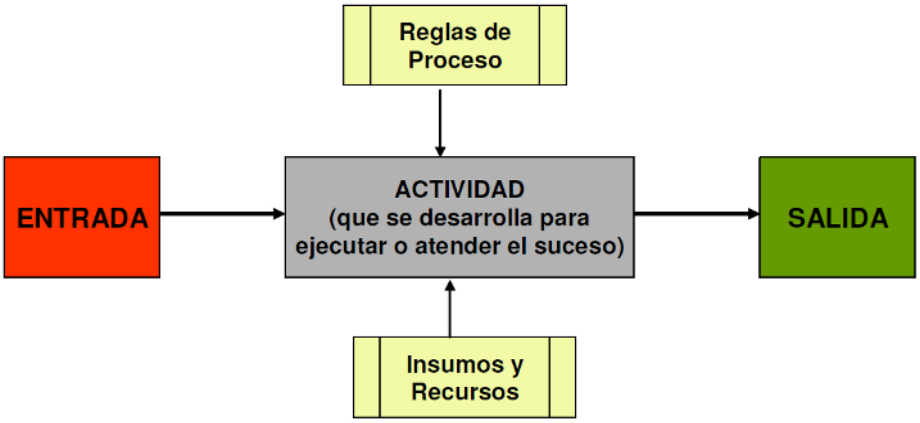
Resumen PPT MDS I - (Segundo Parcial)

Metodologías De Desarrollo De Sistemas I (Universidad Abierta Interamericana)

5.1. Procesos de negocios

Objetivos

Habilidades y competencias que desarrolla la asignatura



Los Sistemas de Información

¿Qué Rol cumplen los SI en las organizaciones?

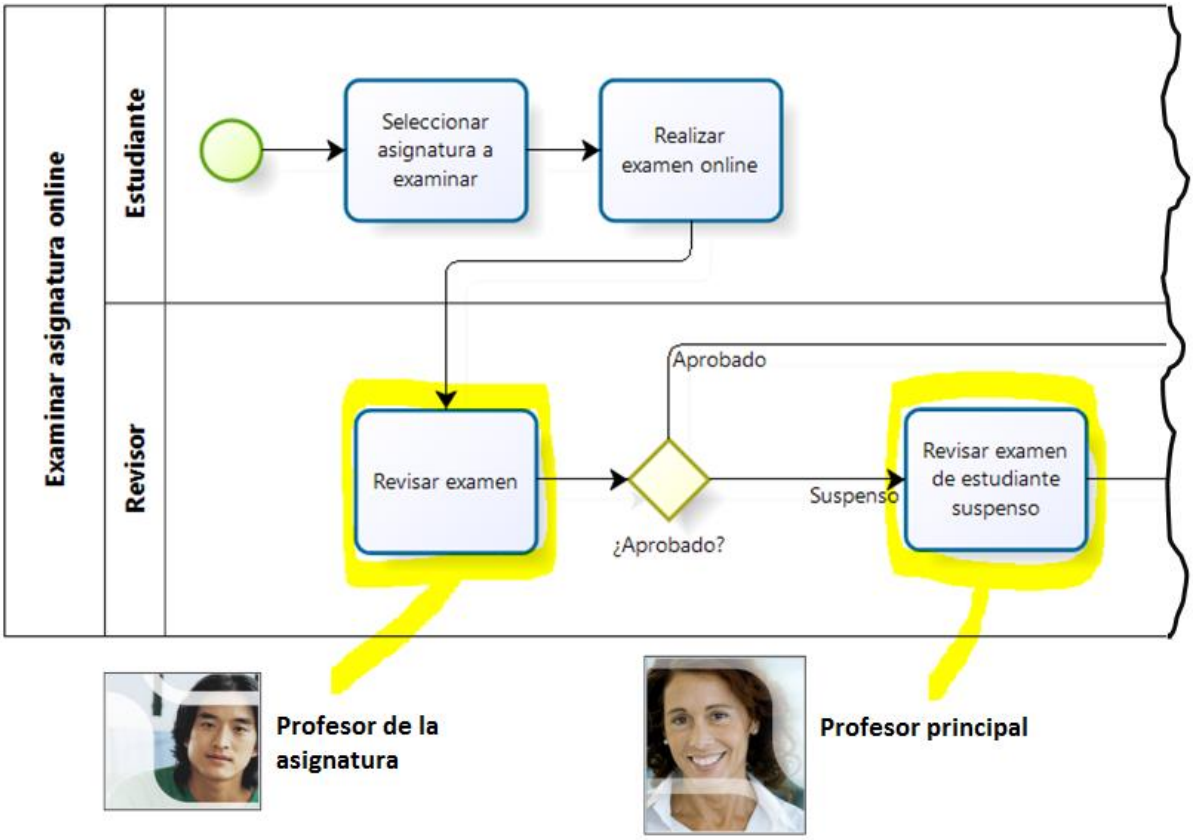
El objetivo de los SI es ayudar al desempeño de las actividades en todos los niveles de la organización.

¿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.

Procesos de Negocios

Algunas Definiciones…



¿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 "

Sistemas de Información Gerencial. Editorial Prentice Hall, 2004. K.C. Laudon, J.P. Laudon.

"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"

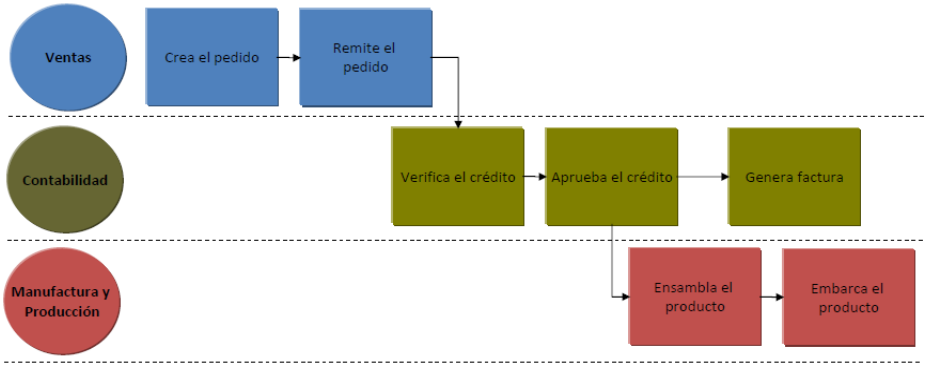
Process Innovation: Reegineering work through Information Tecnology. Hardvard Business School, 1993. Davenport, Thomas

"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",

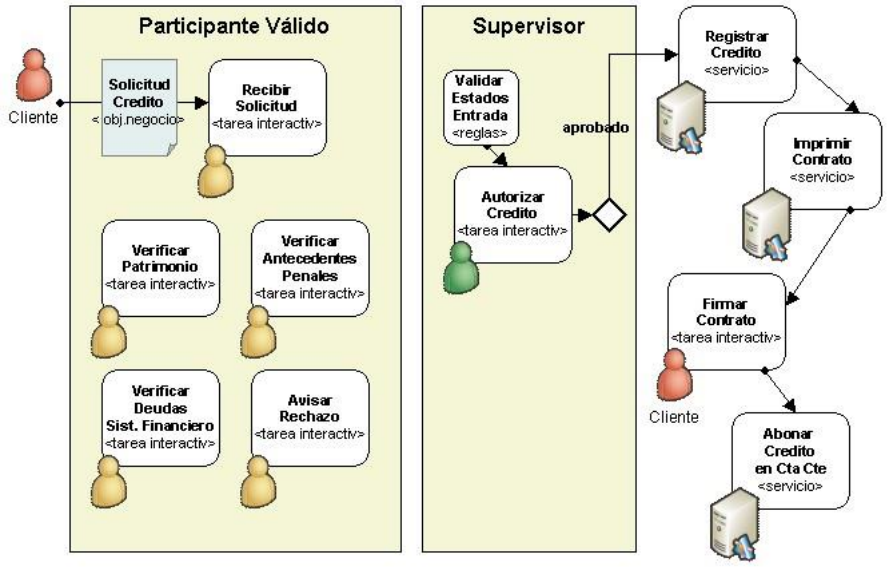
Business Processes: Modelling and Analysis for Re-Engineering and Improvement. Martyn A. Ould

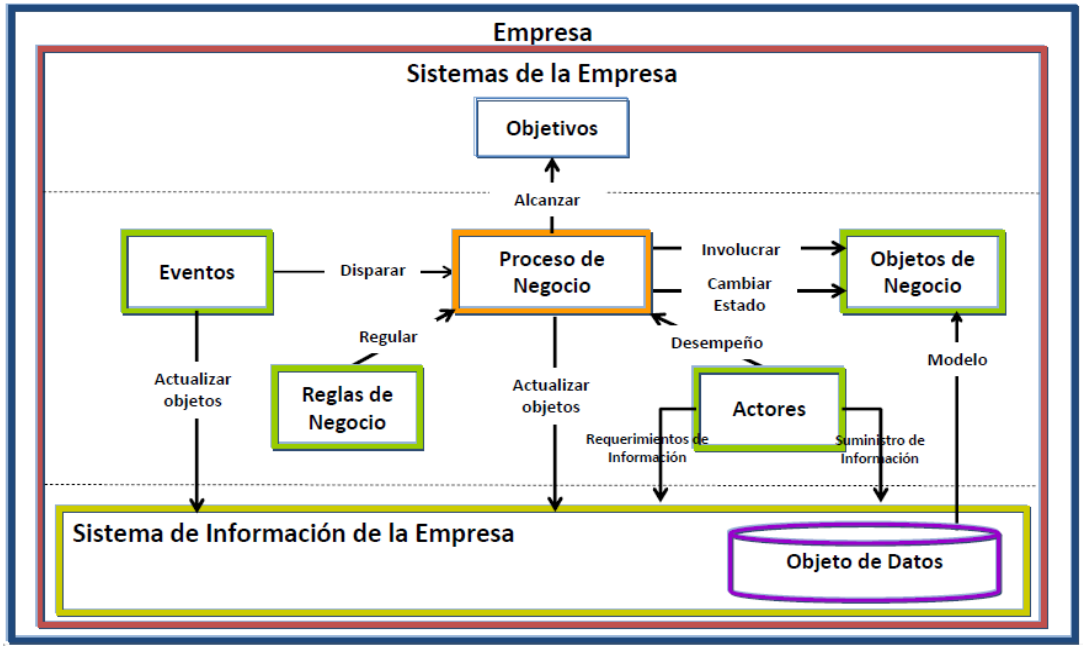
BP: Procesos de Negocios
¿Qué es un proceso de Negocio?

- Diseño y/o Desarrollo de un producto.
- Generar y completar un pedido.
- Provisión de un servicio.
- Contratar un empleado

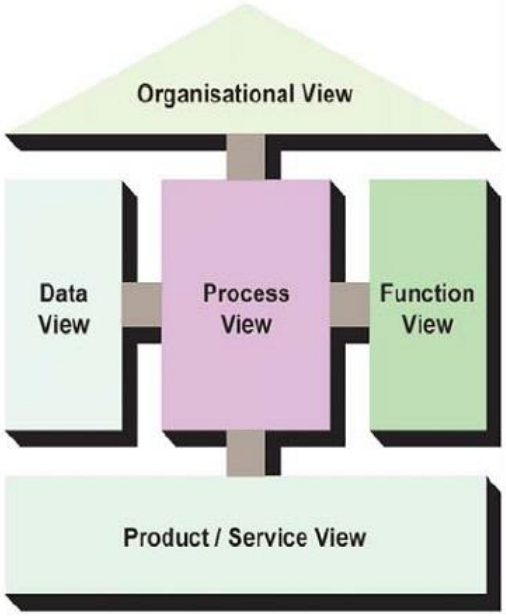


Ejemplo: Como interactúan los recursos y los procesos en un proceso de pago en una entidad crediticia.

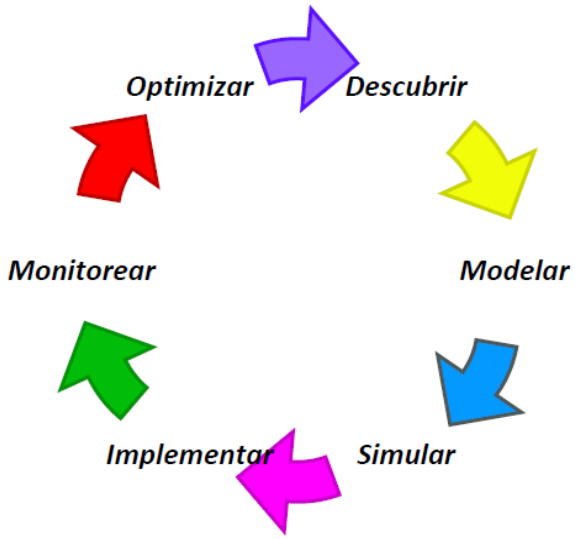




BP: Procesos de Negocios - Puntos de Vista



BP: Procesos de Negocios - Ciclo de Vida



BP: Procesos de Negocios – 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. Representación Narrativa

Ejemplo: "depositar un cheque directamente en la ventanilla de un banco".

El cliente una vez que decidió hacer el depósito, debe dirigirse al banco, buscar una comprobante de depósito y preparar el depósito. Para ello debe ingresar los datos del depósito según exigencias del formulario. Una vez lleno, adjunta el dinero y se dirige a la caja y entrega el depósito. El cajero toma el depósito, ingresa el número de la cuenta corriente y si es correcto acepta el depósito. Para ello cuenta el dinero y valida con lo registrado en el documento. Si es válido ingresa el resto de los datos del depósito y el sistema actualiza el saldo. Terminado aquello, timbra los formularios, una copia se la entrega al depositante y el resto, junto al dinero lo acumula en un caja. Hecho esto termina el proceso.

Forma de Representar - Notaciones de Modelado

- Redes de Petri Carl Adam Petri
- Diagrama Entidad Relación
- Diagrama de flujo de Datos
- Diagramas de Actividades de UML OMG
- SPEM (Software Process Engineering Meta-Model) OMG
- BPMN (Business Process Modeling Notation) OMG
- XPDL (XML Process Definition Language) WfMC
- IDEF (Integration DEFinition) U.S. Air Force

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 a lo largo del proceso.

Modelado del Negocio - Diagramas de Flujo
Representación abstracta (gráfica) de los procesos de una organización, que muestran principalmente cómo y por quiénes son llevadas a cabo las actividades que generan valor para la organización.

Muestran también:

- Los actores involucrados en los procesos.
- Cuáles son las actividades operativas distinguibles.
- Qué actividades son ejecutables y por quién.
- Cuáles son las entradas y salidas de actividades.
- Cuál es la secuencia de las actividades.
- Los recursos consumidos.
- Los eventos que dirigen el proceso.

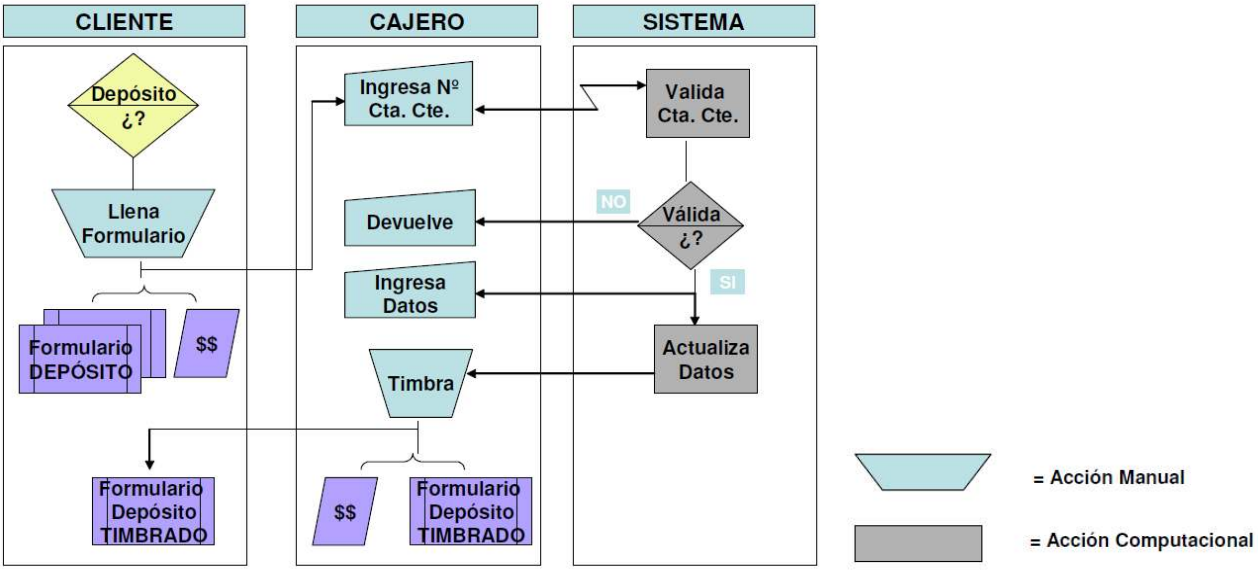
Modelado del Negocio - Ventajas y Desventajas Diagramas de Flujo

- Mejor Entendimiento del sistema y la empresa
- Proporcionar mejores soluciones a la empresa

- Más rápido y entendible
- Mejores Resultados

Para la organización: Efectividad, Eficiencia, Consistencia, Productividad, Ahorro, Calidad.

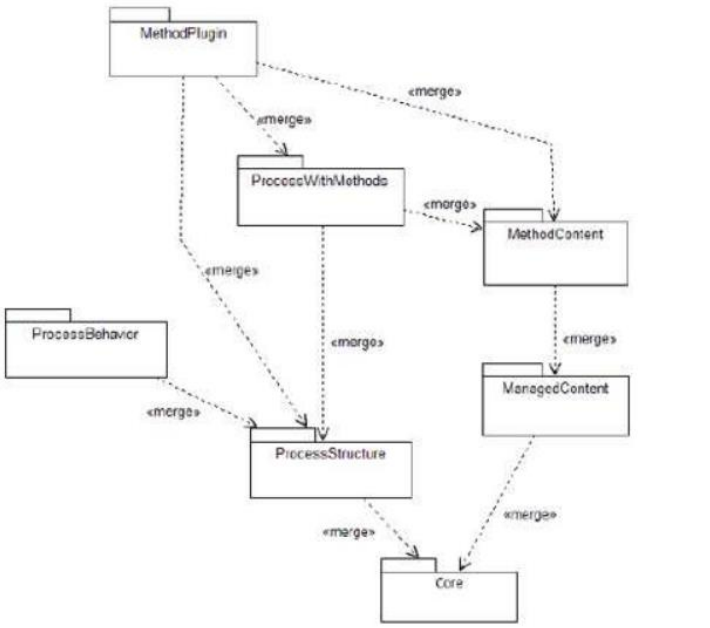
Forma de Representar - Diagramas de Flujos



Herramientas de Modelado (SPEM)

SPEM (Software Process Engineering Metamodel)

Es un estándar de la OMG cuyo objetivo principal es proporcionar un marco formal para la definición de procesos de desarrollo de sistemas y de software, así como para la definición y descripción de todos los elementos que los componen.



Estos paquetes nos van a proporcionar las siguientes capacidades:

Core: Contiene todas las clases y abstracciones que constituyen la base para el resto de los paquetes del metamodelo.

Process Structure: Contiene las clases necesarias para la creación de modelos de procesos.

Process Behavior: Para representar la parte dinámica de los procesos, su comportamiento.

Managed Content: Nos va a permitir dotar a nuestros procesos o sistemas de anotaciones y descripciones que no pueden ser expresadas como modelos y que por lo tanto deben ser documentadas y gestionadas como descripciones en lenguaje natural.

Method Content: Contiene los conceptos de SPEM 2.0 relacionados con los usuarios y la organización. Estos conceptos son necesarios para construir una base de conocimiento sobre desarrollo que pueda ser utilizada independientemente del proceso o proyecto específico.

Process WithMethods: Contiene los elementos necesarios para integrar los conceptos del paquete Process Structure con los conceptos y elementos del paquete Content Method.

Method Plug-In: Introduce los conceptos para diseñar, gestionar y mantener repositorios y librerías de Methods Content y Procesos.

Herramientas de Modelado (BPMN)

BPMN (Business Process Modelling Notation)

Es un estándar de la BPMI (Business Process Management Initiative), cuyo principal objetivo es: "proporcionar una notación fácilmente comprensible por todos los usuarios del negocio, desde los analistas, os desarrolladores técnicos hasta aquellos que monitorizarán y gestionarán los procesos".

Otros objetivos importantes que se plantea esta especificación son:

Crear puentes entre el diseño de los procesos de negocio y la implementación del proceso.

Que los lenguajes basados en XML para describir procesos (como BPEL4WS) tengan una notación gráfica.

Otros objetivos importantes que se plantea esta especificación son:

* Crear puentes entre el diseño de los procesos de negocio y la implementación del proceso.

* Que los lenguajes basados en XML para describir procesos (como BPEL4WS) tengan una notación gráfica.

Los modelos BPMN se expresan gráficamente mediante diagramas BPMN. Estos diagramas constan de una serie de elementos que nos van a permitir diferenciar claramente las tres secciones (o submodelos) básicos que existen en un modelo BPMN.

Son:

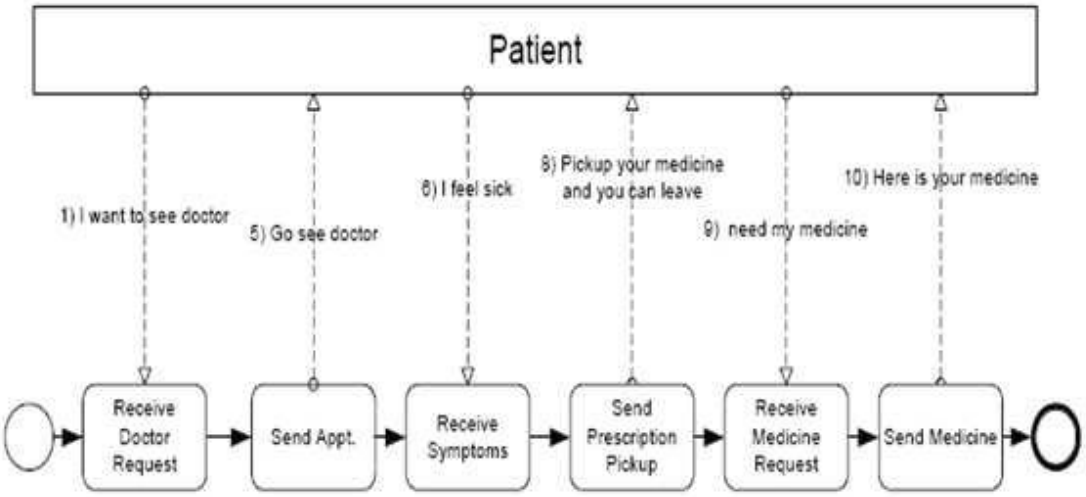
1. Procesos de negocio privados (internos).
2. Procesos abstractos (públicos).
3. Procesos de colaboración (globales).

1) Procesos de negocio privados (internos)

Son los que, dentro de una organización específica, han sido tradicionalmente llamados diagramas de flujo de trabajo o diagramas de workflow. Si usamos calles para representarlos este tipo de procesos únicamente ocuparán una calle, aunque pueda interactuar, mediante el flujo de mensajes, con otros procesos de negocio de la misma clase.

2) Procesos de negocio abstractos (públicos)

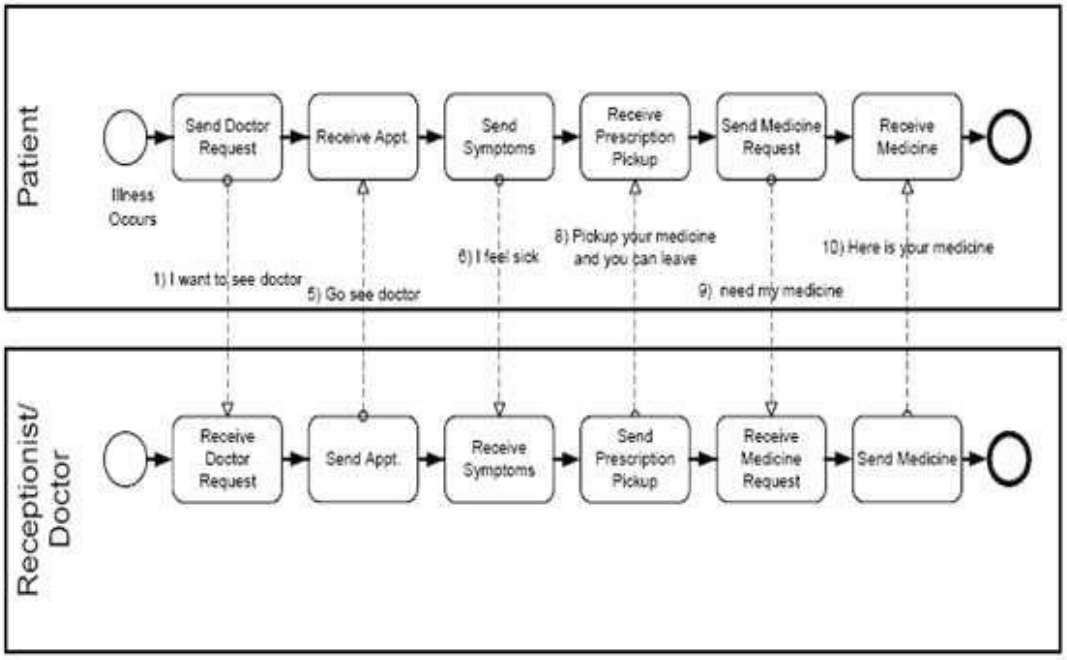
Los procesos de negocio abstractos nos sirven para representar las interacciones existentes entre un proceso de negocio privado y, o bien otro proceso de negocio o bien un participante del proceso. En este tipo de procesos únicamente se incluyen aquellas actividades que se usan para comunicar un proceso privado con el exterior, así como las correspondientes estructuras de control de flujo.



3) Procesos de colaboración (globales)

Este tipo de procesos sirven para mostrar la interacción entre distintas entidades de negocio. Estas interacciones son definidas como secuencias de actividades que representan el intercambio de mensajes entre las distintas entidades.

La colaboración se entiende como la comunicación entre dos o más procesos.



Herramientas de Modelado (BMM)

Método BMM (Business Modeling Modeling Method)

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

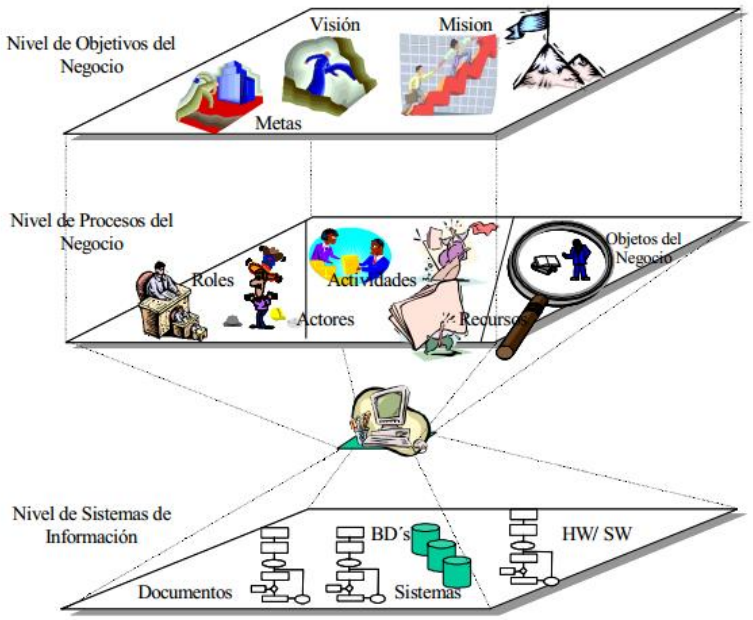
La noción de Sistemas de Negocios:

- Integra los aspectos o elementos más importantes de un negocio.
- Delimita el proceso de modelado.

Divide el Sistema de Negocios en 3 niveles:

- Objetivos
- Procesos
- Sistemas

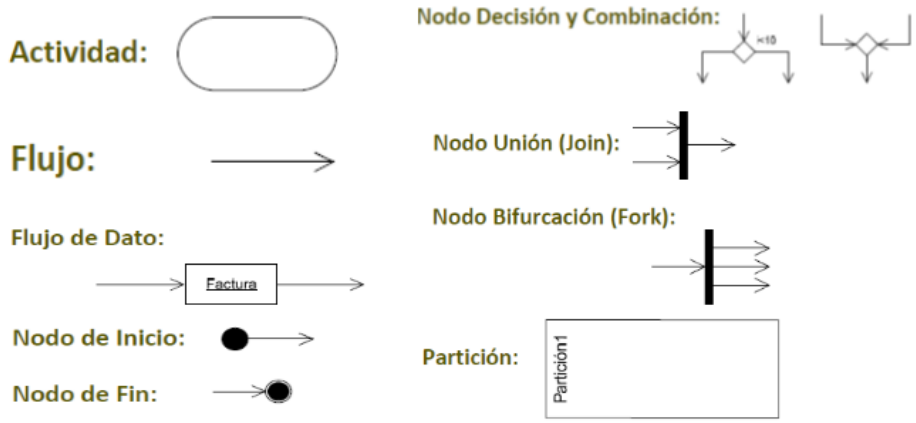
Facilita la alineación de los sistemas a los objetivos y procesos



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.

Elementos:

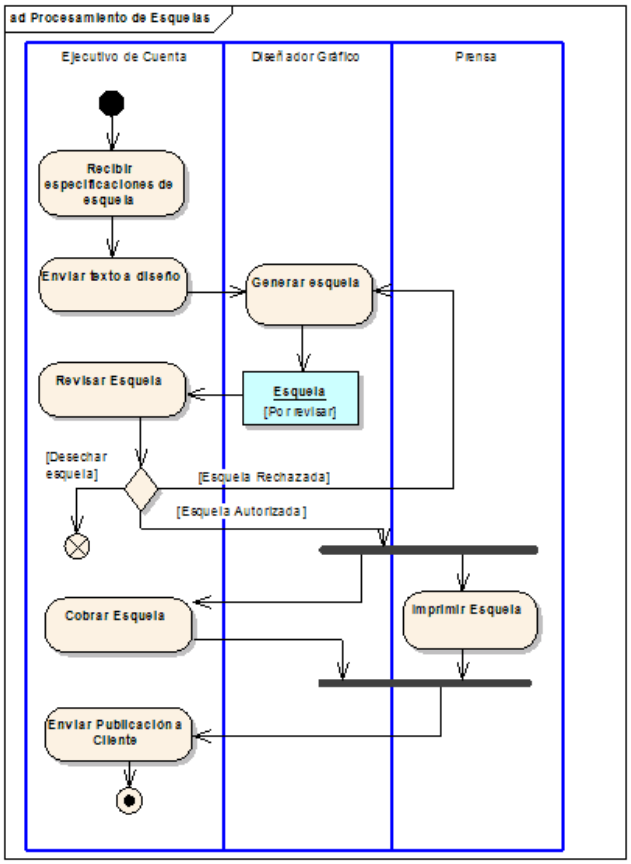


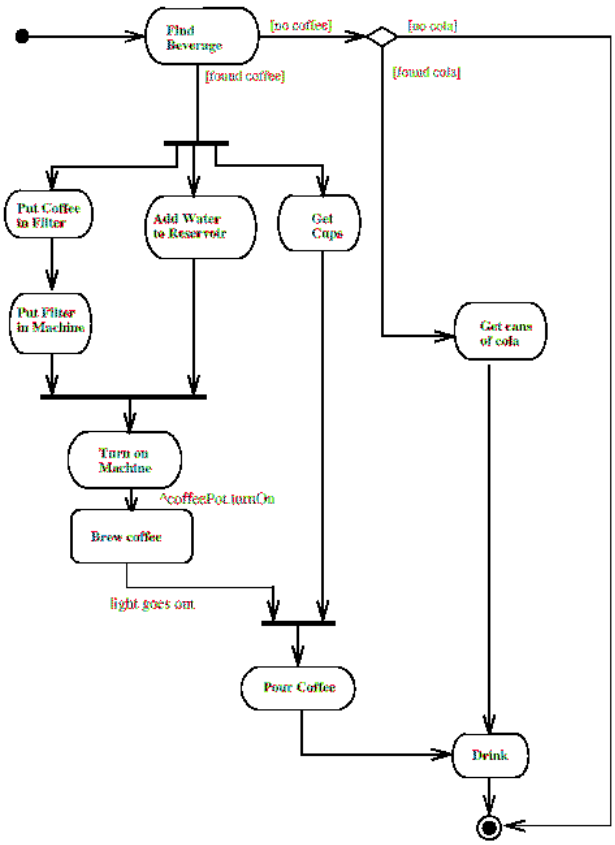
Nodo Decisión: Guían el flujo en una u otra dirección. Poseen un único flujo de entrada y varios flujos de salida.

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





5.2. Proceso de Desarrollo de Software.

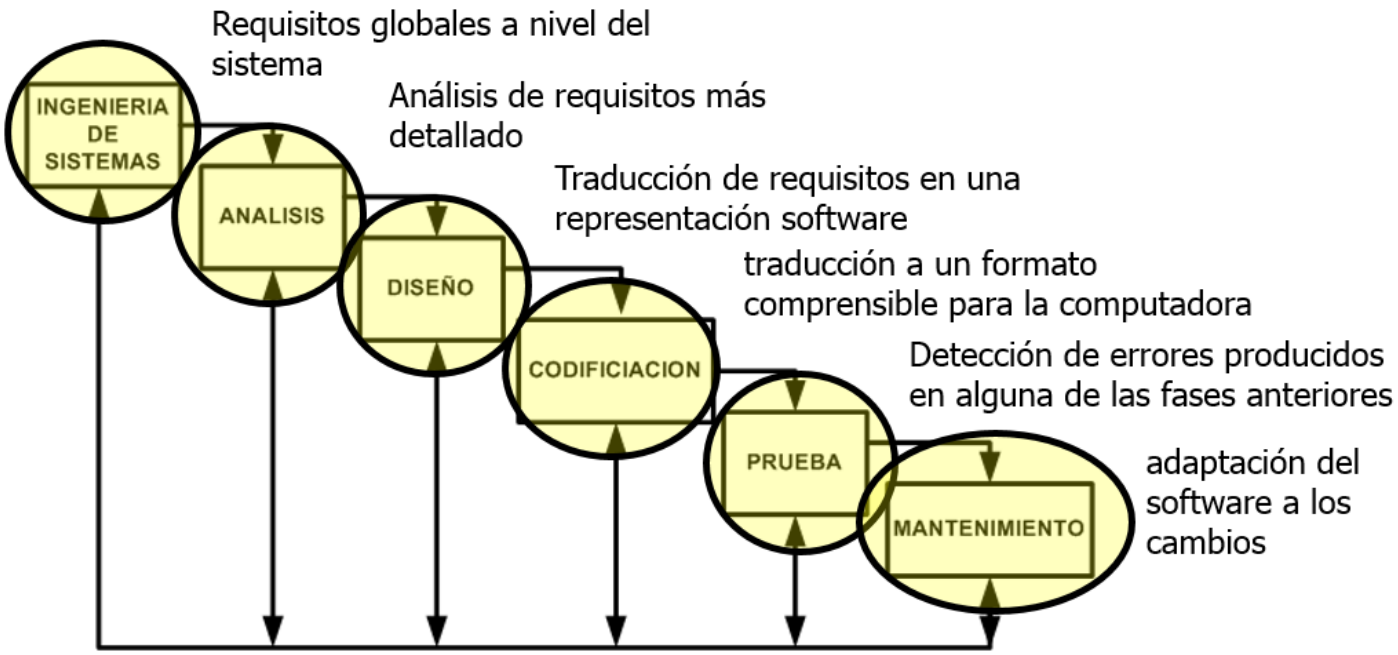
Objetivos

Habilidades y competencias que desarrolla la asignatura

PRIMERA PARTE - Ciclo de Vida
Independientemente del tipo de ciclo de vida utilizado, todos incluyen algunas o todas de estas actividades

- 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
- La elección de un modelo de ciclo de vida se realiza de acuerdo con la naturaleza del proyecto, los métodos a utilizar y los controles y entregas requeridos.

1. Ciclo de vida en cascada



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
    - Los requisitos se van aclarando y refinando a lo largo de todo el proyecto
    - Hasta que se llega a la fase final del desarrollo, esto es, la codificación, no se dispone de una versión operativa del programa.

Ámbito de aplicación del ciclo de vida en cascada

- Sistemas simples y pequeños, donde los requisitos sean fácilmente identificables
- Este modelo describe una serie de pasos genéricos que son aplicables a cualquier otro paradigma (espiral, UP)

2. 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

- El proceso comienza con la determinación de requisitos de información, que pueden ser traducidos directamente a código fuente usando un generador de código.
- 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
- 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
- El resto de las fases del ciclo de vida es igual a las del modelo del ciclo de vida en cascada

Problemas del Ciclo de vida 4GL

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

Ámbito de aplicación del Ciclo de vida 4GL

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

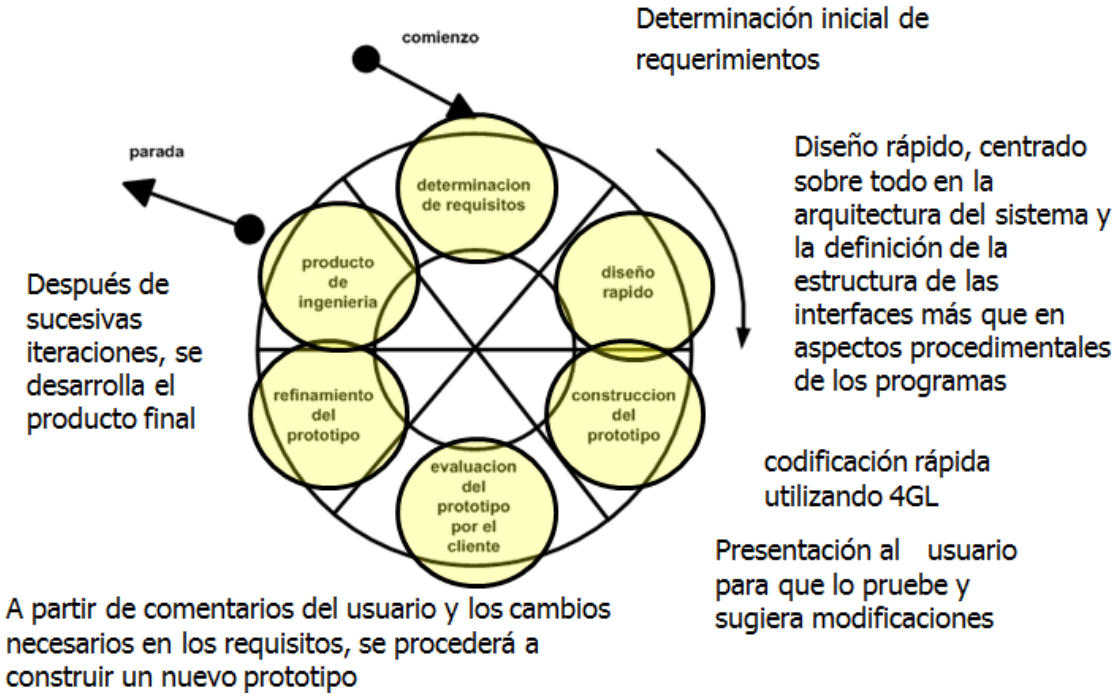
- La mayoría de las herramientas de cuarta generación están orientadas a la generación de informes a partir de grandes bases de datos
- Esto es debido a que la mejor prestación de estos lenguajes está relacionada con la creación de interfaces hombre - máquina y con las consultas a bases de datos.

3. Prototipos

- En el ciclo de vida en cascada se dificulta la obtención clara de todos los requisitos del sistema al inicio del proyecto
- Un modelo de ciclo de vida basado en la construcción de prototipos puede disminuir estos inconvenientes
- La construcción de un prototipo comienza con la realización de un modelo del sistema, a partir de los requisitos que se conocen
- No es necesario realizar una definición inicial completa de los requisitos del usuario

ES UN PROCESO ITERATIVO E INCREMENTAL

Prototipos – construcción



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

- En general, 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
- El prototipo provee una retroalimentación para evaluar y desarrollar nuevos requerimientos
- 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.

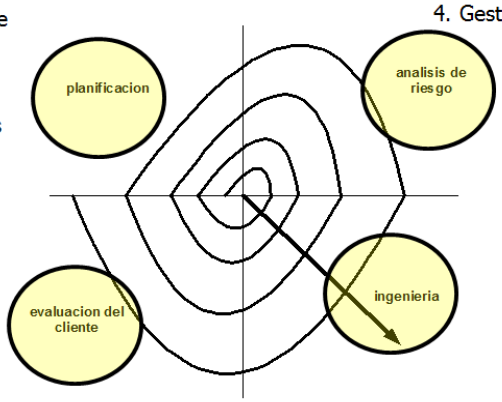
4. Ciclo de vida en espiral

- Importantes proyectos de sistemas fallaron porque los riesgos del proyecto se despreciaron sin estar nadie preparado para algún imprevisto
- Barry Boehm reconoció esto y trató de incorporar el factor de “riesgo del proyecto” al modelo de ciclo de vida, agregándoselo a las mejores características de los modelos de Cascada y de Prototipo
- El modelo en espiral proporciona un modelo evolutivo para el desarrollo de sistemas de software complejos

Ciclo de vida en espiral - construcción

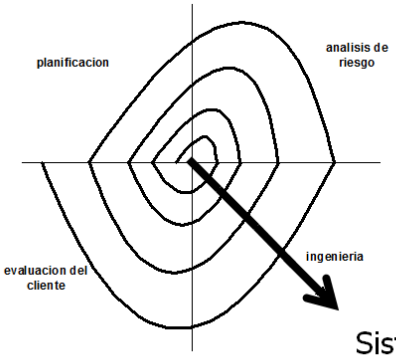
1. Identificación de los riesgos
2. Estimación de los riesgos
3. Evaluación de los riesgos
4. Gestión de riesgos

determinación de los objetivos del proyecto, las posibles alternativas y las restricciones.



valoración, por parte del cliente, de los resultados de la ingeniería

Desarrollo del sistema o un prototipo del mismo.



Con cada iteración se construyen sucesivas versiones del software, cada vez más completas. Aumenta la duración de las operaciones del cuadrante de ingeniería, obteniéndose, al final, el sistema completo

Sistema completo

Problemas Ciclo de vida en espiral

- Falta de un proceso de guía explícito para determinar objetivos, limitaciones y alternativas.
- La determinación del riesgo no es una tarea fácil
- Es necesaria mucha experiencia en proyectos de software

Ámbito de aplicación Ciclo de vida en espiral

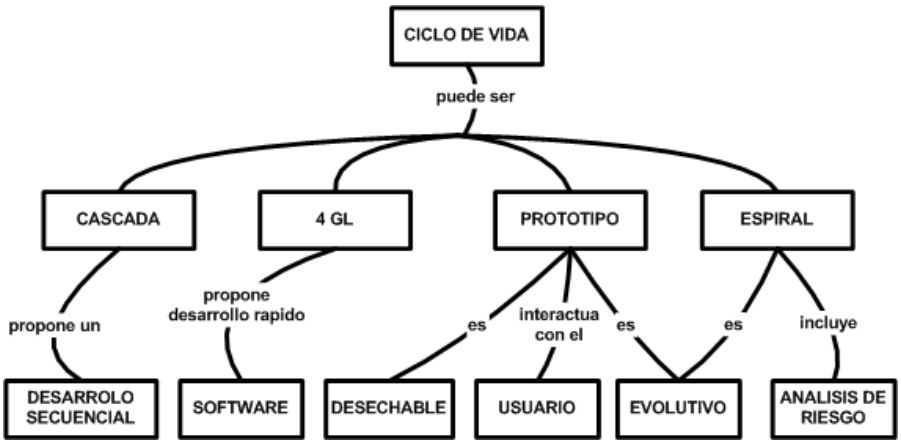
- Evita las dificultades de los modelos existentes a través de un proceso conducido por el riesgo, intentando eliminar errores en las fases tempranas.
- Se adapta bien a proyectos complejos, dinámicos e innovadores

El mantenimiento en el software

- Terminada la fase de pruebas, el software se entrega al cliente y comienza la vida útil del mismo
- El software sufrirá cambios a lo largo de su vida útil. Estos cambios pueden ser debidos a variadas causas.
    1. Durante la utilización, el cliente puede detectar errores en el software, estos se denominan errores latentes
    2. Cuando se producen cambios en alguno de los componentes del sistema informático, por ejemplo cambios en la computadora, en el sistema operativo o en los periféricos, se debe adaptar el software a ellos.
    3. El cliente habitualmente requiere modificaciones funcionales, normalmente ampliaciones, que no fueron contempladas inicialmente en el proyecto.

- En cualquier caso, el mantenimiento supone volver atrás en el ciclo de vida, a las etapas de codificación, diseño o análisis dependiendo de la magnitud del cambio encarado.

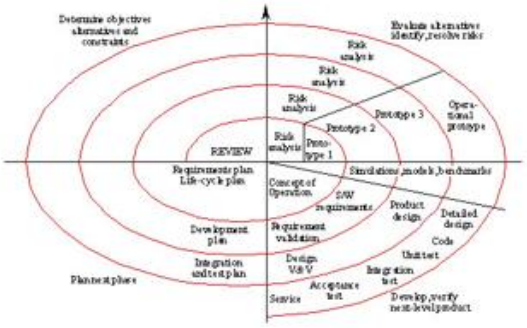
Ciclo de vida - resumen



SEGUNDA PARTE - Proceso de Unificado de desarrollo

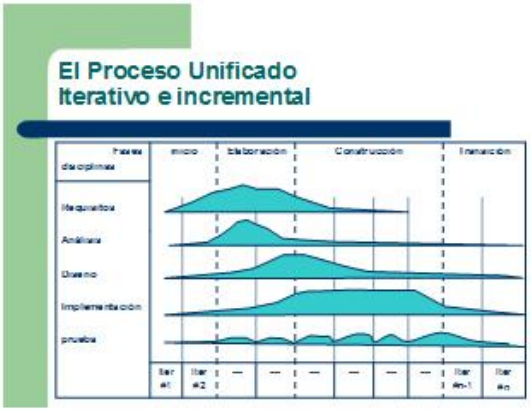
UML no es un proceso... UML es una notación

Por lo tanto, precisa de un proceso de desarrollo (ciclo de vida) que especifique la secuencia de actividades que deben realizarse.



Ciclo de vida en cascada

Ciclo de vida en espiral



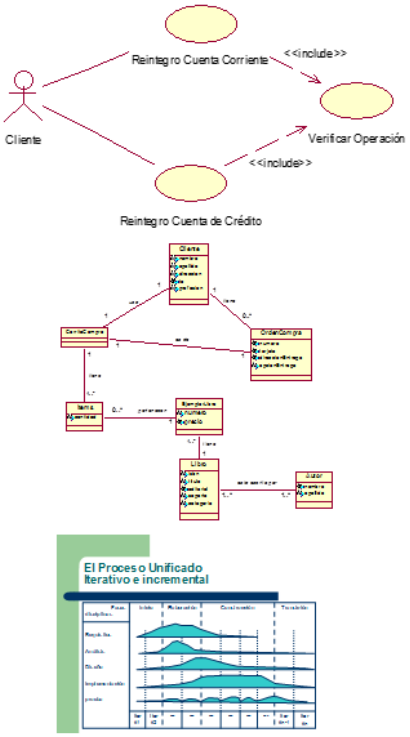
Proceso Unificado

Proceso de Desarrollo de Software

Conjunto de actividades para transformar los requisitos del usuario en un sistema software

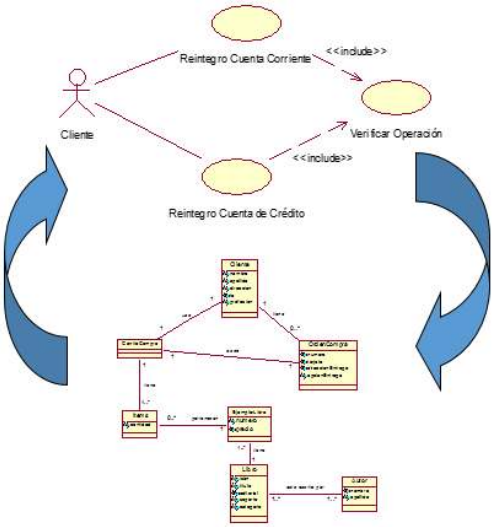
Proceso Unificado (UP)

- *Dirigido por Casos de Uso*
- *Centrado en la Arquitectura*
- *Iterativo e Incremental*

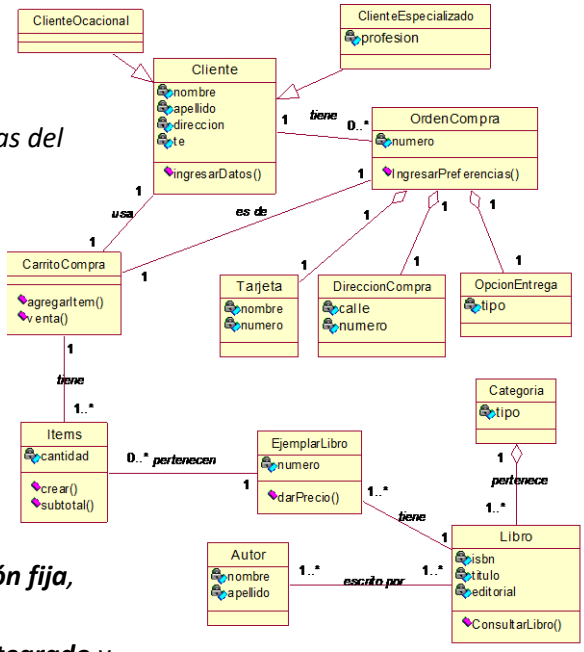


Dirigido por Casos de Uso

- 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 que llevan a cabo los casos de uso



Centrado en la Arquitectura



- *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

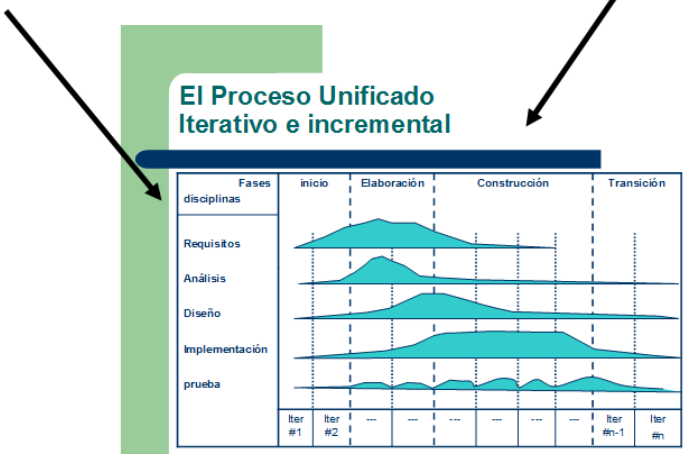
Desarrollo iterativo:

- *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 incrementalmente 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 amplia el sistema incrementalmente*
- *La salida de una iteración NO es un prototipo desechable, es un subconjunto de calidad del sistema final.*

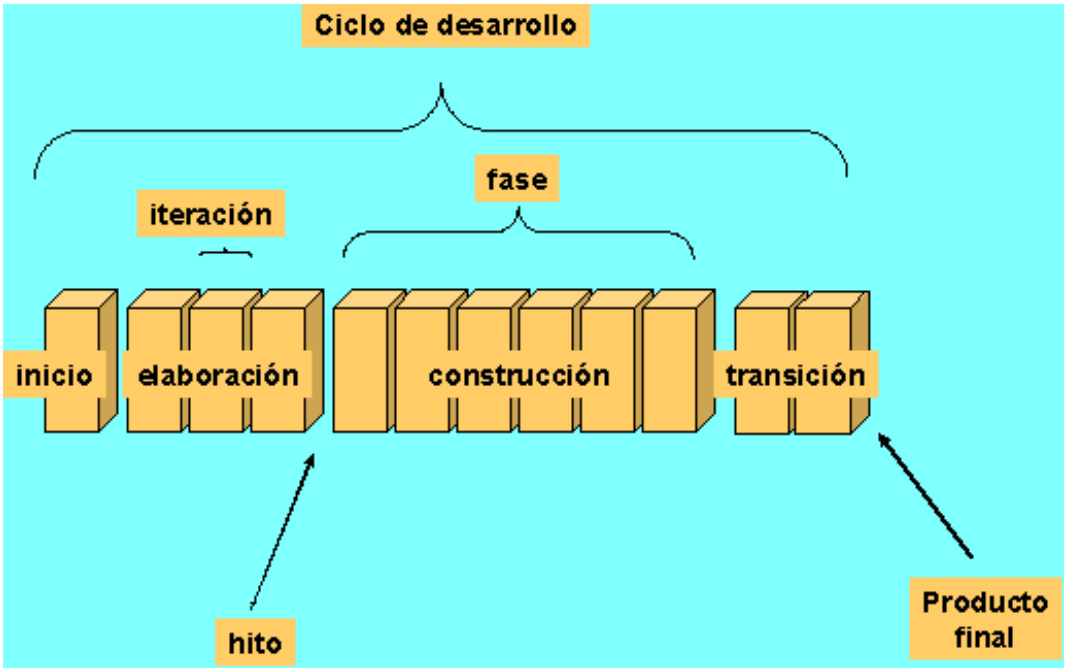
Disciplinas y fases

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

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



Ciclo de desarrollo

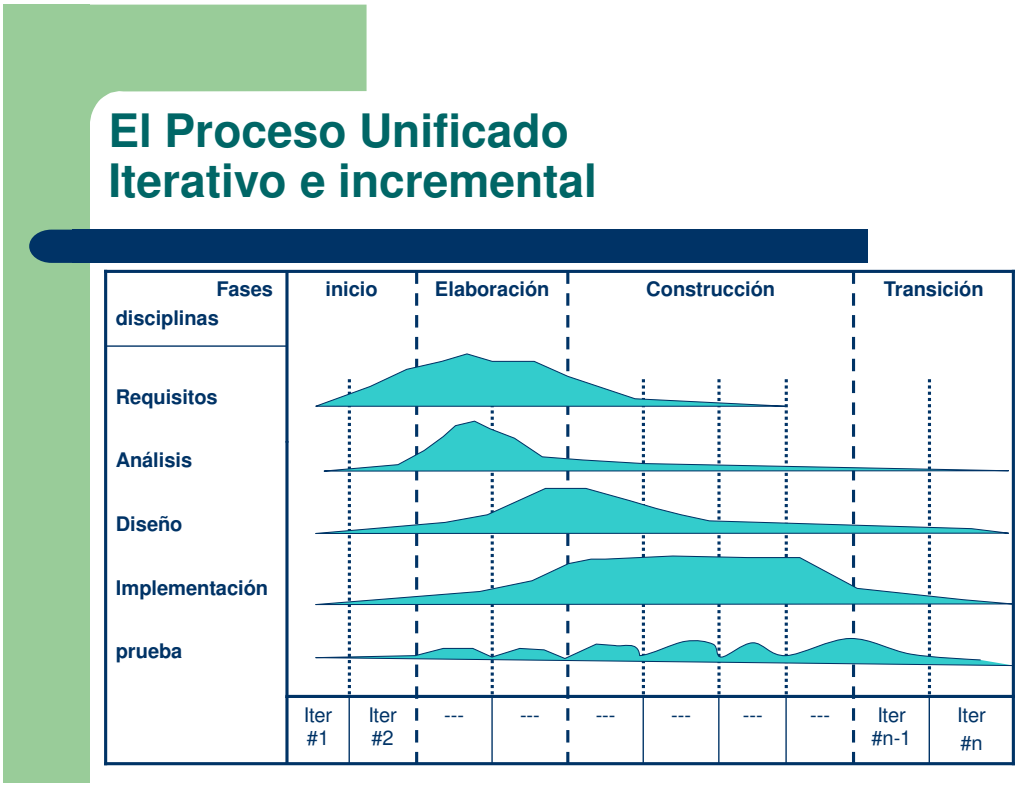


Hito: *punto en donde han de tomarse decisiones importantes*

Producto final: *Sistema en ambiente de producción*

Fases del Proceso Unificado

- ***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)*

No *se entendió la fase de inicio si...*

- *La duración es mayor a unas pocas semanas*
- *Se intenta definir la mayoría de los requisitos*
- *Se espera que los planes y la estimación sea fiable*
- *Se define la arquitectura*
- *No se identifican la mayoría de de los nombres de los casos de uso y los actores*
- *Se escribieron todos los casos de uso en detalle*

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

No *se entendió la fase de elaboración si...*

- *Sólo comprende una iteración*
- *La mayoría de los requisitos se definieron antes de la elaboración*
- ***NO*** *hay programación de código ejecutable*
- *Se intenta llevar a cabo un diseño completo antes de la codificación*
- *Los usuarios no se involucran en la evaluación*

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*

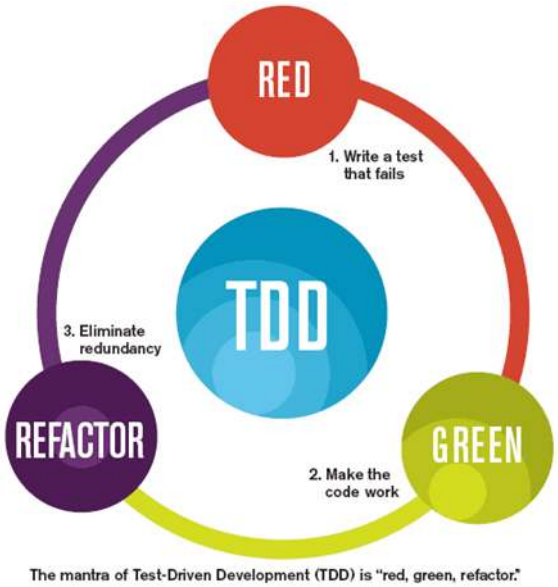
Actividades

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

Sugerencias
- Recuerde que cada fase, si bien pasa por todas las disciplinas (análisis, diseño, etc.) pone énfasis en algunas de ellas más que en otras
- Valore la importancia de los casos de uso (unidad 3.1). A partir de ellos se crean todos los demás modelos del sistema
- Tenga en cuenta que después de cada iteración (que corresponde aproximadamente a un ciclo de vida en cascada) se va definiendo un sistema ejecutable, pero incompleto, que debe corroborar con el usuario
- Recuerde que inicio **no** es igual a requisitos, elaboración **no** es igual a análisis y construcción **no** es igual a diseño, etc.
- Recuerde que cada iteración tiene un duración fija de 2 a 6 semanas aproximadamente dependiendo de la magnitud del proyecto

TERCERA PARTE - Metodologías ágiles - Desarrollo guiado por Test

¿Cómo funciona? - Ciclo de Vida R-G-R (Red-Green-Refactor)



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

- Escritura de código (Baby Steps)
- Obtener ROJO algo mal
- Escritura de código
- Obtener VERDE ok
- Refactor corrijo

Este es el ciclo de vida del desarrollo guiado por test, Rojo-Verde-Refactor y se debería repetir esto una y otra vez hasta que tengamos las características funcionales y completas.

Características
- La obtención de un buen resultado depende de la calidad de las pruebas.
- No solo se basa en pruebas, depende también del refactor del código (mejor código, limpio, mantenible).
- El refactor solo se aplica si es necesario.
- El test es escrito por el desarrollador. (Visión desde su punto de vista)
- El foco principal de TDD es testar la funcionalidad de bajo nivel

Ventajas
¿Por qué es mejor hacer las pruebas antes qué el código?

- Se condicionan las pruebas a lo implementado: pueden obviarse casos de prueba.
- Se realiza un profundo análisis funcional y se refinan los requisitos.
- Se implementa solamente código necesario:
  - Baja redundancia
  - Sin código muerto ("por si acaso")
  - Código limpio

Buenas prácticas

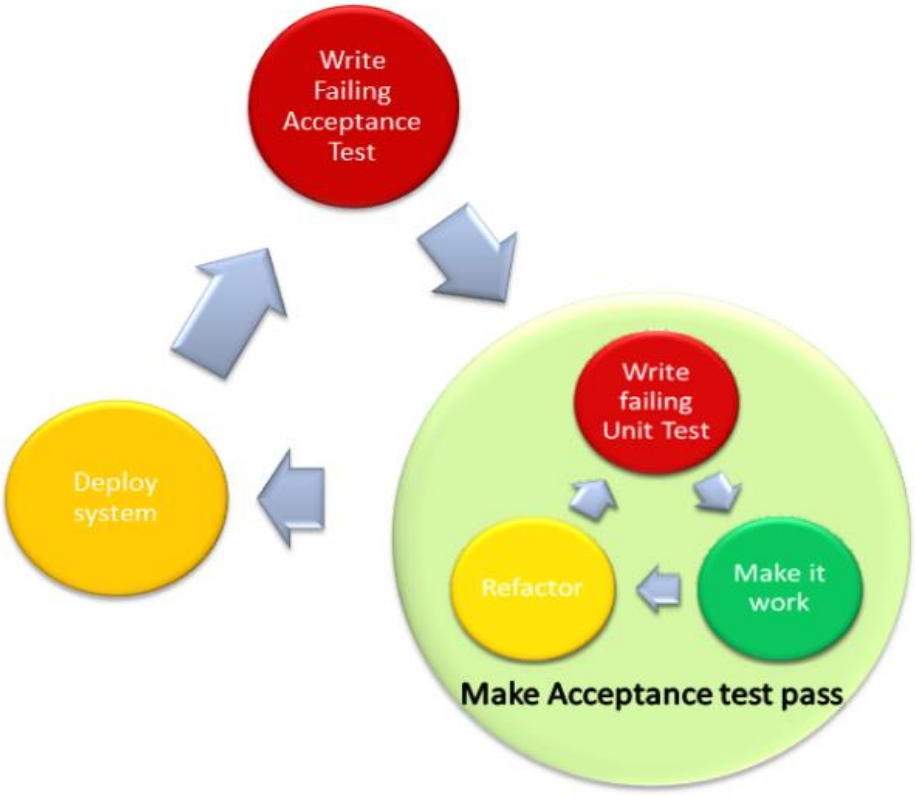
- Tener el código separado de los tests, en carpetas diferentes.
- Los tests deben fallar la primera vez que son escritos.
- Los nombres de los tests deben ir acorde con la intención, deben ser nombres expresivos.
- Refactorizar para eliminar código duplicado después de pasar los tests.
- Repetir las pruebas después de cada refactorización.
- Solo se debe escribir nuevo código, cuando algún test falla. Cada test debe comprar un nuevo comportamiento, o diferente.
- Escribe primero el *assert*.
- Todos los tests deben ser pasados antes de escribir otro test.
- Solo se refactoriza cuando todos los tests pasan.
- Escribe el mínimo y simple código para pasar las pruebas.
- No usar las dependencias entre tests. Los tests deben pasar en cualquier orden.
- Los tests deben ser rápidos.
- Crear solo un test por iteracion

BDD Y ATDD

ATDD (Acceptanec Test Driven Develpment)

- Los criterios de aceptación son definidos antes del proceso de desarrollo y van a guiar al proceso subsecuente.
- Ejercicio colaborativo que involucra al product owner, analistas de negocio, testers y desarrolladores, y ayuda a asegurar que todos los miembros del proyecto entendieron que se necesita para que el proyecto este correcto
- los equipos crean uno o más test de nivel de aceptación para una característica antes de comenzar a trabaja
- Los test unitarios aseguran que la aplicación se construyó correctamente
- Los test de aceptación aseguran que se ha desarrollado la funcionalidad esperada

ATDD



5.3. Planificación y programacion

Objetivos

Aplicar las herramientas de planeamiento, programación y control de tareas en el desarrollo de sistemas de información

Habilidades y competencias que desarrolla la asignatura

- [Plantea]
- [el Proceso de Desarrollo de Software]
- [Para planear, programar, supervisar y controlar los tiempos de un proyecto]
- [Aplicando el método del camino crítico] / [Aplicando el estándar UML]

Datos vs Información

Planeamiento y Programación de Proyectos

Un proyecto, cualquiera sea su característica, 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.

El análisis y diseño de sistemas de información involucra una serie de actividades diversas que al integrarse constituyen un proyecto informático

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.

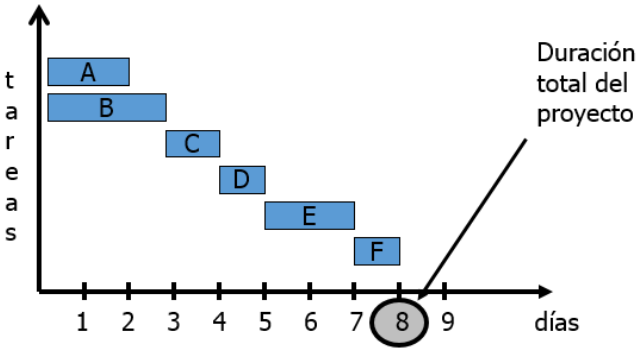
Por último, el **control** se encarga de verificar si se cumple con lo planeado y lo programado anteriormente

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
- Se destaca la sencillez y la facilidad de comprensión al integrar gráficamente la planificación, la programación y el progreso del proyecto.
- Permite visualizar rápidamente los elementos principales, su programación en el tiempo y, además, el progreso en cada uno de ellos.

Diagrama de Gantt – construcción

Actividad	Duración	Precedencia
A	2	-----
B	3	----
C	1	B
D	1	A, C
E	2	D
F	1	E



Método del camino crítico (CPM) Critical Path Method

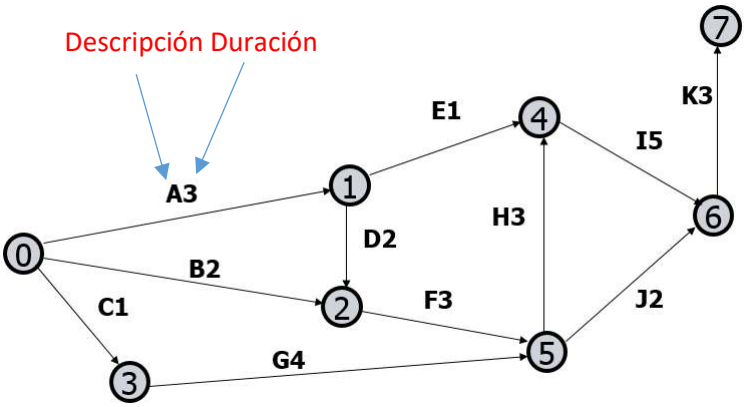
- El Diagrama de Gantt tiene como principal ventaja su simplicidad. Pero:
  - No permite determinar el impacto del atraso de una actividad
  - En proyectos complejos es necesaria mayor información
- Una opción (y complemento al Gantt) es el método del camino critico

Las preguntas principales en la elaboración de un proyecto son:

- ¿Cuál es el tiempo total que se requiere para terminar el proyecto?
- ¿Cuáles son las fechas programadas de inicio y de terminación para cada actividad específica?
- ¿Qué actividades son "críticas" y deben terminarse exactamente según lo programado para poder mantener el proyecto dentro del programa?
- ¿Cuánto se pueden demorar las actividades "no críticas" antes de que ocasionen demoras en el proyecto total?

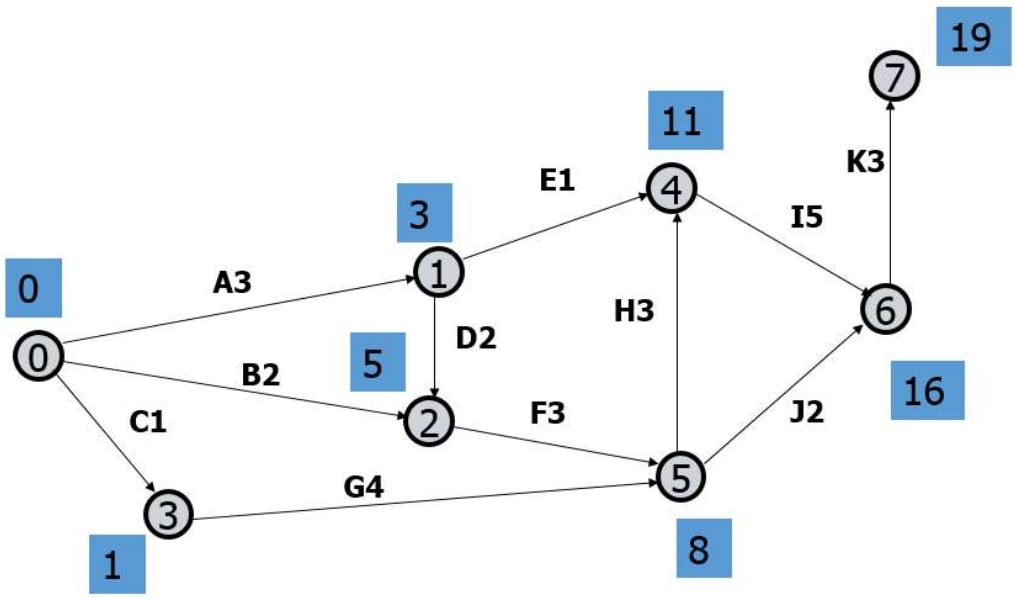
1- Armado de la malla. Establezco, mediante un gráfico, las precedencias de las tareas

Tarea	Descripción	Duración	Precedencia
0-1	A	3	-
0-2	B	2	-
0-3	C	1	-
1-2	D	2	A
2-5	F	3	B,D
3-5	G	4	C
1-4	E	1	A
5-4	H	3	F,G
4-6	I	5	E,H
5-6	J	2	F,G
6-7	K	3	I,J



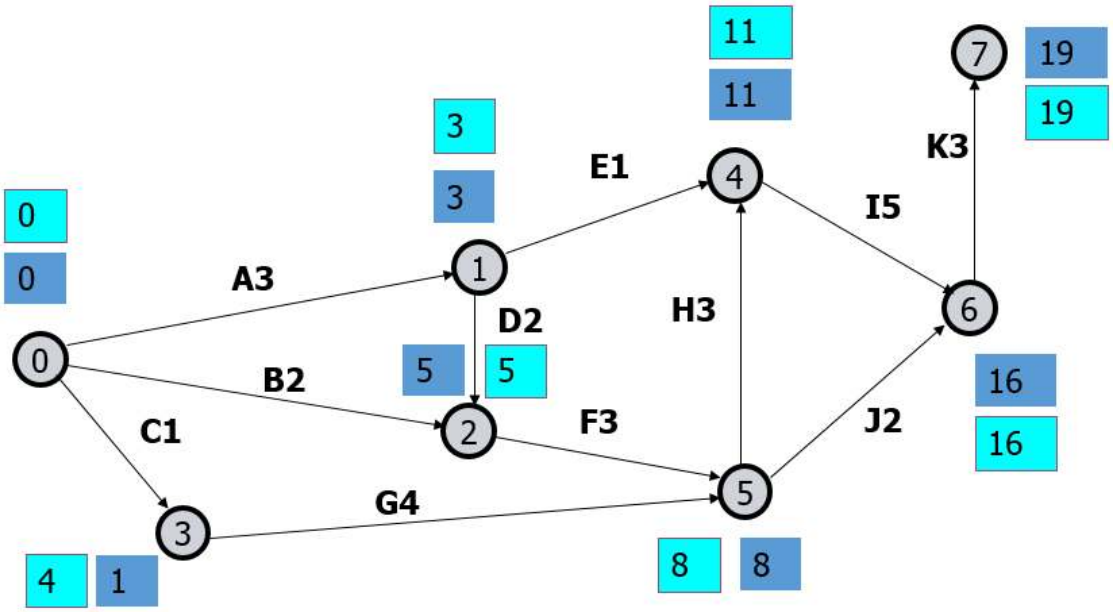
2- La **Fecha temprana** de un nodo determina el momento a partir del cual pueden comenzarse las tareas que nacen de ese nodo

$T_{ej} = t_{ei} + t_{ij}$



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

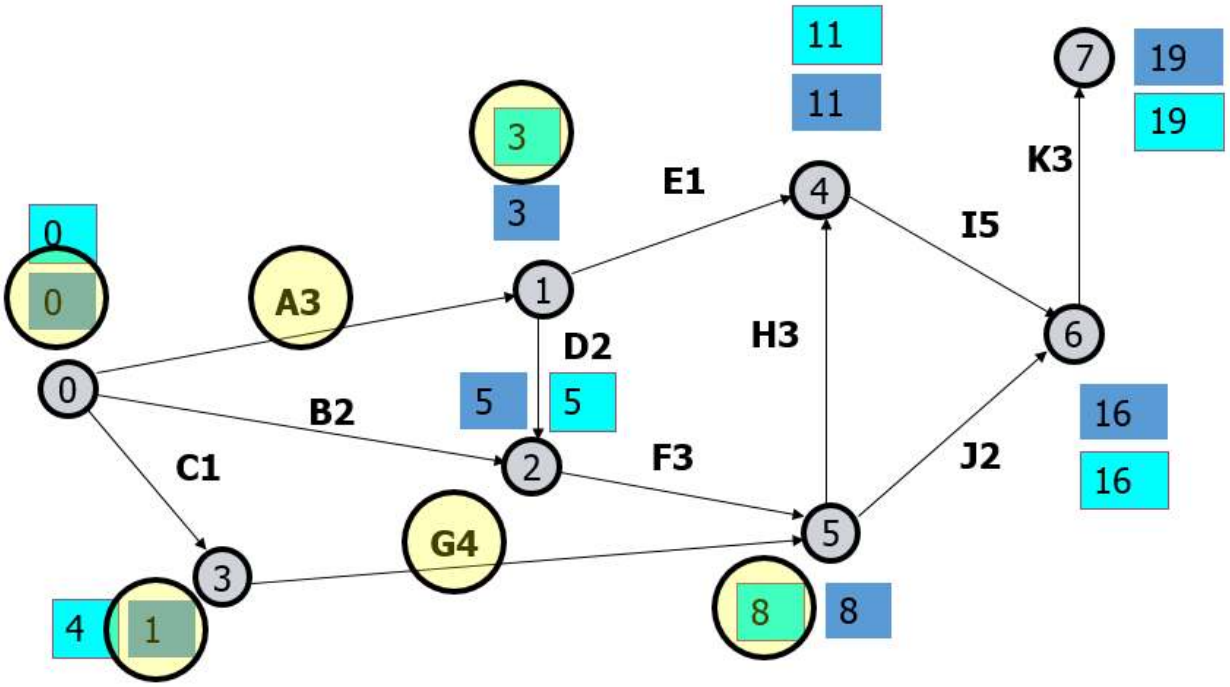
$T_{ai} = t_{aj} - t_{ij}$



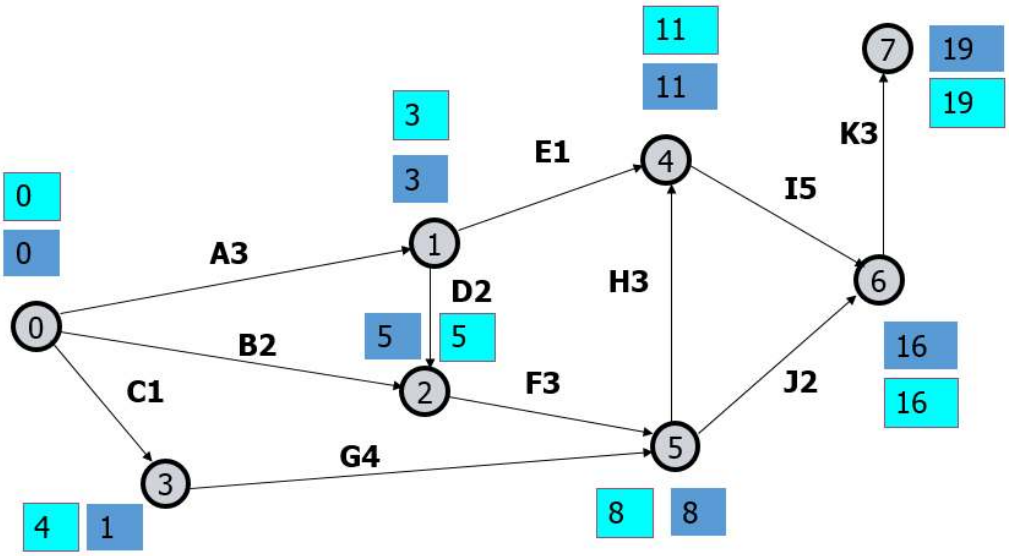
Máximo del proyecto

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

$MT = t_{aj} - t_{ei} - t_{ij}$



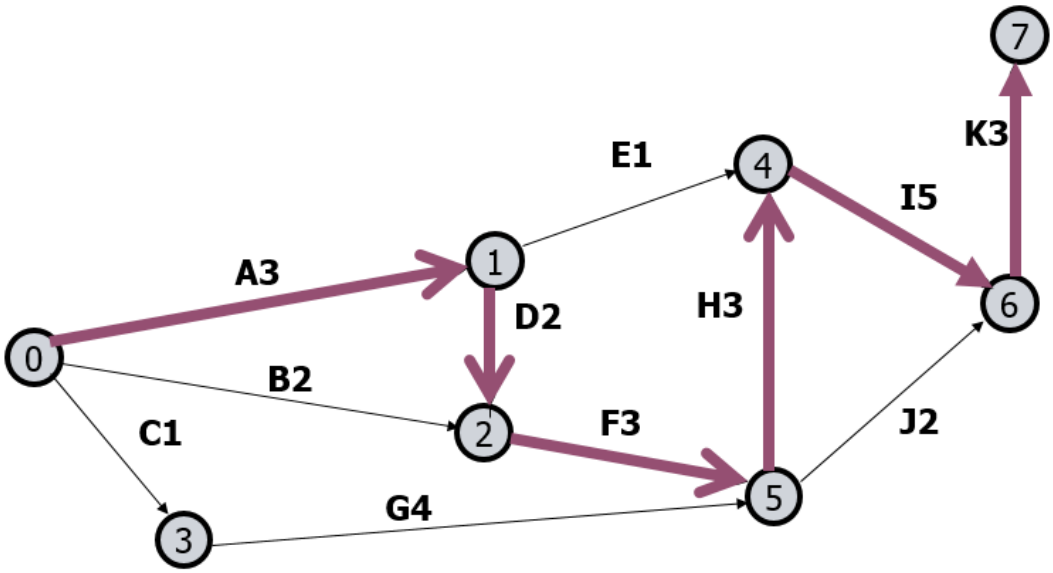
Margen Total MT = $t_{aj} - t_{ei} - t_{ij}$



Tarea	T_{aj}	T_{ei}	T_{ij}	MT
A	3	0	3	0
B	5	0	2	3
C	4	0	1	3
D	5	3	2	0
E	11	3	1	7
F	8	5	3	0
G	8	1	4	3
H	11	8	3	0
I	16	11	5	0
J	16	8	2	6
K	19	16	3	0

Los que tienen MT (Margen Total) en cero forman el camino crítico. Ya que no tienen margen.

Camino crítico = margen total = 0



Metodologías ágiles

Metodologías ágiles para gestión de proyectos: SCRUM

Ágiles – Ventajas

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

- 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

Tradicionales Vs. Ágiles

- Entorno muy cambiante
  - Costos, riesgos
- Requisitos
- Burocracia
- Tiempo de Respuesta
- Planificación
- Planificación vs. Respuesta al cambio

Scrum - Introducción

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 auto-gestión, auto-organización y compromiso.
- Simplicidad. Supresión de artefactos innecesarios en la gestión del proyecto

Desarrollo secuencial vs. superpuesto

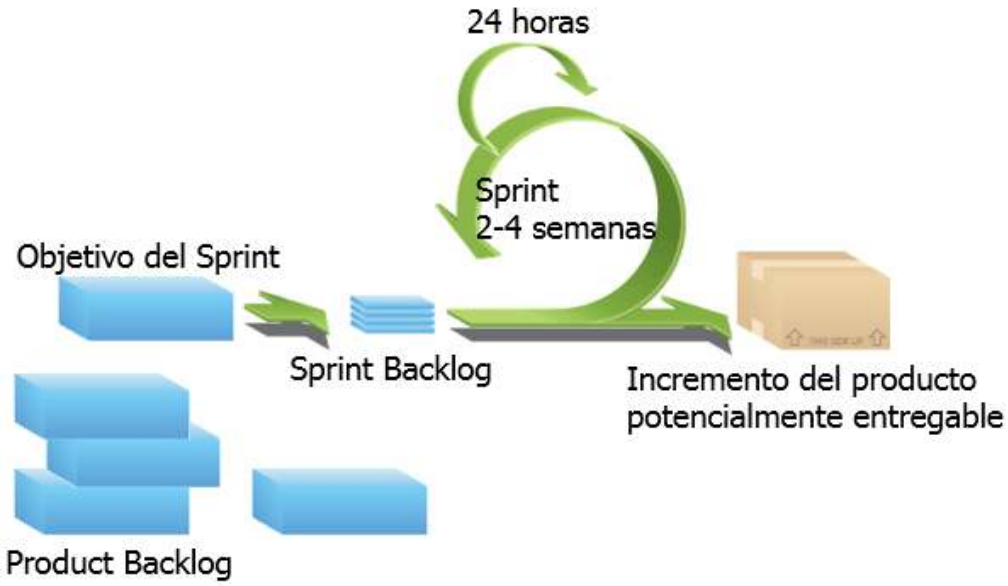
Scrum en 100 palabras

- 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.

Scrum – Introducción

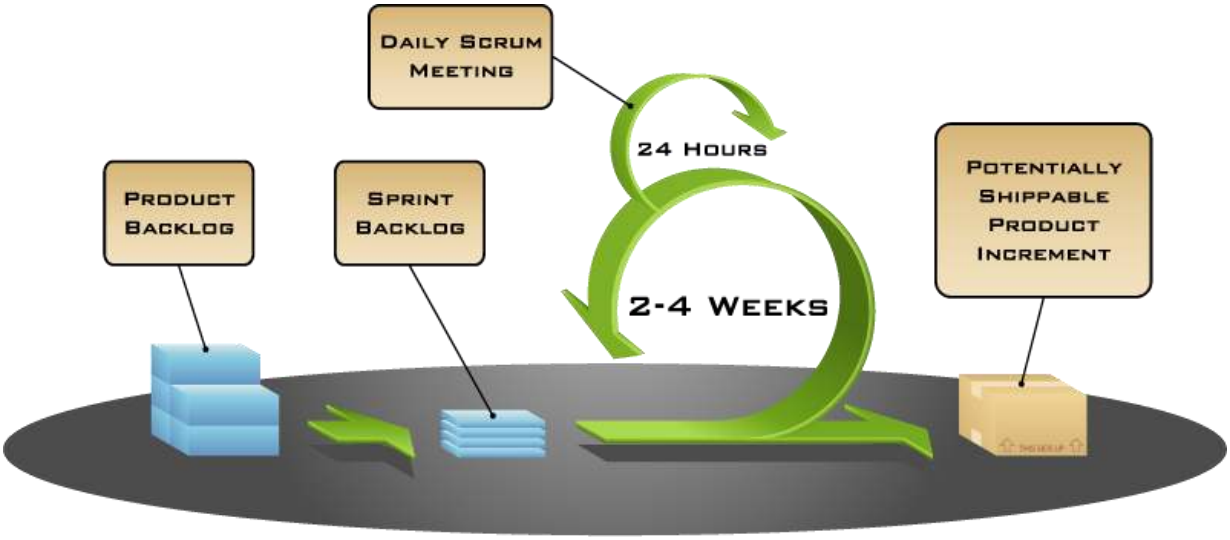
- Conjunto de prácticas y roles
- Posee documentos y artefactos específicos
- Equipos auto-organizados
- Gestión de la expectativa del cliente (cambios)
- Mejoras
    - Productividad
    - Calidad
    - Motivación

Esencia



El Manifesto Ágil - una declaración de valores

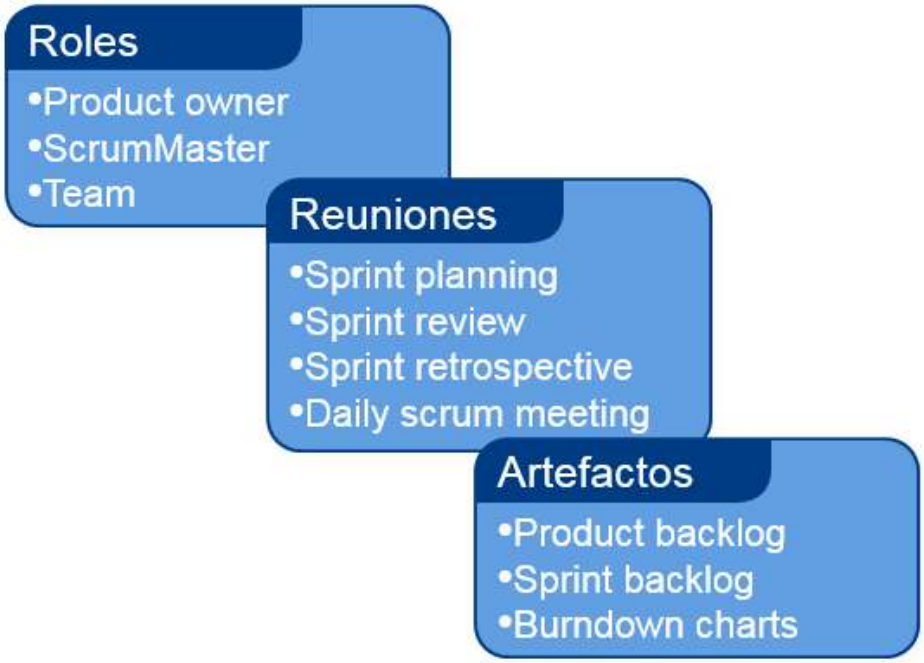
Artefactos



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.

Scrum Framework



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.

El ScrumMaster

- 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

El Team

- Típicamente de 5 a 9 personas
- Multi-funcional:
  - Programadores, testers, analistas, diseñadores, etc.
- Los miembros deben ser full-time
  - Puede haber excepciones (Ej.: Infraestructura, SCM, etc.)
- Los equipos son auto-organizativos
  - Idealmente, no existen títulos pero a veces se utilizan de acuerdo a la organización
- Solo puede haber cambio de miembros entre los sprints

Scrum Framework

Daily Scrum

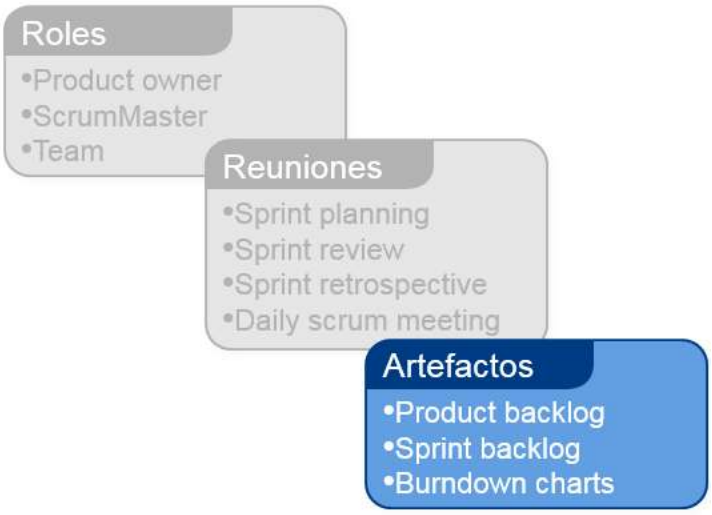
- Parámetros:
  - Diaria
  - Dura 15 minutos
  - Parados
- No para la solución de problemas:
  - Todo el mundo está invitado
  - Sólo los miembros del equipo, ScrumMaster y Product Owner, pueden hablar
  - Ayuda a evitar otras reuniones innecesarias

Todos responden 3 preguntas:

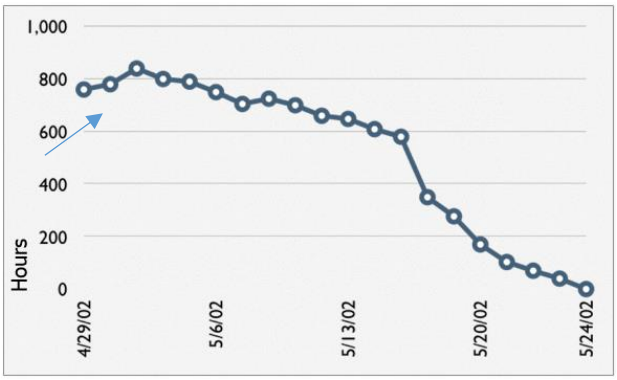
1. ¿Qué hiciste ayer?
2. ¿Qué vas a hacer hoy?
3. ¿Hay obstáculos en tu camino?

- No es dar un status report al Scrum Master
- Se trata de compromisos delante de pares

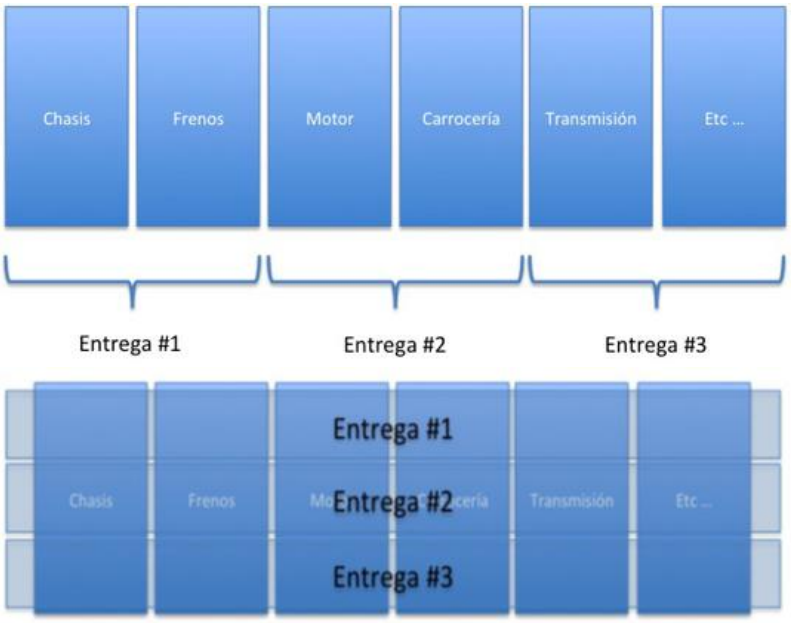
Scrum Framework



Un Sprint Burndown Chart



Ejemplo: Construcción de un vehículo.

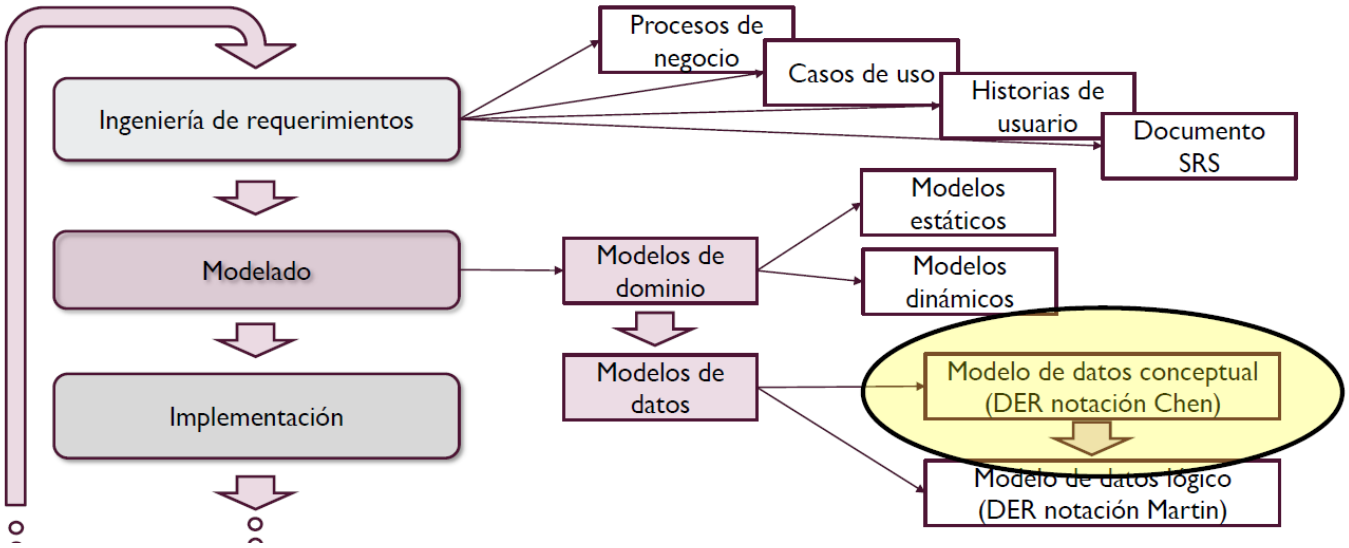


6.1. Modelo Entidad Interrelación

Objetivos

Habilidades y competencias que desarrolla la asignatura

Modelos de datos

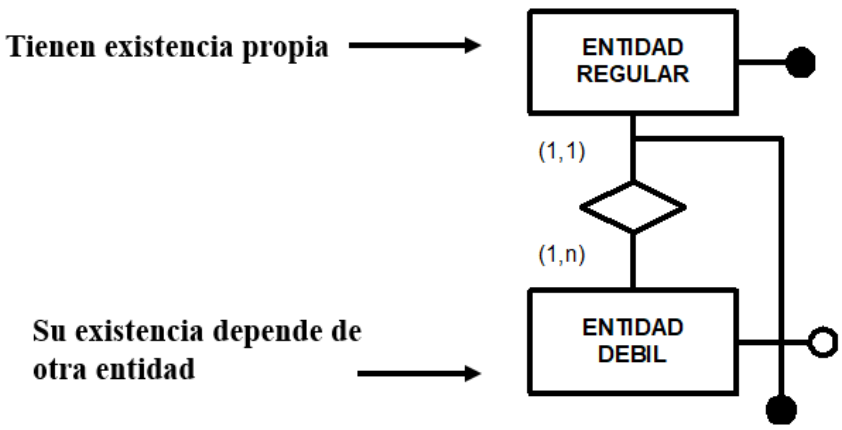


Modelo entidad interrelación → **Modelo conceptual** *Independiente de la implementación*

Transformación

Modelo Relacional → **Modelo lógico** *dependiente de la implementación*

Entidades regulares y débiles



Atributos

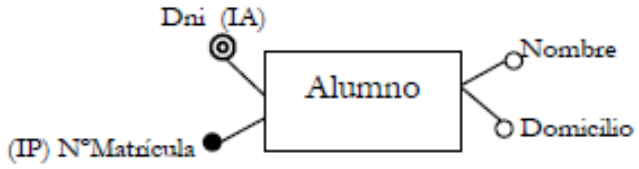
Las entidades tienen atributos que lo describen o identifican. Estos pueden ser:

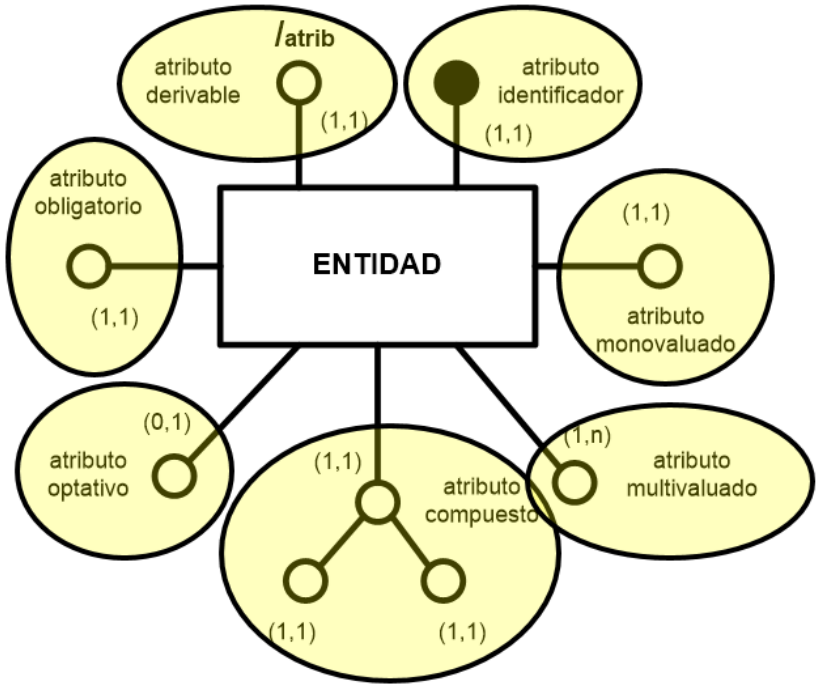
- Descriptivos / Identificatorios
- Monovaluados / Multivaluados
- Obligatorios / Optativos

(Esta caracterización depende de la multiplicidad máxima y mínima)

- Simples / Compuestos
- derivable

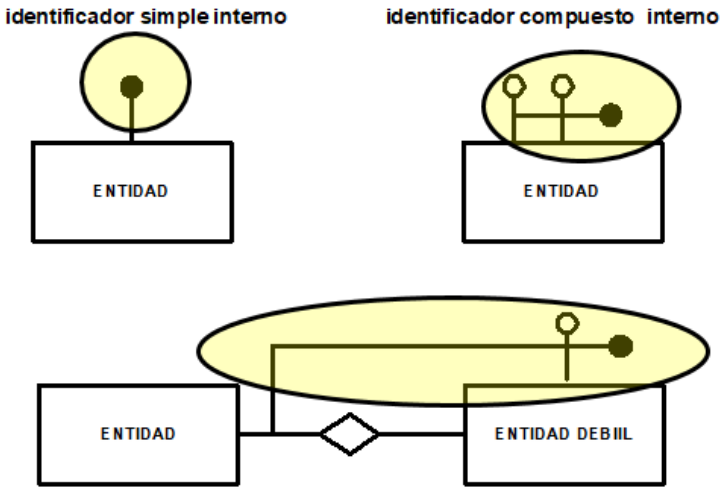
Cada atributo está asociado a un dominio particular. (tipo de dato)





el par (1,1) puede omitirse

Identificadores



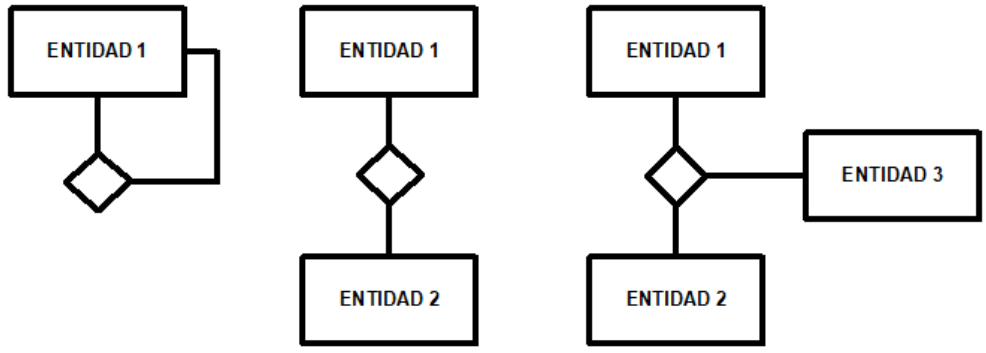
identificador externo, compuesto y mixto

Propiedades de los identificadores ***minimalidad*** *y* ***unicidad***

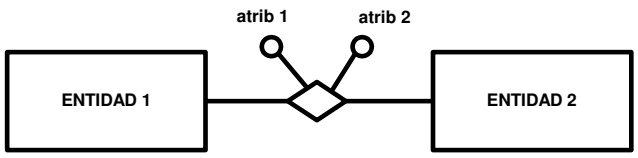
Minimalidad**:** cada concepto debe tener un significado distinto

Unicidad: determina si el posible dato del atributo puede ser tomado como *identificador* único de la *entidad,*

Interrelaciones

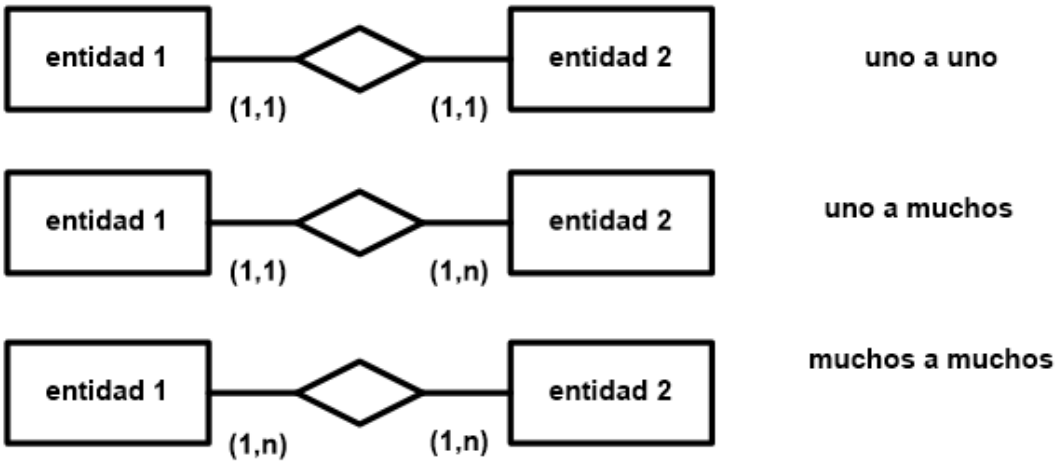


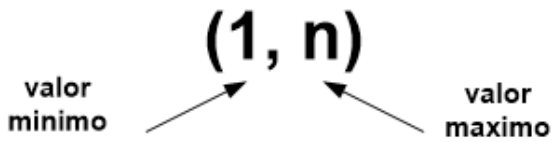
interrelación unaria **interrelación binaria** **interrelación n-aria**



Las interrelaciones pueden tener atributos. Se denominan: **atributos descriptivos**

Multiplicidad en las interrelaciones binarias

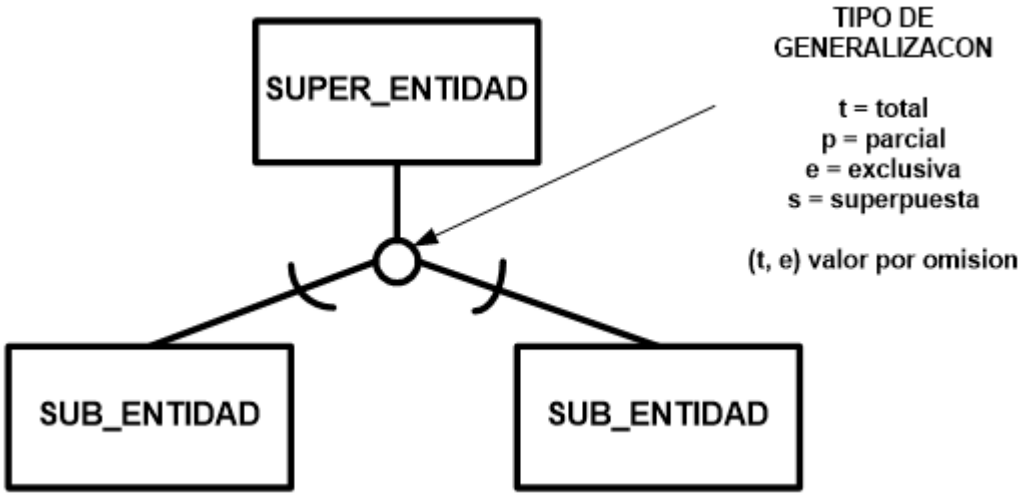




Para analizar la multiplicidad, parto de un objeto de la entidad y pregunto con cuántos objetos de la otra puede conectarse

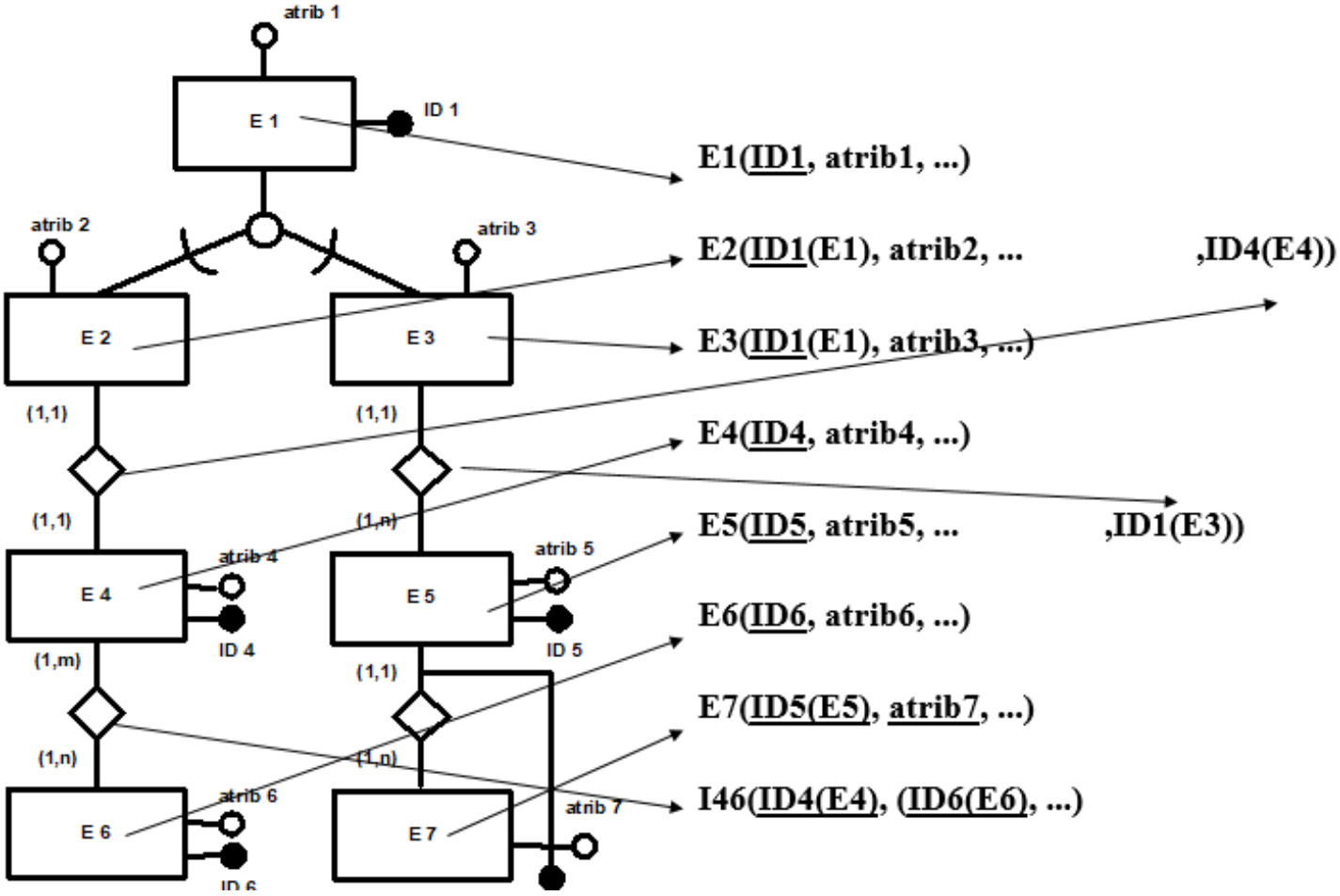
Relación de Generalización

Las sub_entidades heredan atributos, interrelaciones y generalizaciones de la super_entidad



Transformación

Entidades	Se transforman en Tablas
Atributos	Se transforman en Campos de la tabla
Identificador	Se transforma en Identificador de la tabla
Interrelación binaria (y n-arias) M:N	Se transforman en Tabla, el identificador es la unión de los identificadores de las entidades intervinientes, cada uno de ellos es clave foránea con referencia a la entidad (tabla) respectiva. Los atributos descriptivos pasan a ser campos a la tabla generada
Interrelaciones 1:N, 1:1	Atributo identificador del lado "1" pasa como clave foránea al lado "N"
Generalización	Opción 1, sub_entidades y sub_entidades se transforman en tablas, los atributos de ambas pasan a las tablas respectivas como campos, el identificador de la super_entidad será el identificador de la tabla, el atributo identificador en las sub_entidades será, además, clave foránea con referencia a la super_entidad
	Opción 2, desaparece la super_entidad y todos los atributos de ésta pasan a ser atributos en las sub_entidades (incluyendo el identificador)
Entidad débil	Se transforma en Tabla
Atributos	Se transforman en campos de la tabla
Identificador	Unión del identificador de la entidad fuerte (que además es clave foránea con referencia a ésta) más el atributo discriminante



Pasaje del modelo ER al modelo relacional / Paso técnicos

- **Paso 1**. Por cada tipo normal de entidades E del esquema ER, se crea una relación R que contenga todos los atributos simples de E. Se
- **Paso 2**. Por cada tipo de entidad débil D del esquema ER con entidades propietarias E, se crea una relación R y se incluyen todos los atributos simples de D como atributos de R. Además, se incluyen como atributos de clave

externa de R los atributos de clave primaria de la o las entidades propietarias de D. La clave primaria de R es la combinación de las claves primarias de las propietarias y la clave parcial de D, si existe.

- **Paso 3**. Por cada relación binaria 1:1, R, del esquema ER, se identifican las entidades S y T que participan de R. Se elige un de las entidades; por ejemplo, S y se incluye como clave foránea de S a la clave primaria de T. Es mejor elegir una entidad con participación total en R en el papel de S. Se incluyen todos los atributos simples de la relación R como atributos de S.

- **Paso 4**. Por cada relación binaria 1:N, R, se identifica la relación S que representa el tipo de entidades participante del lado N de la relación. Se incluye como clave foránea en S a la clave primaria de la relación T que participa en R; la razón es que cada instancia del lado N está relacionada con un máximo de una instancia del lado 1. Se incluyen

- **Paso 5**. Por cada tipo de vínculo binario M:N, R, se crea una nueva entidad S. Se incluyen como atributos de clave foránea en S a las claves primarias de las entidades que participan de la relación; su combinación constituirá la clave primaria de S. También se incluyen todos los atributos componentes de la relación R en S. todos los atributos simples de la relación como atributos de S.

- **Paso 6**. Por cada atributo multivaluado A se crea una nueva relación R que contiene un atributo corespondiente a A más el atributo de clave primaria K (como clave foránea en R) de la entidad que contiene a A como atributo multivaluado. La clave primaria de R es la combinación de A y K.

- **Paso 7**. Por cada relación *n*-aria R, n > 2, se crea una nueva relación S que representa a R. Se incluyen como atributos de clave foránea en S las claves primarias de las relaciones que representan las entidades participantes. También se incluyen los atributos de la relación *n*-aria como atributo de S. La clave primaria de S casi siempre es una combinación de todas las claves foráneas que hacen referencia a las entidades que participan de la relación.

Ejemplo

El sistema administra la información relacionada con un club deportivo. Los socios pueden ser de distintas categorías; ésta depende de su antigüedad. La cuota social, tiene dos componentes, consta de un básico que depende de la categoría del socio más un plus que depende del deporte elegido. Se desea tener, además, información sobre los deportes que practica cada socio, con la siguiente información: deporte, día y hora de práctica, profesor y arancel. Un socio puede practicar distintos deportes. Cada profesor trabaja en un sólo deporte, pero, por supuesto, en cada deporte trabaja más de un profesor.

Traducción de Requerimientos a Consultas

Una práctica no muy extendida, pero muy útil, para aclarar los requerimientos y validar el diseño propuesto, es traducir los primeros en consultas: un diseño será apropiado de si es capaz de satisfacer las consultas que se realizan sobre el modelo propuesto. por ejemplo:

- Listado de socios por categoría
- Listado de socios y sus cuotas pagas
- Listado de socios con los deportes que practican, días y horarios
- Listado de pagos de cada socio el año 2005 en concepto de cuotas social

Identificando Entidades e Interrelaciones

Entidades:

- SOCIO,
- CATEGORÍA,
- PROFESOR,
- DEPORTE,
- CUOTA

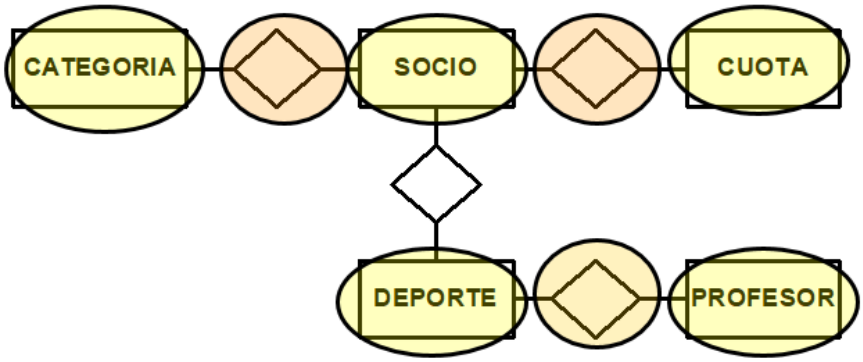
Interrelaciones:

- SOCIO-DEPORTE,
- DEPORTE-PROFESOR,
- SOCIO-CATEGORIA,
- SOCIO-CUOTA

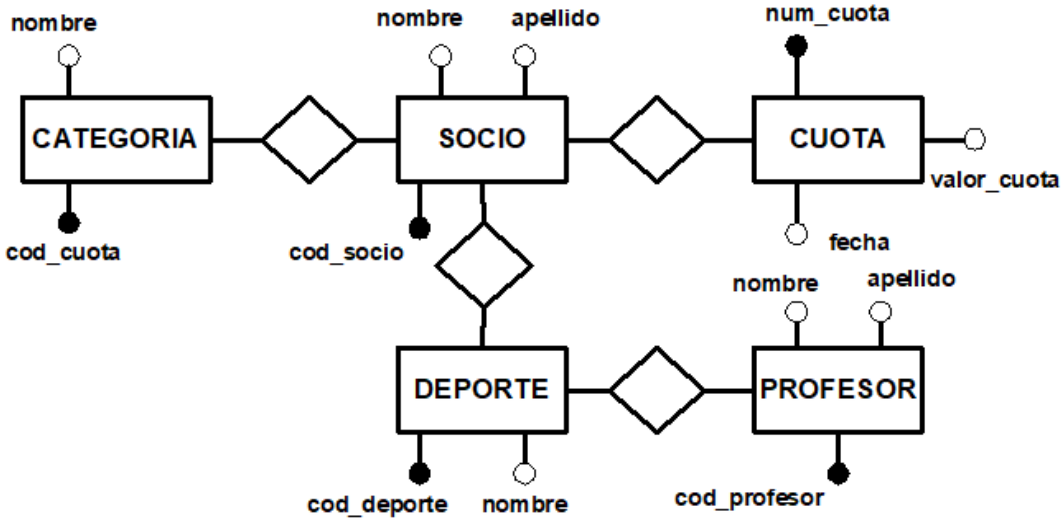
Primera Aproximación al Modelo

Entidades: SOCIO, CATEGORÍA, PROFESOR, DEPORTE, CUOTA

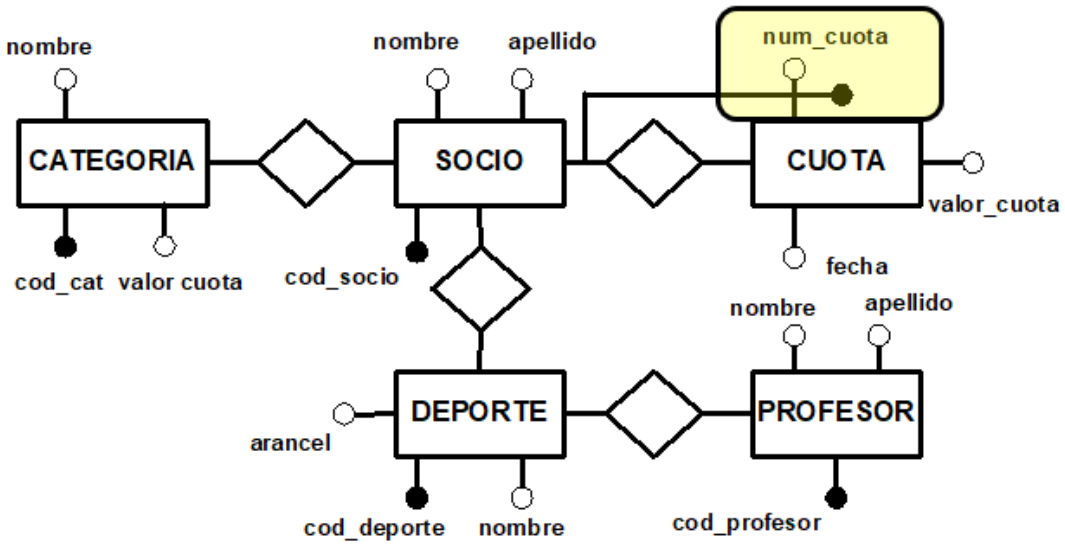
Interrelaciones: SOCIO-DEPORTE, DEPORTE-PROFESOR, SOCIO-CATEGORIA, SOCIO-CUOTA



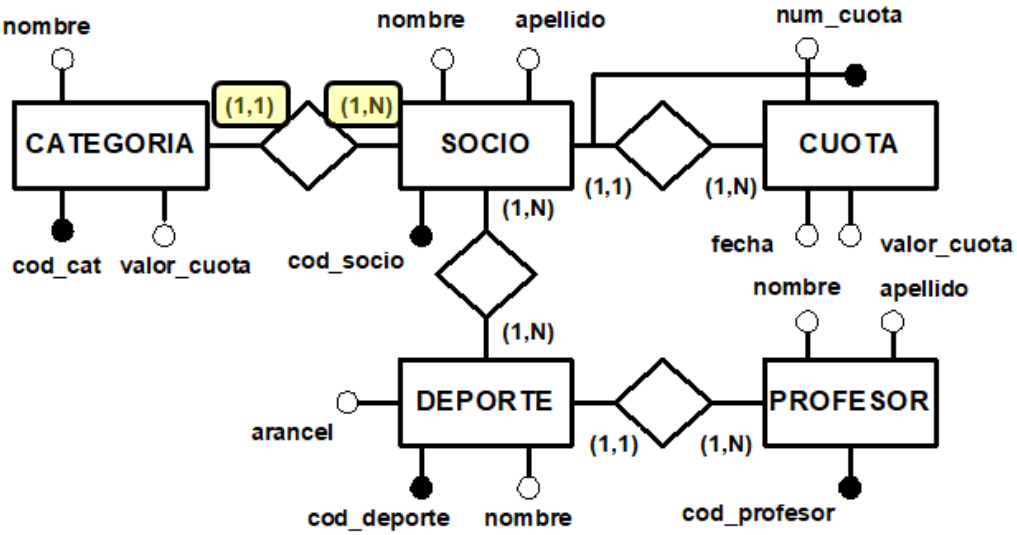
Atributos en las Entidades



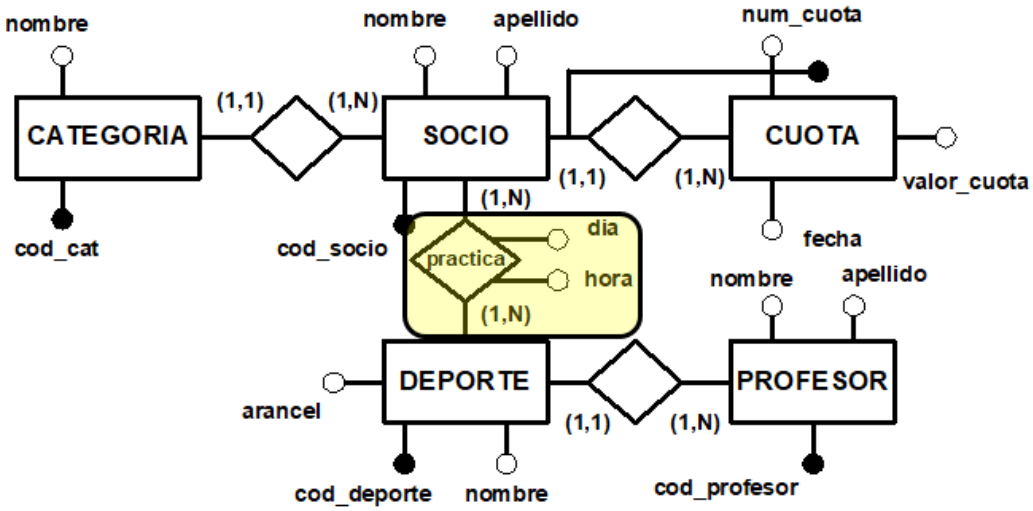
De entidad regular a débil



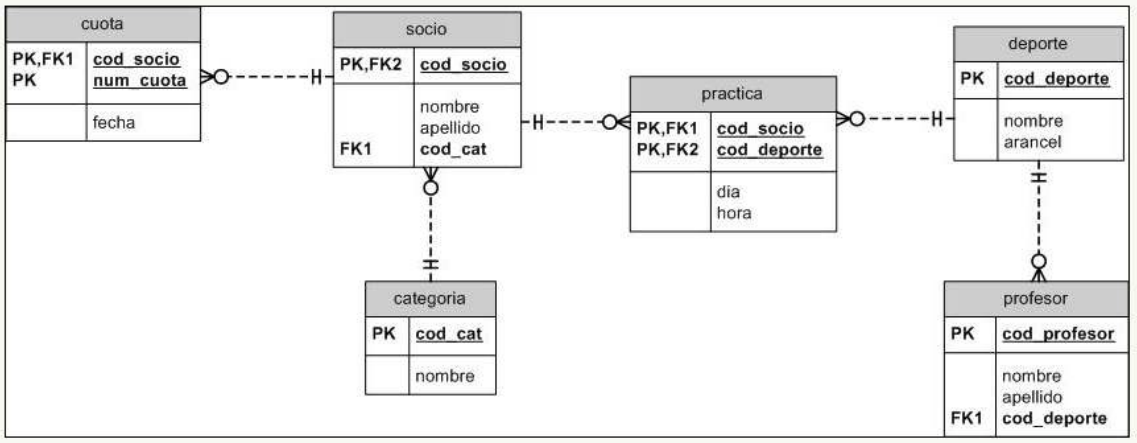
Multiplicidad de las interrelaciones



Atributos de las Interrelaciones



El mismo diagrama diagrama transformado a notación Martin(pata de gallo).



Ejemplo - Consultas en SQL

Listado de socios por categoría

```
SELECT socio.nombre, socio.apellido, categoria.nombre
FROM socio, categoria
WHERE socio.cod_cat = categoria.cod_cat
ORDER BY categoria.nombre;
```

socio.nombre	apellido	categoria.nombre
Mario	Perez	Cadete
Jorge	Rodriguez	Cadete
Juana	Rey	Juvenil

Listado de socios y sus cuotas pagas

```
SELECT socio.cod_socio, socio.apellido,
       cuota.num_cuota, cuota.fecha
FROM socio, cuota
WHERE socio.cod_socio = cuota.cod_socio
ORDER BY socio.cod_socio,
         cuota.num_cuota;
```

cod_socio	apellido	num_cuota	fecha
01	Rodriguez	01	01/01/2005
01	Rodriguez	02	02/01/2005
02	Perez	01	01/01/2005
02	Perez	02	02/01/2005

Listado de socios con los deportes que practican, días y horarios

```
SELECT socio.apellido, socio.nombre,
       deporte.nombre, practica.dia, practica.hora
FROM socio, deporte, practica
WHERE socio.cod_socio = practica.cod_socio
AND deporte.cod_deporte = practica.cod_deporte
ORDER BY socio.apellido;
```

apellido	socio.nombre	deporte.nombre	dia	hora
Perez	Mario	Basquet	lunes	8:00
Rey	Juana	Futbol	miercoles	13:00
Rodriguez	Jorge	Natación	martes	10:00
Rodriguez	Jorge	Basquet	lunes	8:00

Monto total pagado por cada socio el año 2020 en concepto de cuotas sociales

```
SELECT socio.cod_socio, socio.apellido,
SUM(cuota.valor_cuota) AS [pago 2020]
FROM socio, cuota
WHERE socio.cod_socio=cuota.cod_socio
AND cuota.fecha BETWEEN '2020/01/01' AND
'2020/12/31'
GROUP BY socio.cod_socio, socio.apellido;
```

cod_socio	apellido	pago 2020
01	Rodriguez	30.600
02	Perez	22.800

6.2. Base de Datos (Parte 1)

Modelado y Persistencia de Datos.

Conceptos de Base de datos.

Objetivos

- Comprender la importancia del concepto y utilización de una base de datos.
- Identificar las anomalías de actualización en una estructura de almacenamiento ineficiente.
- Comprender como se logran armar estructuras de almacenamiento eficientes.

Habilidades y competencias que desarrolla la asignatura

Conceptos de Base de datos.

Habilidades y competencias que desarrolla la asignatura.

- Lograr analizar las características distintivas de las bases de datos.
- Adquirir conocimientos de los principios fundamentales de una base de datos.
- Lograr diseñar una arquitectura básica de una base de datos.

Base 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 específico.

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 Base de Datos 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 Bases de Datos

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

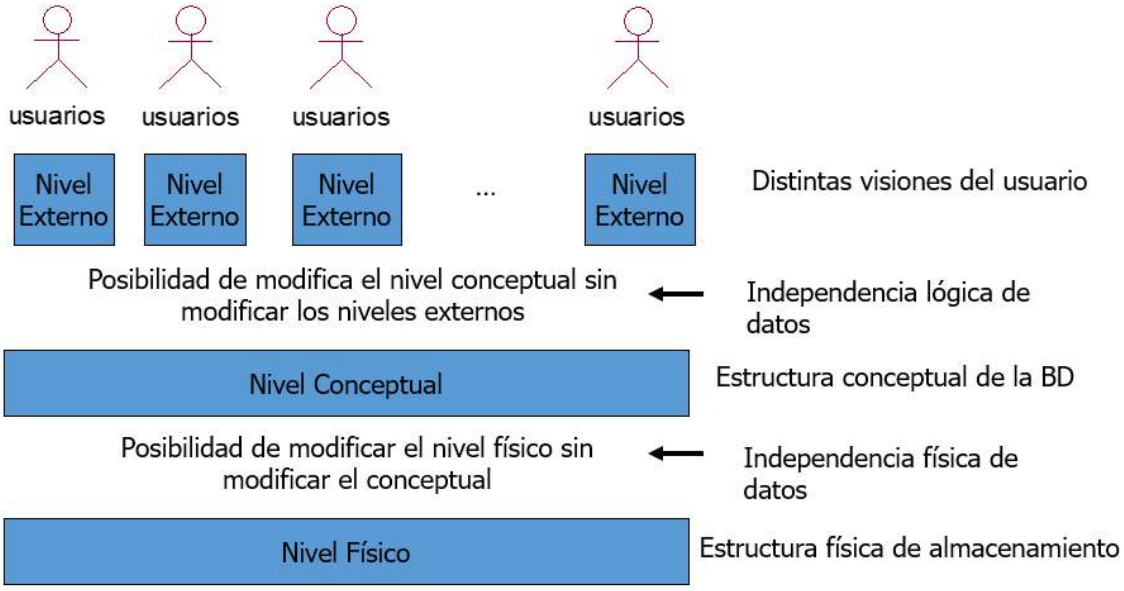
Administrador de la base de datos: supervisa y controla recursos

- primarios: base de datos
- secundarios: SGBD y programas

Diseñador de la base de datos

Usuario Final

Arquitectura de una Base de Datos



Sistemas de bases de datos

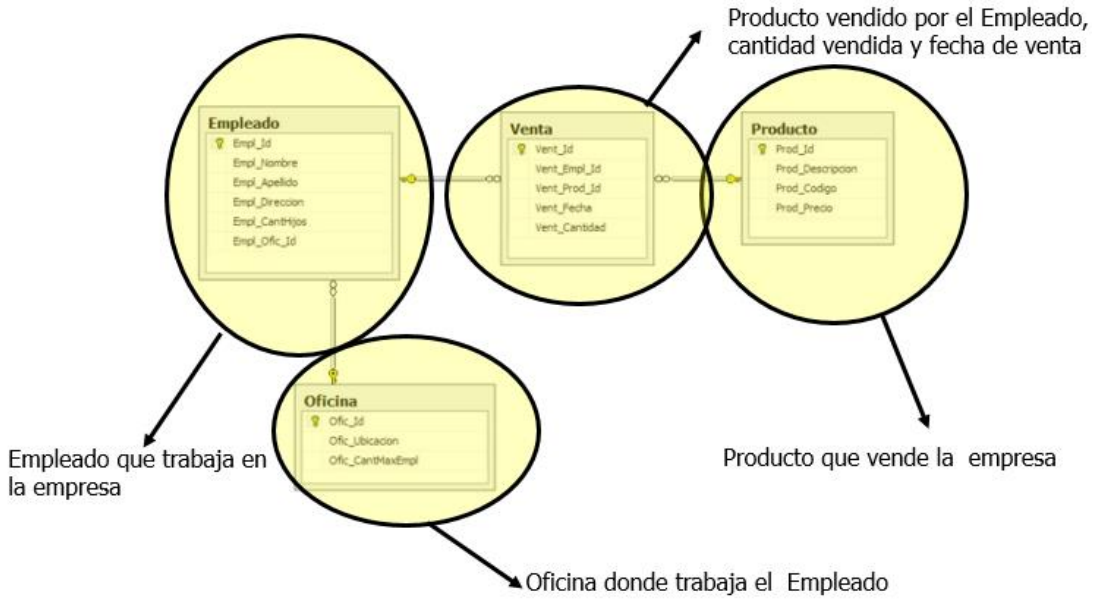
Comparing the features of the mainstream ways of modeling data versus the semantic web model						
Model	Example Format	Data	Metadata	Identifier	Query Syntax	Semantics (Meaning)
Object Serialization	.NET CLR Object Serialization	Object Property Values	Object Property Names	e.g. Filename	LINQ	N/A
Relational	MS SQL, Oracle, MySQL	Table Cell Values	Table Column Definitions	Primary Key (Data Column) Value	SQL	N/A
Hierarchical	XML	Tag/Attribute Values	XSD/DTD	e.g. Unique Attribute Key Value	XPath	N/A
Graph	RDF/XML, Turtle	RDF	RDFS/OWL	URI	SPARQL	Yes, using RDFS and OWL

6.2. SQL (Parte 2)

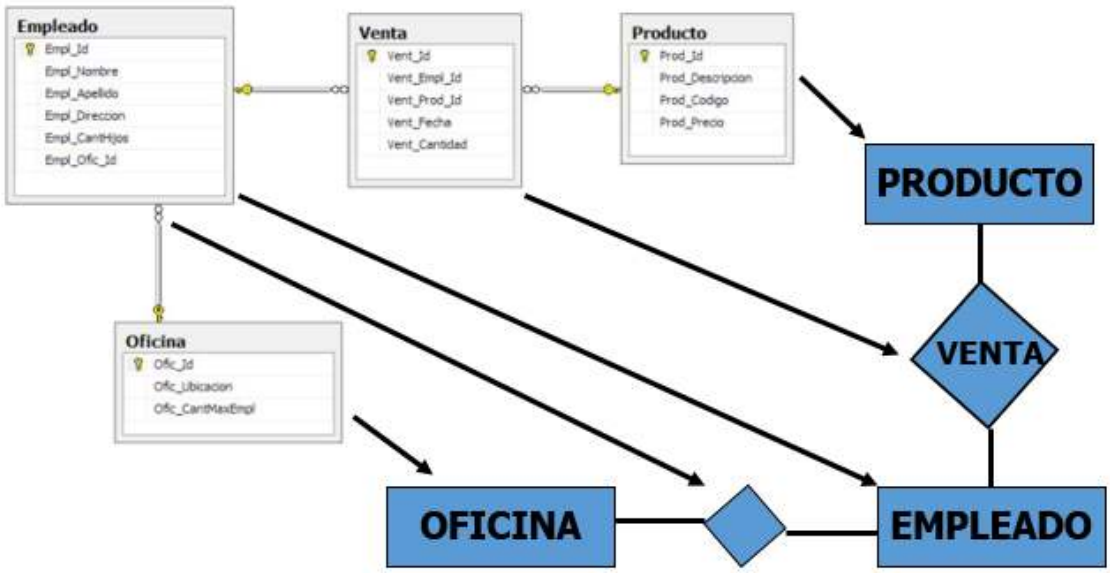
Objetivos

Realizar operaciones de búsqueda de información sobre una estructura de datos utilizando el lenguaje de consultas estructurado.

Ejemplo



Una aproximación al Modelo Entidad Interrelación



Buenas prácticas

Nombres de tablas ¿Singular o plural?

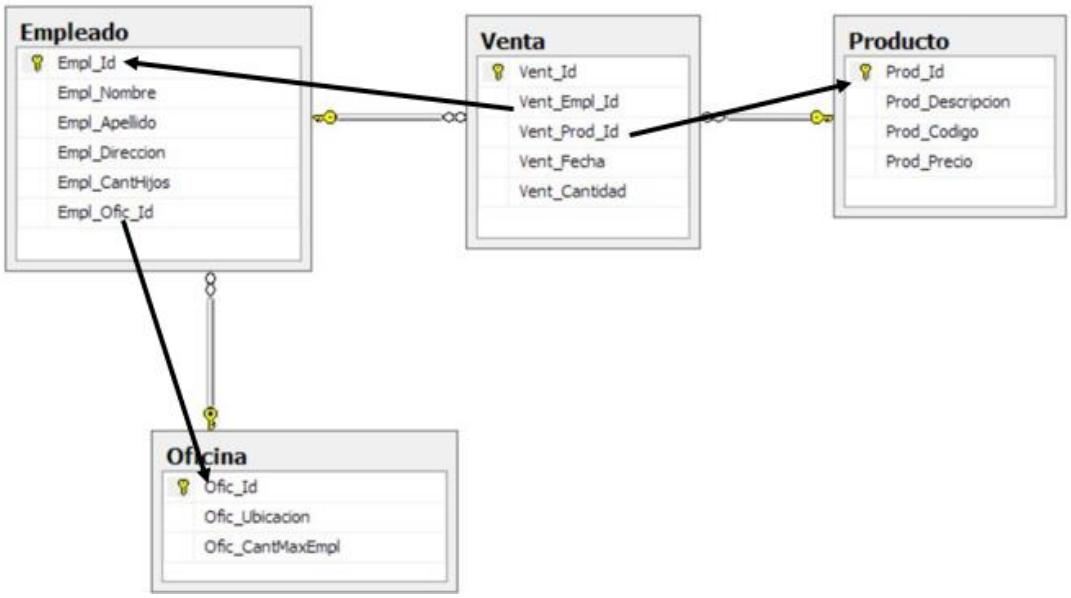
No hay un estándar. La comunidad establece buenas prácticas.

Mantenga coherencia en el modelo.

Si usa singular, úselo en todo el modelo

Si usa plural, sea consistente en todo el modelo.

Es conveniente agregar mnemónico de tabla antecediendo el nombre del campo.

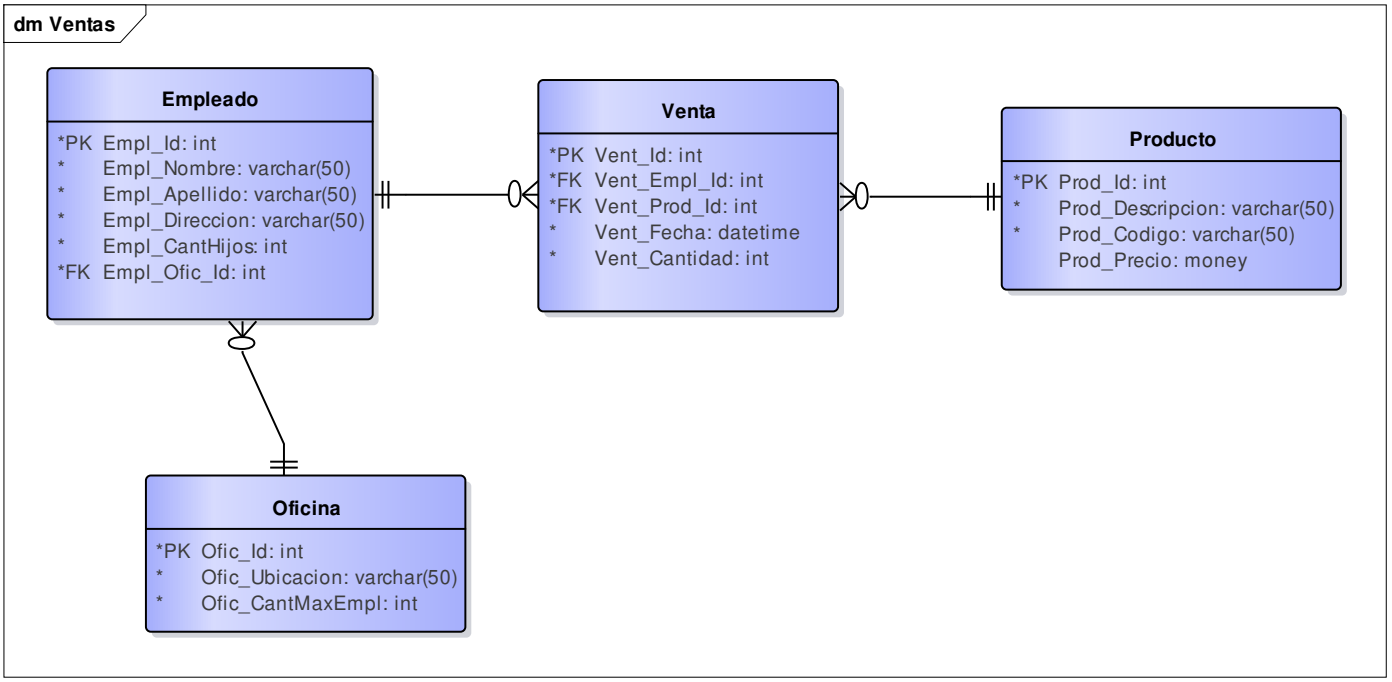


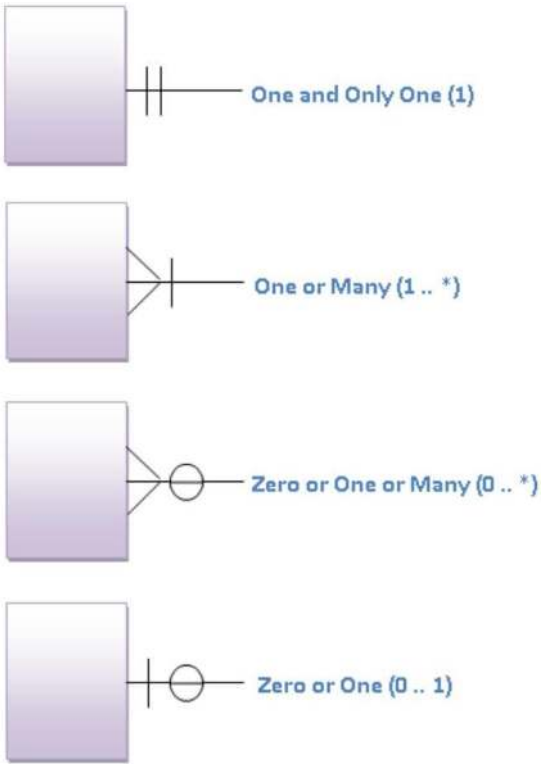
Procurar correspondencia terminológica en el modelo.

Por ejemplo, evitar cosas como:

Clave foránea “Medico_Id” apuntando a la clave primaria “Profesional_Id” de la tabla “Doctor”

Notación Martin (o Pata de gallo)





Otras notaciones

Bachman

IDEF1X

Crow's Foot

Barker's

Chen

Min-Max/ISO

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.

SQL sintaxis básica

SELECT campo(s)

FROM tabla(s)

WHERE <condición>

GROUP BY tabla(s)

HAVING <condición>

ORDER BY campo(s)

Seleccione estos campos (SELECT), de las siguientes tablas (FROM), que cumplan estas condiciones (WHERE), agrúpelos bajo estas condiciones (HAVING) y ordénelos con estos campos (ORDER BY).

SQL consultas básicas

Consultas de recuperación de campos

Selecciones todos los campos de la tabla Empleado

SELECT *

FROM Empleado;

Seleccione el nombre y el apellido del empleado

SELECT nombre, apellido

FROM Empleado;

Seleccione el nombre y el apellido de los empleados que vivan en Morón

SELECT nombre, apellido

FROM Empleado

WHERE direccion ='Moron'

Consultas con operadores lógicos (>, >=, <, <=, <>, AND, OR, NOT)

Seleccione el nombre y el apellido de los empleados que vivan en Morón y tengas más de 3 hijos

SELECT nombre, apellido

FROM Empleado

WHERE direccion ='Moron' AND cant_hijos > 3;

Consultas con ordenamientos

Seleccione el nombre y el apellido y ordénelos por apellido y nombre

SELECT nombre, apellido

FROM Empleado

ORDER BY apellido, nombre;

Consultas usando operadores comodín (*, ?)

Seleccione el nombre y el apellido de los empleados cuyo apellido termine con ´s´ y el nombre tengan una ´m´ como tercera letra

SELECT nombre, apellido

FROM Empleado

WHERE apellido LIKE '%s' AND nombre LIKE '_ _m%';

Funciones de agrupación Count(), Count (*), Sum(), Avg(), Max(), Min()

Determine la cantidad de empleados

SELECT count(*)

FROM Empleado;

Determine la cantidad total de hijos que tienen los empleados en toda la empresa

SELECT sum(cant_hijos)

FROM Empleado;

Consultas con más de una tabla

Listar el nombre y apellido de cada empleado y la ubicación de la oficina que ocupa

SELECT nombre, apellido, ubicación

FROM empleado, oficina

WHERE empleado.oficina = oficina.cod_of

Usando inner join

SELECT nombre, apellido, ubicacion

FROM empleado inner join oficina

on empleado.oficina = oficina.cod_of

nombre	apellido	ubicacion
jorge	perez	1º piso
raul	martinez	1º piso
maria	fernandez	1º piso
rosa	lima	1º piso
pedro	suarez	2º piso

Como se arma la consulta

empleado

cod_emp	nombre	apellido	direccion	cant_hijos	oficina
01	jorge	perez	merlo	4	01
02	raul	martinez	padua	2	01
03	maria	fernandez	moron	3	02
04	rosa	lima	moreno	1	02
05	pedro	suarez	merlo	0	03

oficina

cod_of	ubicacion	cant_max_emp
01	1º piso	20
02	1º piso	25
03	2º piso	10
04	3º piso	15

SELECT nombre, apellido, ubicacion
FROM empleado, oficina
WHERE empleado.oficina = oficina.cod_of

Resultado de la consulta

nombre	apellido	ubicacion
jorge	perez	1º piso
raul	martinez	1º piso
maria	fernandez	1º piso
rosa	lima	1º piso
pedro	suarez	2º piso

Consultas de agrupamiento

Determinar la cantidad de ventas por empleado

SELECT empleado.cod_emp, empleado.nombre, Sum(venta.cantidad)

FROM empleado INNER JOIN venta ON empleado.cod_emp = venta.cod_emp

GROUP BY empleado.cod_emp, empleado.nombre;

cod_emp	nombre	cantidad
01	jorge	30
02	raul	45
03	maria	14

Como se arma la consulta

empleado

cod_emp	nombre	apellido	direccion	cant_hijos	oficina
01	jorge	perez	merlo	4	01
02	raul	martinez	padua	2	01
03	maria	fernandez	moron	3	02
04	rosa	lima	moreno	1	02
05	pedro	suarez	merlo	0	03

venta

cod_emp	cod_prod	fecha	cantidad
01	01	19/02/2006	10
01	02	20/02/2006	15
01	03	21/02/2006	5
02	01	17/03/2006	15
02	02	18/03/2006	30
03	01	25/04/2006	4
03	02	26/04/2006	10

resultado de la consulta

cod_emp	nombre	cantidad
01	jorge	30
02	raul	45
03	maria	14

```
SELECT empleado.cod_emp, empleado.nombre, Sum(venta.cantidad)
FROM empleado INNER JOIN venta ON empleado.cod_emp = venta.cod_emp
GROUP BY empleado.cod_emp, empleado.nombre;
```

Guía de aprendizaje

ALUMNO
COD_ALU TEXTO
NOM_ALU
APE_ALU
CALLE_ALU
NRO_ALU
LOC_ALU
FECH_NAC
FECH_ING
COD_DIV TEXTO

PROFESOR
COD_PRO TEXTO
NOM_PRO
APE_PRO
CALLE_PRO
NRO_PRO
LOC_PRO

MATERIAS
COD_MAT TEXTO
DES_MAT
AÑO_MAT

DIVISION
COD_DIV TEXTO
AÑO_DIV
TUR_DIV

PROF_MAT
COD_PRO TEXTO
COD_MATTEXTO

NOTAS
COD_ALU TEXTO
COD_MAT TEXTO
FECH_NOT
NOT_NOT