsables del desarrollo del producto.

Personal de pruebas: Son quienes elaboran y ejecutan los planes de pruebas.

Administradores de proyectos: Personas que planifican, organizar y controlan un proyecto informático. Gestiona todos los recursos necesarios para logran el fin perseguido.

**ELICITACIÓN**

Es el proceso de adquirir (“eliciting”) [sonsacar] todo el conocimiento relevante necesario para producir un modelo de los requerimientos de un dominio de problema.

Objetivo: entender el dominio del problema en particular.

**TÉCNICA DE ELICITACIÓN**

• Entrevista de comienzo y final abierto

• Entrevistas estructuradas

• Brainstorming (Lluvia de Ideas)

• Prototipos

• Escenarios

• Observación

**CATEGORÍAS DE PROBLEMAS EN LA ELICITACIÓN DE LOS REQUISITOS**

Los problemas se pueden dividir en 3 grupos:

* Problemas de Alcance:

1. Organización -> entender metas.
2. Entorno -> analizar usuario y ambiente de trabajo.
3. Proyecto -> restricciones impuestas por los involucrados (Costo, calidad, tiempo).

* Problemas de Comunicación:

1. Deficiente comunicación entre usuarios y técnicos.
2. Muchas comunidades involucradas.
3. Estructura adecuada de la información.

* Problemas de volatilidad:

1. Requisitos no permanecen estáticos.
2. Negocio cambia con el tiempo (metas, objetivos, etc.).

**PROBLEMAS PARA IDENTIFICAR REQUISITOS**

--No siempre son obvios y tienen diversas fuentes

– No son fáciles de identificar con palabras (Métodos formales)

– Muchos tipos distintos, a distinto nivel de detalle

– El número puede llegar a ser inmanejable

– Muchos Interesados y partes responsables

– Pueden ser afectados por el tiempo y cambiar a lo largo del ciclo de vida de desarrollo

– Se interrelacionan con otros

– Es difícil manejar un vocabulario común con expertos del negocio.

**TIPOS DE REQUISITOS/**

Existen varios tipos de requisitos como lo son:

* Requisitos de Usuarios: Necesidades que los usuarios expresan verbalmente.
* Requisitos del Sistema: Son los componentes que el sistema debe tener para realizar determinadas tareas.
* Requisitos Funcionales: Servicios que el sistema debe proporcionar.
* Requisitos no funcionales: Restricciones que afectan al sistema.

**TIPOS DE REQUISITOS**

* Funcionales: Especifica una acción que debe ser capaz de realizar el sistema, sin considerar restricciones físicas.

Describen la funcionalidad o los servicios que se espera proveerá el sistema.

* No funcionales: Especifica restricciones físicas sobre un requisito funcional (rendimiento, plataforma, fiabilidad)

No se refieren directamente a las funciones específicas que entrega el sistema, sino a las propiedades emergentes de éste como la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento.

A nivel hardware y Redes.

**GESTIÓN DE LOS REQUISITOS**

La Gestión de requisitos es el proceso de gestionar los cambios en los requisitos de un sistema y se integra en la Gestión del proyecto.

BENEFICIOS DE LA GESTIÓN DE LOS REQUISITOS

* Mejor control de proyectos complejos.
* Mejora en la calidad del software y en la satisfacción del cliente.
* Reducción en los retrasos y en los costos del proyecto.
* Mejora en la comunicación del equipo.
* Facilita la conformidad con estándares y regulaciones.

**REQUISITOS DE CASOS DE USO**

Los casos de uso son una técnica para la especificación de requisitos funcionales, presentan una ventaja sobre la descripción textual de los requisitos funcionales.

Los casos de uso son, ante todo, requisitos funcionales.

**¿QUÉ ES UN CASO DE USO?**

Los casos de uso son una técnica que se utiliza para documentar los requerimientos funcionales de un sistema desde el punto de vista de los usuarios.

Un caso de uso es un texto muy simple con cierto formato que describe cómo se debería comportar un sistema ante la interacción con uno o más usuarios.

**CASOS DE USO**

Describe tanto lo que hace el actor como lo que hace el sistema cuando interactúa con él.

Un caso de uso realiza cierto trabajo cuyo efecto es tangible para un actor.

**CASOS DE USO – ACTORES**

Un actor es alguien o algo que interactúa con el sistema (una persona, una organización, un programa o sistema de hardware o software).

Un actor estimula al sistema con algún evento o recibe información del sistema.

Un actor es externo al sistema.

Un actor cumple un rol definido (Ej.: Cliente, Banco, empleado).

**TIPO DE ACTORES**

Actores primarios: utilizan las funciones principales del sistema.

Actores secundarios: efectúan tareas administrativas o de mantenimiento.

**CASOS DE USO Y ESCENARIOS**

Cada secuencia específica del caso de uso se denomina Escenario.

Un escenario es una secuencia específica de acciones entre los actores y el sistema (es una instancia de un caso de uso).

**CASOS DE USO Y COLABORACIONES**

De todas maneras, un caso de uso deberá implantarse y esto lo hará mediante una sociedad de clases (incluyendo su estructura estática y dinámica) que se modela como una colaboración.

**PRE Y POST CONDICIONES**

Precondiciones: establece lo que siempre debe cumplirse antes de comenzar un caso de uso. No se prueban en el caso de uso, se asumen que son verdaderas.

Post condiciones: establece qué debe cumplirse cuando el caso de uso se completa con éxito.

**LOS CASOS DE USO Y LAS FASES EN EL UP**

* En la fase de inicio se reconocen la mayoría de los casos de uso, pero no su descripción detallada.
* En la fase de elaboración, se refinan en las sucesivas iteraciones.
* En la fase de construcción, se escriben casos de uso menores.
* En la fase de transición, no se describen casos de uso.

**Problema**: los límites del sistema no están definidos o son inconsistentes.

**Síntoma**: No se entiende “el sistema” Mirando el modelo de casos de uso no queda del todo claro que hay dentro y que hay fuera.

**Cura**: Ser explícito acerca del contexto, dibujar el diagrama de contexto aunque sea sencillo, identificar el mismo acorde a su descripción. Documentar y entender alcances, límites y objetivos.

**Problema**: los nombres de los casos de uso están escritos desde el punto de vista del sistema.

**Síntoma**: Los nombres describen lo que el sistema hace, en lugar de describir lo que el actor tiene como objetivo. Los pasos de la especificación describen la funcionalidad interna en lugar de la interacción a través de la frontera del sistema. El modelo parece un diagrama de flujos.

**Cura**: Nombrar los casos de uso desde la perspectiva del actor. Enfocarse en lo que el Sistema tiene que hacer para lograr los objetivos, en lugar de como lo va a hacer. Optimizar la descomposición funcional y prestar atención a las relaciones (extend, include, use) cuando no hay actores.

**Problema**: los nombres de los actores son inconsistentes.

**Síntoma**: Diferentes nombres describen el mismo Rol, ya que muchas veces los requisitos vienen de diversas fuentes. Cuanto más grande es el problema, más personas colaboran en la solución.

**Cura**: Establecer un glosario al principio del proyecto y utilizarlo para describir los actores, donde se especifique el Nombre del Actor, su significado, y los posibles alias del mismo.

**Problema**: muchos casos de uso.

**Síntoma**: El modelo de Casos de Uso tiene un gran número de CU.

**Cura**: Unir CU triviales que sean fragmentos de otros CU Reales. Eliminar CU que describen puramente procesos “internos”, con respecto a los límites del Sistema.

**Problema**: las relaciones entre actores y CU se asemejan a una telaraña.

**Síntoma**: Demasiadas Relaciones entre Actores y CU. Un actor interactúa con cada CU / Un CU interactúa con cada actor.

**Cura**: Examinar los actores para descubrir distintos niveles de abstracción para encontrar los más explícitos, los cuales participarán en un conjunto más limitado de tareas.

**Problema**: la Especificación del Caso de Uso es muy extensa.

**Síntoma**: Demasiadas Relaciones entre Actores y CU. Un actor interactúa con cada CU / Un CU interactúa con cada actor.

**Cura**: Definir casos de uso más específicos y más definidos con el fin de bajar la granularidad de los pasos. Quitar los pasos que describen funcionamiento interno (de implementación). Volver a escribir el caso de uso centrado en la interacción esencial.

**Problema**: la Especificación del Caso de Uso es confusa.

**Síntoma**: El CU carece de Contexto, “no cuenta una historia”. Un actor interactúa con cada CU / Un CU interactúa con cada actor. Los pasos se ven como un programa de computadora.

**Cura**: Especificar donde el CU es relevante. Reescribir la secuencia centrado en las interacciones esenciales. Romper los flujos donde exista la palabra “si” en flujos alternativos, dejando el flujo principal más fácil de entender y más corto. No describir los algoritmos con el CU (usar árboles de decisión, tablas de decisión, pseudocódigo, etc.). Asegurarse que ningún paso especifique implementación.

**Problema**: el CU no describe correctamente el requisito funcional.

**Síntoma**: Las relaciones entre Actores y CU no describen qué puede hacer quien con el Sistema.  Se confunden los casos de uso CRUD, mezclando en muchas oportunidades creación y edición dentro de un mismo caso de uso.

**Cura**: Asegurarse que cada actor asociado a un CU esté autorizado para llevarlo a cabo.

**Problema**: el cliente no entiende los Casos de Uso.

**Síntoma**:

* El cliente no sabe nada de CU, pero se le ha dado un documento funcional para su aprobación.
* Los casos de uso no cuentan una historia.
* La organización de los CU no coincide como el cliente piensa el problema.
* La especificación está escrita en lenguaje técnico y específico.
* El cliente odia los CU.

**Cura**:

* Enseñar lo suficiente para entender cada documento funcional.
* Agregar información necesaria para que sea una historia. Escribir contexto.
* Escribir en vocabulario del cliente.
* Entregar la documentación que el cliente quiere, aunque se usen CU para elicitar y documentar requisitos funcionales.

**Problema**: los casos de uso nunca se terminan.

**Síntoma**:

* Los CU deben cambiar con cada cambio de la UI.
* Demasiados flujos alternativos posibles.
* Requisitos desconocidos.

**Cura**:

* Desacoplar detalles de la interfaz de usuario en la especificación.
* No poner diseño en los CU.
* Definir bien el alcance y contextualizar los CU con otros.

**Introducción al Paradigma OO**

**PARADIGMA:** Es una forma de ver el Mundo.

Un paradigma es una forma de entender y representar la realidad: un conjunto de teorías, estándares y métodos que, juntos, representan un modo de organizar el pensamiento.

**ENFOQUE TRADICIONAL:**

El enfoque tradicional del análisis y diseño estructurados, se descompone el problema en funciones o procesos y estructuras de datos.

**ENFOQUE ORIENTADO A OBJETOS:**

En un enfoque OO se busca descomponer el problema, no en funciones, sino en unidades más pequeñas denominadas objetos.

**PARADIGMA ORIENTADO A OBJETOS:**

El paradigma OO define los programas en términos de comunidades de objetos. Los objetos con características comunes se agrupan en clases.

**DISEÑO ORIENTADO A OBJETOS:**

El diseño orientado a objetos es fundamentalmente diferente a los enfoques de diseño estructurado tradicionales: requiere un modo distinto de pensar acerca de la descomposición, y produce arquitecturas de software muy alejadas del dominio de la cultura del diseño estructurado.

**TECNOLOGIA ORIENTADO A OBJETOS:**

La tecnología orientada a objetos se apoya en los sólidos fundamentos de la ingeniería, cuyos elementos reciben el nombre global de modelo de objetos.

**METODOLOGÍA ORIENTADA A OBJETOS**

• Análisis orientado a objetos: Es un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos que se encuentran en el vocabulario del dominio del problema.

• Diseño orientado a objetos: Es un método de diseño que abarca el proceso de descomposición orientada a objetos y una notación para describir los modelos lógico y físico, así como los modelos estático y dinámico.

• Programación orientada a objetos: Es un método de implementación en que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representa una instancia de alguna clase, y cuyas clases son, todas ellas, miembros de una jerarquía de clases unidas mediante relaciones de herencia.

**CLASES**

Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.)

Una clase es un conjunto de objetos que comparten una estructura común y un comportamiento común.

Una clase especifica una estructura de datos y las operaciones permisibles que se aplican a cada uno de sus objetos.

Los objetos (Un objeto es una instancia de una clase) se vinculan mediante enlaces enviando mensajes a operaciones que activan los métodos:

Mensaje: es una solicitud para que se lleve a cabo la operación indicada y se produzca el resultado.

Operaciones: es una función o transformación que se aplica a un objeto de una clase.

Métodos: es la implementación de una operación.

**RELACIONES BÁSICAS ENTRE CLASES🡨-🡪RELACIONES DERIVADAS ENTRE CLASES**

• Asociación: La asociación es una relación bidireccional. La relación entre clases conocida como Asociación, permite asociar objetos que colaboran entre sí. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro. Ej: dada una instancia de cliente podríamos encontrar el objeto que denota sus compras. Posee CARDINALIDAD y esta puede ser:

- UNO A UNO

- UNO A MUCHOS

- MUCHOS A MUCHOS

• Herencia: Capacidad por la cual una clase de orden inferior puede recibir estructura o acciones de una o más clases de orden superior. La subclase posee la capacidad de incorporar parte estructural y acciones propias. Existen dos tipos de herencia:

- HERENCIA SIMPLE

- HERENCIA MULTIPLE

• Polimorfismo: Capacidad por la cual una acción puede responder de distinta forma de acuerdo a la subclase que la implementa.

• Agregación: La instancia se crea afuera, pero la agregación se hace dentro. Ej: Cuando se destruye el almacén, no se destruyen los clientes.

• Composición: También llamada “agregación con contención física”. La creación de la instancia se hace dentro. Cuando un objeto no existe sin el otro. Sus ciclos de vida están íntimamente relacionados. Ej: cuando se destruye el almacén, también las cuentas.

• Uso: Es una asociación refinada, donde se establece que abstracción es cliente y cual servidor. El cliente hace USO del servidor.

• Instanciación: Acción por la cual se crean instancias de una clase. Los objetos creados corresponden al tipo de la clase que los origina.

• Metaclase: Es la clase de una clase. Es una clase donde las instancias son ella misma.

**DIAGRAMA DE CLASES**

Es muy importante interpretar cada uno de los elementos del diagrama de clases, ya que esto va a ser la estructura del funcionamiento del sistema:

• Navegabilidad: describe en la relación, que clase contiene el atributo.

• Multiplicidad: describe la relación según la cantidad de instancias.

Un diagrama de clases sirve para mostrar las relaciones entre las clases involucradas en el sistema.

Está compuesto por:

* Clases
  + Atributos
  + Métodos
  + Visibilidad (público, privado, protegido, etc.)
* Relaciones (Estructurales) y multiplicidad

**Multiplicidad**

Determina cuantos objetos de cada tipo Intervienen en una relación.

Cuantas instancias de UNA CLASE que se relacionan con UNA instancia de otra clase.

Tabla

Descripción generada automáticamente

**Visibilidad**

- Publico (+) es accesible desde todos lados.

- Privado (-) es accesible solo dentro de la clase.

- Protegido (#) es accesible por las clases derivadas.

**OBJETOS**

“Un objeto es cualquier cosa real o abstracta, acerca de la cual almacenamos datos y las operaciones que controlan dichos datos”

Los objetos son entidades que combinan un estado (es decir, datos) y un comportamiento (esto es, procedimientos o métodos). Estos objetos se comunican entre ellos para realizar tareas”.

**Propiedades de los Objetos**

El estado: de un objeto abarca todas las propiedades (normalmente estáticas) del mismo, más los valores actuales (normalmente dinámicos) de cada una de esas propiedades.

El comportamiento: nos muestra cómo actúa y reacciona un objeto, en términos de sus cambios de estado y paso de mensajes.

La identidad: es aquella propiedad de un objeto que lo distingue de todos los demás objetos.

**Abstracción**

La abstracción es la propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las restantes características (no esenciales).

Desde el punto de vista Orientado a Objetos, la abstracción:

- Expresa las características esenciales de un objeto, la cual distingue al objeto de los demás.

- Provee límites conceptuales.

**Encapsulamiento**

El Encapsulamiento es la propiedad que permite asegurar que el contenido de la información de un objeto está oculta al mundo exterior.

El encapsulamiento, al separar el comportamiento del objeto de su implantación, permite la modificación de éste sin que se tengan que modificar las aplicaciones que lo utilizan.

Si un objeto tiene más características de las necesarias los mismos resultarán difíciles de usar, modificar, construir y comprender.

**¿QUÉ OBSERVAR PARA SABER SI UNA CLASE O UN OBJETO ESTÁN CONSTRUIDOS CON CALIDAD?**

• Acoplamiento: Medida de la fuerza de la asociación establecida por una conexión entre dos objetos.

• Cohesión: Establece el grado de conectividad interna. Indica que tan específico es un objeto.

• Suficiencia: Indica si se poseen las características necesarias de la abstracción como para permitir una interacción significativa y eficiente.

• Complexión: Indica si la interfaz de la clase captura todas las características de la abstracción.

• Primitivas: Denotan operaciones sencillas que se pueden acceder solo por medio de la representación interna básica de la abstracción

RELACIONES DE ASOCIACIÓN

“Se descompone (clases) para comprender, se une (asociación) para construir”

Los enlaces entre objetos son instancias de la asociación entre sus clases

RELACIONES DE JERARQUÍA

La generalización consiste en factorizar los elementos comunes de un conjunto de clases en una clase más general llamada superclase

La herencia es una técnica de los lenguajes de programación para construir una clase a partir de una o varias clases, compartiendo atributos y operaciones

**Análisis OO**

**El enfoque tradicional** del análisis y diseño estructurados, se descompone el problema en funciones o procesos y estructuras de datos.

**En un enfoque OO** se busca descomponer el problema, no en funciones, sino en unidades más pequeñas denominadas objetos.

**Durante el análisis OO** se presta especial atención a encontrar y describir los objetos (conceptos) del dominio del problema.

**Durante el diseño OO** se presta atención a la definición de los objetos software y en como colaboran para satisfacer los requisitos.

Análisis = descomposición de un dominio de interés en clases conceptuales

**La finalidad del análisis OO** es crear una descripción del dominio desde una perspectiva de clasificación de objetos: identificación de conceptos, atributos e interrelaciones significativas.

**El modelo del dominio** NO es una descripción de los objetos software, es una visualización de los conceptos del mundo real y sus vinculaciones (se representan mediante diagrama de clases, sin operaciones).

Modelo del dominio = representación visual de las clases conceptuales del mundo real

Se visualizan en el modelo de dominio:

• Clases conceptuales

• Asociaciones entre clases conceptuales

• Atributos de las clases conceptuales

**La vista estática** se ocupa de definir la estructura del sistema en cuestión. Está compuesta por las clases y las relaciones que dan soporte para cumplir los planteado en los RF.

* Modelo de Dominio.

**La vista dinámica** define las responsabilidades que tendrá el sistema.

* Diagrama de Secuencia del Sistema.

**CLASES CONCEPTUALES**

Una clase conceptual se puede considerar en términos de:

Símbolo: palabras o imágenes que representan la clase conceptual.

Intensión: la definición de la clase conceptual.

Extensión: el conjunto de ejemplos a los que se aplica la clase conceptual.

**MODELO DEL DOMINIO**

Análisis = descomposición de un dominio de interés en clases conceptuales.

Modelo del dominio = representación visual de las clases conceptuales del mundo real.

Se visualizan en el modelo de dominio:

• Clases conceptuales.

• Asociaciones entre clases conceptuales.

• Atributos de las clases conceptuales.

**DIAGRAMA DE SECUENCIA DE SISTEMA (DSS)**

Los diagramas de secuencia de sistema se utilizan en la etapa de análisis para documentar casos de uso.

Muestra, para un escenario específico de un caso de uso, los eventos que generan los actores externos.

Muestran los mensajes que podrían ser traducidos a operaciones dentro del sistema.

**Procesos de negocios**

**Los Sistemas de Información**

**¿Qué Rol cumplen los Sistemas de Información en las organizaciones?**

El objetivo de los Sistemas de Información es ayudar al desempeño de las actividades en todos los niveles de la organización.

**¿Cómo intentan lograrlo?**

Por ello, y para poder desarrollar SI exitosos dentro de las organizaciones, es preciso conocer bien las funciones, actividades y procesos que realiza la organización.

**¿Qué es un proceso de Negocio?**

Los procesos de negocios se refieren a la manera de organizar, coordinar y enfocar el trabajo para elaborar un producto o servicio valioso.

**¿Qué es un proceso de Negocio?**

Un conjunto estructurado, medible de actividades diseñadas para producir un producto especificado, para un cliente o mercado específico. Implica un fuerte énfasis en CÓMO se ejecuta el trabajo dentro de la organización, en contraste con el énfasis en el QUÉ, característico de la focalización en el producto.

**¿Qué es un proceso de Negocio?**

Un proceso de negocio contiene actividades con propósito, es ejecutado colaborativamente por un grupo de trabajadores de distintas especialidades, con frecuencia cruza las fronteras de un área funcional, e invariablemente es detonado por agentes externos o clientes de dicho proceso.

**Optimización**

Flujo = simplificarlo, de modo que no haya redundancia de trabajo, demoras, pasadas por un mismo lugar, etc.

Costo = minimizar el número de personas que participan y racionalizar el uso de los recursos que intervienen.

Tiempo = reducirlo, que tienda a tiempo real o bien lo antes que sea posible (la salida debe estar disponible en el momento que se requiera).

Calidad = mejorarla de modo de conseguir el 100% de lo indicado en las normas correspondientes.

Espacio = desde el punto de vista de la información, reducir el espacio que media entre una estación de trabajo y otra, de modo que en vez que viaje el medio que lo contiene, se haga viajar sólo los datos que son propios de esa entrada o salida.

Servicio = satisfacer a plenitud al cliente que recibe la salida de una acción, de modo de lograr su fidelización.

**Forma de Representar**

**Tipos de Notaciones de Modelado**

Dependiendo de las metodologías y estrategias empleadas:

• Orientados a proceso: Se centran en las diferentes tareas a completar para llevar a cabo un proceso completo.

• Orientados a recurso: Se centran en la utilización y distribución de los recursos que son necesarios para llevar a cabo la realización del proceso.

• Orientados a datos: Se centran en la definición de los datos y en las transformaciones que sufren estos al largo del proceso.

**Modelado del Negocio Diagramas de Flujo**

Representación abstracta (gráfica) de los procesos de una organización, que muestran principalmente cómo y por quién son llevadas a cabo las actividades que generan valor para la organización.

**Herramientas de Modelado BMM**

Método BMM (Business Modeling Method)

Método de Modelado de Negocios orientado al desarrollo de sistemas de información empresarial.

Divide el Sistema de Negocios en 3 niveles:

\* Objetivos

\* Procesos

\* Sistemas

Facilita la alineación de los sistemas a los objetivos y procesos

Herramientas de Modelado Diagramas de Actividad

• Es un diagrama UML (OMG).

• Se utiliza para la representación del comportamiento dinámico de un sistema.

• Se centra en la secuencia de actividades que se llevan a cabo.

Diagrama

Descripción generada automáticamenteElementos

**Diagrama, Esquemático

Descripción generada automáticamente**

Nodo Decisión: Guían el flujo en una u otra dirección. Poseen un único flujo de entrada y varios flujos de salida.

Nodo Combinación: Tiene la misma representación que el nodo anterior, pero a diferencia de este tienen múltiples flujos entrada pero un único flujo de salida.

Nodo Fork: Divide un único flujo de entrada en varios flujos de salida que se ejecutarán de manera concurrente.

Nodo Join: Para sincronizar múltiples flujos. Varios flujos de entrada y único flujo de salida

Procesos de negocio abstractos (públicos)

Los procesos de negocio abstractos nos sirven para representar las interacciones existentes entre un proceso de negocio privado y, o bien otro proceso de negocio o bien un participante del proceso

**Proceso de desarrollo de software**

El desarrollo de software es el proceso de dividir el trabajo en distintas fases para mejorar el diseño, la gestión del producto, y la gestión de proyecto. Es también conocido como el ciclo de vida del desarrollo de software.

* *Ingeniería de sistema*: Se analiza el sistema estableciendo los requisitos más generales.
* *Análisis*: Se analizan en detalle los requisitos del sistema y se definen las funciones.
* *Diseño*: Es una especificación técnica de cómo construir el sistema.
* *Codificación*: Inicia la producción del sistema, traduciendo las especificaciones anteriores a software.
* *Prueba*: Se valida y verifica que el software cumpla con los requisitos bajo un entorno controlado.
* *Utilización*: Se utiliza el software en un entorno de producción, en un ambiente real.
* *Mantenimiento*: Se corrigen errores manifestados en la etapa anterior y se atienden nuevos requerimientos del usuario
* *Sustitución*: Cuando el sistema queda obsoleto y su actualización se vuelve muy costosa se opta por sustituir el sistema por otro, dando comienzo a un nuevo ciclo de vida.

**CICLO DE VIDA**

El ciclo de vida de desarrollo de sistemas es una sucesión de etapas por las que atraviesa el software desde que comienza un nuevo proyecto hasta que éste se deja de utilizar.

**CICLO DE VIDA EN CASCADA**

En Ingeniería de software el desarrollo en cascada, también llamado secuencial, es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior.

INGENIERIA DE SISTEMAS: Se analiza el sistema estableciendo los requisitos más generales.

ANALISIS: Análisis de requisitos más detallado.

DISEÑO: Traducción de requisitos en una representación software.

TRADUCCION / CODIFICACION: Traducción a un formato comprensible para la computadora.

PRUEBA: Detección de errores producidos en alguna de las fases anteriores.

MANTENIMIENTO: Adaptación del software a los cambios.

*PROBLEMAS DEL CICLO DE VIDA EN CASCADA*

• Los proyectos de desarrollo de sistemas, en general, no siguen un ciclo de vida estrictamente secuencial, siempre hay iteraciones.

• Dificultad para establecer, inicialmente, todos los requisitos del sistema.

*ÁMBITO DE APLICACIÓN DEL CICLO DE VIDA EN CASCADA*

Sistemas simples y pequeños, donde los requisitos sean fácilmente identificables.

Debido a estos problemas, el ciclo de vida en cascada es utilizado preferentemente en sistemas simples y de baja complejidad, ya que no es difícil identificar los requisitos del software.

**HERRAMIENTAS 4GL**

Conjunto muy diverso de métodos y herramientas que tiene por objeto facilitar el desarrollo de software.

• Generación de código.

• Generación de pantallas y de informes.

• Gestión de entornos gráficos.

• Herramientas de acceso a bases de datos.

La ventaja principal de estas herramientas es la generación automática de código a partir de especificaciones de alto nivel de abstracción.

**CICLO DE VIDA 4GL**

Las fases del ciclo de vida 4G son iguales a las del ciclo de vida en cascada, con la diferencia de que se si la especificación es pequeña, se puede pasar directamente del análisis de requisitos a la generación automática de código, sin realizar ningún tipo de diseño, utilizando las herramientas 4G.

*PROBLEMAS DEL CICLO DE VIDA 4GL*

La mayoría de estas herramientas no logran prescindir totalmente de la codificación. • Normalmente, el código generado es ineficiente.

El problema que se plantea es el mismo que en el ciclo de vida en cascada, es muy difícil que se puedan establecer todos los requerimientos desde el comienzo.

*ÁMBITO DE APLICACIÓN DEL CICLO DE VIDA 4GL*

El ámbito de aplicación de esta técnica está restringido, casi exclusivamente, al software de gestión.

Ya que las herramientas de cuarta generación están orientadas a la generación de informes a partir de grandes bases de datos o a la creación de interfaces hombre-máquina.

**Ciclo de vida 4G**

Las fases de este modelo de ciclo de vida son iguales que las del ciclo de vida en cascada, la diferencia está en que luego de definir los requisitos se puede pasar a la etapa de codificación utilizando herramientas de 4GL, pasando de las etapas de análisis y diseño.

En este modelo el problema es que el código generado por las herramientas 4GL suele ser ineficiente, y que al final se debe codificar de todas maneras.

**PROTOTIPOS:**

El prototipo sirve como mecanismo para identificar los requisitos del software. Se lo considera un proceso iterativo e incremental. Su construcción suele llevar las siguientes etapas:

1. Determinación de requisitos,
2. Diseño rápido, centrado en la arquitectura del sistema y la definición de la estructura de las interfaces.
3. Construcción de prototipo, codificando rápidamente con herramientas 4GL
4. Evaluación de prototipo por el cliente, para que de sus opiniones.
5. Refinamiento del prototipo, a partir de las opiniones del usuario. Esto genera un nuevo prototipo
6. Producto de ingeniería, Luego de varias iteraciones se desarrolla el producto final.

*PROBLEMAS CON LOS PROTOTIPOS*

Uno de los principales problemas con este método es que, con demasiada frecuencia, el prototipo pasa a ser parte del sistema final.

Se olvida aquí que el prototipo ha sido construido de forma acelerada, sin tener en cuenta consideraciones de eficiencia, calidad del software o facilidad de mantenimiento.

*ÁMBITO DE APLICACIÓN DEL PROTOTIPO*

cualquier aplicación que presente mucha interacción con el usuario, o que necesite algoritmos que puedan construirse de manera evolutiva, yendo de lo más general a lo más específico es una buena candidata.

Cuando el sistema no requiere de especificación de grandes cantidades de detalles algorítmicos, ni de muchas especificaciones de procesos para describir algoritmos con los cuales se obtienen resultados.

**Modelo Espiral:**

El modelo en espiral tiene las mejores características del modelo en cascada y de prototipos, pero se diferencia en que agrega el factor de riesgo del proyecto. La metodología consiste en pasar por diferentes etapas completando iteraciones. Con cada iteración se construyen versiones más completas del software y aumenta el tiempo que ocupa cada etapa.

Las etapas serían las siguientes:

1. Planificación: Determinación de objetivos y alcance.
2. Análisis de riesgo: Identificación, evaluación y gestión de riesgos.
3. Ingeniería: Desarrollo de prototipo
4. Evaluación del cliente: El cliente evalúa el prototipo y luego comienza una nueva iteración considerando la experiencia del cliente.

*ÁMBITO DE APLICACIÓN CICLO DE VIDA EN ESPIRAL*

Este modelo es adecuado para proyectos complejos, dinámicos e innovadores.

*PROBLEMAS CICLO DE VIDA EN ESPIRAL*

Las inconvenientes con este modelo son que cada iteración conlleva más tiempo, la identificación de riesgos es una tarea compleja.

Falta de un proceso de guía explícito para determinar objetivos, limitaciones y alternativas.

La determinación del riesgo no es una tarea fácil.

**Proceso unificado de desarrollo (UP):**

El proceso de desarrollo de software es el conjunto de actividades para transformar los requisitos del usuario en un sistema de software.

El proceso unificado está: dirigido por **casos de uso**, centrado en la **arquitectura** y es **iterativo e incremental**.

* Los **casos de uso** representan los requisitos funcionales y guían el diseño, la implementación y la prueba. Basándose en los casos de uso los desarrolladores crean modelos de diseño e implementación que llevan a cabo los casos de uso.
* La **arquitectura** es una vista de diseño con las características más importantes, dejando los detalles de lado. Describe diferentes vistas del sistema. Constituyen la “forma del sistema”, incluye aspectos estáticos y dinámicos. La arquitectura y los casos de uso evolucionan en paralelo.
* **Iterativo e incremental**

**Desarrollo iterativo:**

* El desarrollo se organiza en una serie de mini proyectos de **duración fija**, llamados **iteraciones** (2 a 6 semanas).
* El resultado de cada uno es un sistema que puede ser **probado, integrado y ejecutado**.
* Cada iteración incluye sus propias actividades de: análisis de requisitos, diseño, implementación y prueba.
* El sistema crece de forma **incremental a lo largo** del tiempo, iteración tras iteración.
* El resultado de cada iteración es un sistema **ejecutable**, pero incompleto.
* En general, cada iteración aborda nuevos requisitos y amplía el sistema de forma incremental.
* La salida de una iteración **NO** es un prototipo desechable, es un subconjunto de calidad del sistema final.

Disciplina: conjunto de actividades y artefactos vinculados en un área determinada: requisitos, diseño, implementación, etc.

Fases: periodo de tiempo entre dos hitos principales de un proceso de desarrollo.

**Ciclo de desarrollo**

• Inicio: visión aproximada, incluye: análisis del negocio, alcance, estimaciones imprecisas.

• Elaboración: visión refinada, implementación iterativa del núcleo central de la arquitectura, resolución de riesgos altos, identificación de más requisitos.

• Construcción: implementación iterativa del resto de requisitos de menor riesgo.

• Transición: pruebas beta.

**Fase de inicio**

Actividades a realizar (una o algunas):

* **Visión y análisis del negocio** (objetivos y restricciones de alto nivel).
* **Modelo de casos de uso** (requisitos funcionales y no funcionales relacionados).
* **Especificaciones complementarias** (otros requisitos).
* **Listas de riesgos** (del negocio, técnicos, etc.).
* **Prototipos** (para clarificar la visión).
* **Plan de iteración** (qué hacer en la primera iteración).

**Fase de elaboración**

Actividades a realizar:

* Se descubren y estabilizan la mayoría de los casos de uso.
* Se reducen o eliminan los riesgos importantes.
* Se implementan y prueban los elementos básicos de la arquitectura.
* Duración: 2 a 4 iteraciones con una duración de 2 a 6 semanas (dependiendo de la duración del proyecto).

Artefactos de la fase de elaboración:

* Modelo del dominio (entidades del dominio).
* Modelo de diseño (diagramas de clases, de iteración. etc.).
* Modelo de datos.
* Modelo de pruebas.
* Modelos de implementación.

El producto resultante **no** es un prototipo desechable.

El código y el diseño son de calidad y se integran al sistema final.

**Fase de construcción**

Objetivos:

* Terminar de construir la aplicación.
* Realizar pruebas alfa.
* Preparar pruebas beta (para la transición).
* Preparación de guías de usuario.
* Preparación de materiales de aprendizaje.

**Fase de transición**

Objetivo: poner el sistema en producción.

Actividades:

* Realización de pruebas beta.
* Reaccionar a la retroalimentación a partir de las pruebas beta.
* Conversión de datos.
* Cursos de entrenamiento.

**Metodologías Ágiles:**

Las metodologías ágiles son un conjunto de diferentes estrategias de trabajo aplicadas al desarrollo que tienen en común su ejecución en ciclos iterativos. Esto permite realizar entregas y pruebas parciales del producto al cliente e ir incorporando cambios, hasta obtener la solución completa en el plazo acordado.

La aplicación de estos modelos proporciona mayor rapidez a la ejecución y entrega de soluciones informáticas, además de otras ventajas que explicaremos en este artículo.

**Test Driven Development (TDD)**, o **desarrollo dirigido por test**:

Es una técnica de ingeniería de *software* para diseñar *software*. Como su propio nombre indica, esta técnica dirige el desarrollo de un producto a través de ir escribiendo pruebas, generalmente unitarias.

El ciclo *Red-Green-Refactor*, también conocido como algoritmo del TDD, se basa en:

* **Red**: Escribir un test que falle, es decir, tenemos que realizar el test antes de escribir la implementación. Normalmente se suelen utilizar test unitarios, aunque en algunos contextos puede tener sentido hacer TDD con test de integración.
* **Green**: Una vez creado el test que falla, implementaremos el mínimo código necesario para que el test pase.
* **Refactor**: Por último, tras conseguir que nuestro código pase el test, debemos examinarlo para ver si hay alguna mejora que podamos realizar.
* Una vez que hemos cerrado el ciclo, empezamos de nuevo con el siguiente requisito.

Esta forma de programar ofrece dos beneficios principales. El primero y más obvio es que obtenemos un código con una buena cobertura de test.

El segundo beneficio es que escribir primero las pruebas nos ayuda a diseñar la API que va a tener nuestro componente, ya que nos obliga a pensar en cómo queremos utilizarlo. Esto suele acabar derivando en componentes con responsabilidades bien definidas y bajo acoplamiento.

**Planificación y programación de proyectos**

**PROYECTO**

Un proyecto, se puede considerar como la sucesión de un conjunto de tareas interrelacionadas que deben ejecutarse en un orden determinado y con el fin de alcanzar un objetivo.

**Planeamiento • Programación • Control**

Cuando se emprende la realización de un proyecto se reconoce en su desarrollo tres etapas bien diferenciadas:

El planeamiento establece qué debe hacerse y en qué secuencia.

La programación determina cuándo debe hacerse, esto es, acota en el tiempo lo planeado.

El control se encarga de verificar si se cumple con lo planeado y lo programado anteriormente.

**DIAGRAMA DE GANTT**

Es una de las técnicas más simples utilizadas en la administración de proyectos.

Consiste en representar las tareas por medio de barras, cuyas longitudes son proporcionales a la duración de las tareas.

Permite visualizar rápidamente los elementos principales, su programación en el tiempo y, además, el progreso en cada uno de ellos.

**MÉTODO DEL CAMINO CRITICO (CPM)**

El método del camino critico es una opción y complemento al Gantt ya que Gantt no permite determinar el impacto del atraso de una actividad y en proyectos complejos es necesaria mayor información.

1)Armado de la malla Establezco, mediante un gráfico, las precedencias de las tareas.

2) La Fecha temprana de un nodo determina el momento a partir del cual pueden comenzarse las tareas que nacen de ese nodo.

3) La Fecha tardía de un nodo determina hasta cuándo puede retrasarse una tarea sin alterar la fecha de finalización del proyecto**.**

4) El Margen Total, para cada tarea, establece la flexibilidad que tiene esa tarea con relación al retraso en su comienzo.

Camino crítico = margen total = 0

**Metodologías ágiles**

Las metodologías ágiles de desarrollo están especialmente indicadas en proyectos con requisitos poco definidos o cambiantes.

**Ventajas**

* Capacidad de respuesta a cambios de requisitos a lo largo del desarrollo.
* Entrega continua y en plazos breves de software funcional.
* Trabajo conjunto entre el cliente y el equipo de desarrollo.
* Importancia de la simplicidad, eliminado el trabajo innecesario.
* Atención continua a la excelencia técnica y al buen diseño.
* Mejora continua de los procesos y el equipo de desarrollo.

**Scrum**

Scrum es un proceso ágil que nos permite centrarnos en ofrecer el más alto valor de negocio en el menor tiempo. Nos permite rápidamente y en repetidas ocasiones inspeccionar software real de trabajo (cada dos semanas o un mes).

El negocio fija las prioridades. Los equipos se auto organizan a fin de determinar la mejor manera de entregar las funcionalidades de más alta prioridad. Cada dos semanas o un mes, cualquiera puede ver el software real funcionando y decidir si liberarlo o seguir mejorándolo en otro sprint.

Es un método adaptativo de gestión de proyectos que se basa en los principios ágiles:

* Colaboración estrecha con el cliente.
* Predisposición y respuesta al cambio.
* Prefiere el conocimiento tácito de las personas al explícito de los procesos.
* Desarrollo incremental con entregas funcionales frecuentes.
* Motivación y responsabilidad de los equipos por la autogestión, autoorganización y compromiso.
* Simplicidad. Supresión de artefactos innecesarios en la gestión del proyecto.
* Conjunto de prácticas y roles.
* Posee documentos y artefactos específicos.
* Equipos autoorganizados.
* Gestión de la expectativa del cliente (cambios).
* Mejoras:
* Productividad.
* Calidad.
* Motivación.

**SCRUM FRAMEWORK / Arquitectura:**

**Sprints**

* En Scrum los proyectos avanzan en una serie de “Sprints”.
  + Análogo a las iteraciones.
* La duración típica es 2–4 semanas o a lo sumo un mes calendario.
* La duración constante conduce a un mejor ritmo.
* El producto es diseñado, codificado y testeado durante el Sprint.

**Roles**

* **Product owner**:
  + Define las funcionalidades del producto.
  + Decide sobre las fechas y contenidos de los releases.
  + Es responsable por la rentabilidad del producto (ROI).
  + Prioriza funcionalidades de acuerdo al valor del mercado/negocio.
  + Ajusta funcionalidades y prioridades en cada iteración si es necesario.
  + Acepta o rechaza los resultados del trabajo del equipo.
* **Scrum master**:
  + Representa a la gestión del proyecto.
  + Responsable de promover los valores y prácticas de Scrum.
  + Remueve impedimentos.
  + Se asegura de que el equipo es completamente funcional y productivo.
  + Permite la estrecha cooperación en todos los roles y funciones.
  + Escudo del equipo de interferencias externas.
* **Team:**
  + Típicamente de 5 a 9 personas
  + Multifuncional: Programadores, testers, analistas, diseñadores, etc.
  + Los miembros deben ser full-time: puede haber excepciones (Ej.: Infraestructura, SCM, etc.)
  + Los equipos son autoorganizativos: Idealmente, no existen títulos, pero a veces se utilizan de acuerdo a la organización
  + Solo puede haber cambio de miembros entre los sprints.

**Reuniones**

* Daily scrum:
  + Parámetros:
    - Diario.
    - Dura 15 minutos.
    - Los miembros están parados.
  + No para la solución de problemas:
    - Todo el mundo está invitado.
    - Sólo los miembros del equipo, ScrumMaster y Product Owner, pueden hablar.
    - Ayuda a evitar otras reuniones innecesarias.
  + Todos responden 3 preguntas:
    - ¿Qué hiciste ayer?
    - ¿Qué vas a hacer hoy?
    - ¿Hay obstáculos en tu camino?
  + **No** es dar un status report al Scrum Master.
  + Se trata de compromisos delante de pares.

**BASES DE DATOS – CONCEPTOS**

Conjunto de datos relacionados entre sí con un significado implícito

• Representa un aspecto del mundo real

• No es una colección aleatoria

• Se diseña para un propósito especifico

**SISTEMA GESTOR DE BASE DE DATOS (SGBD, DBMS)**

Software de propósito general para definir, construir y manipular una base de datos, permite:

• Definir: especificar estructuras de datos, tipos y restricciones.

• Construir: guardar datos en algún medio de almacenamiento controlado por el SGBD.

• Manipular: realizar consultas y actualizaciones.

**Base de Datos** 🡪 Conjunto de datos relacionados entre sí con un significado implícito.

**+**

**Gestor de Base de Datos** 🡪 Software de propósito general para definir, construir y manipular una base de datos.

**=**

**Sistema de Base de Datos** 🡪 Conjunto de datos y software.

**DIFERENCIA ENTRE ARCHIVOS Y BD**

Archivos: cada usuario define e implementa los archivos para una aplicación específica.

Base de datos: único almacenamiento, varios usuarios.

Archivos: la definición de datos es parte de los programas.

Base de datos: la definición y descripción completa de la BD está en un catálogo que es utilizado por el SGBD y los usuarios.

Archivos: la estructura de los archivos viene integrada en los programas de acceso. Cualquier modificación de la estructura, requiere modificación de los programas.

Base de datos: los programas de acceso se escriben independientemente de las estructuras.

Archivos: cada usuario mantiene sus propios archivos.

Base de datos: la vista de los diferentes usuarios se integra durante el diseño de la base de datos.

**CARACTERÍSTICAS DISTINTIVAS DE LAS BD**

Abstracción de datos: ofrece una representación conceptual, no incluyen detalles de almacenamiento.

Manejo de múltiples vistas de usuario: cada usuario puede tener una vista diferente de la base de datos.

Transacciones multiusuarios: varios usuarios tienen acceso simultaneo a la base de datos.

Múltiples interfaces para diferentes usuarios.

Restricciones de integridad.

Respaldo y recuperación de datos.

**ACTORES**

* Administrador de la base de datos: supervisa y controla recursos.
  + primarios: base de datos
  + secundarios: SGBD y programas
* Diseñador de la base de datos
* Usuario Final

**ARQUITECTURA DE UNA BD**

Nivel Externo 🡪 Distintas visiones del usuario.

Nivel Conceptual 🡪 Estructura conceptual de la BD.

Nivel Físico 🡪 Estructura física de almacenamiento.

Independencia lógica de datos 🡪 Posibilidad de modifica el nivel conceptual sin modificar los niveles externos.

Independencia física de datos 🡪 Posibilidad de modificar el nivel físico sin modificar el conceptual.

**BUENAS PRÁCTICAS**

No hay un estándar. La comunidad establece buenas prácticas.

Mantenga coherencia en el modelo.

Si usa singular, úselo en todo el modelo.

Gráfico, Diagrama, Gráfico de cajas y bigotes

Descripción generada automáticamenteSi usa plural, sea consistente en todo el modelo.

**NOTACIÓN MARTIN (O PATA DE GALLO)**

Ventajas de notación Martin (Pata de gallo):

Ampliamente usada.

Muestra al mismo tiempo opcionalidad y cardinalidad máxima.

**SQL**

SQL (lenguaje de consulta estructurado).

LDD (lenguaje de definición de datos) permite crear las estructuras, modificarlas y borrarlas.

LMD (lenguaje de manipulación de datos) permite hacer consultas, altas, bajas y modificaciones del contenido de las tablas.