Universidad Nacional del Litoral

FACULTAD DE INGENIERÍA
Y CIENCIAS HÍDRICAS

Ingeniería en Informática

Ingeniería de Software I

TEMA I – Sistemas de información

Introducción

Aunque en la actualidad, la computadora es un objeto común en la industria, gobierno, medicina y aún en la política, la gente prefiere mantenerse al margen y acudir a quienes entienden el poder que les puede brindar una computadora.

Por otra parte, cada día es mayor la cantidad de profesionales de distintas disciplinas, que toman conciencia que mediante el uso de un computador, sus trabajos se pueden facilitar enormemente, y motivados por ello, toman distintos tipos de cursos que tal vez les sean provechosos y puedan aplicarlos de una manera correcta (por ejemplo, procesadores de textos, planillas electrónicas, diseño asistido, diseño gráfico, etc.). Debe notarse que, la forma de aplicar tales herramientas por parte de los profesionales que tienen claros sus objetivos, en función de sus conocimientos científicos en la materia de su especialidad, únicamente usarán la computadora para determinadas tareas afines a su principal actividad. Puede existir otro número de tareas que tal vez necesiten ser resueltas implicando el uso del computador, pero que no puedan hacerlas sin recurrir a profesionales de la informática.

Algo similar ocurre con las organizaciones de distinto tipo. Pueden tener empleados muy eficaces como operadores de computadoras, pero hay cosas que no las podrán hacer por las limitaciones de sus conocimientos. Tal vez se recurra también en este caso a profesionales de la informática.

Un profesional de la informática, no debe ser simplemente un programador. Un programador, ve sólo una parte del problema (implementar físicamente en una computadora un programa que resuelva un problema determinado). Un profesional de la informática debe poseer una óptica más amplia, que le permita hacer recomendaciones, brindar opiniones, y asesoramientos en varios aspectos.

Tal vez, en la estructura de una determinada organización, existan reparticiones destinadas a tareas que involucren el manejo de computadoras, o bien que hagan desarrollos de apoyo a distintas actividades. Sus empleados serán “gente de sistemas”, entre los cuales puede haber distintas categorías de por ejemplo Administradores de Sistemas, Administradores de Bases de Datos, Analistas, Programadores, etc. Estos ejercen una influencia considerable en la organización para la que trabajan; en base a sus recomendaciones se instalan nuevos sistemas y se descartan viejos. Tomando como ejemplo un sistema de gestión comercial, mediante la información generada como resultado de los sistemas que se desarrollen, la gerencia puede decidir el curso de la acción para un producto nuevo o ya existente; también es responsable de los informes empleados en la selección de una estrategia que determine la imagen de toda la organización. En ocasiones el analista con experiencia, puede abordar parte de cada una de estas situaciones.

El análisis y el diseño de sistemas, es una de las actividades más difíciles de enseñar en un salón de clases, ya que parte de estas actividades depende de herramientas, experiencia y situaciones muy difíciles de recrear en el aula tradicional. A menudo, cuando se estudia en un instituto o universidad, se subraya la teoría y se descuidan las aplicaciones.

El contenido de esta materia no consiste en una metodología para encarar cuestiones relativas a la ingeniería de software, sino brindar herramientas que permitan posteriormente al profesional, seleccionar adecuadamente y a su criterio de qué manera afrontará los problemas que se le presenten. Se hace hincapié en que siempre es necesario tener una metodología de

trabajo (ya sea clásica o propia) por los siguientes motivos:

- Es beneficioso que tanto usuarios como el equipo de trabajo tengan una muy clara idea de cuáles son los propósitos de cada etapa y sus respectivas fases (conozcan el método).

- Únicamente a través de una metodología ordenada y clara, es posible obtener resultados satisfactorios de un trabajo en equipo, pues ella permite coordinar y hacer comprender a cada integrante del grupo de trabajo, la participación e integración de su respectiva tarea con las del resto.

- La existencia de una metodología, permite fijar con claridad, puntos de control que permitan evaluar la marcha de un proyecto y generar en consecuencia las medidas correctivas pertinentes.

- El trabajo se hace en forma gradual, permitiendo que:

    - se conozca acabadamente el problema
    - se diseñe una solución al máximo grado de detalle
    - se desarrolle la solución planteada

- Se incrementa la probabilidad de que errores significativos, sean detectados antes.

Los conceptos presentados, no apuntan al uso de ningún lenguaje, base de datos u otros componentes de software en particular, sino que son principios aplicables a cualquier situación.

1.1 Noción de sistema

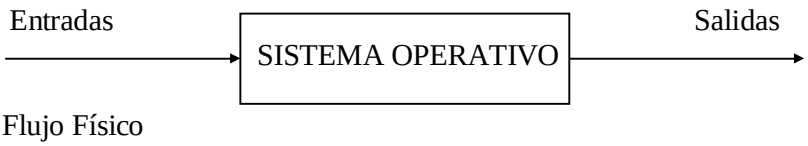
Un sistema es un conjunto de elementos materiales o inmateriales (hombres, máquinas, métodos, reglas, etc.) en interacción, que transforman, mediante procesos, elementos (entradas) en otros elementos (salidas).

Por ejemplo:

Una caldera recibe carbón y mediante la combustión lo transforma en calor.

Se examinarán sólo sistemas constituidos por organizaciones que funcionen con vistas a la realización de determinados objetivos.

Tal sistema físico o sistema operativo transforma un flujo físico de entradas (materias primas, flujos financieros, etc.), en un flujo físico de salidas (productos terminados, flujos financieros, etc.).



En el sentido más amplio, un sistema es simplemente un conjunto de componentes que interactúan para alcanzar algún objetivo. Los sistemas son todo lo que rodea al ser humano, por ejemplo, se sienten sensaciones físicas originadas por un complejo sistema nervioso, un conjunto de partes que incluye el cerebro, espina dorsal, nervios y células sensitivas especiales debajo de la piel, que trabajan conjuntamente para hacer sentir calor, frío, etc. El hombre se comunica por medio del lenguaje, que es un sistema altamente desarrollado de palabras y símbolos que tienen significado; vive de acuerdo con un sistema económico en el cual los bienes y servicios se intercambian por otros de valor comparable y por medio de los cuales (al menos debería ser así) los participantes de este intercambio se benefician. Una organización, también es un sistema. Sus partes tienen nombres como depósito, producción, ventas, investigación, embarque, contabilidad, personal, etc. Estos componentes trabajan todos juntos para crear una utilidad que beneficie a los empleados y a los accionistas de la firma. Cada una de estas partes también es un sistema en sí mismo. El departamento de contabilidad, por ejemplo, puede consistir en cuentas por pagar o por cobrar, facturación, auditoría, etc.

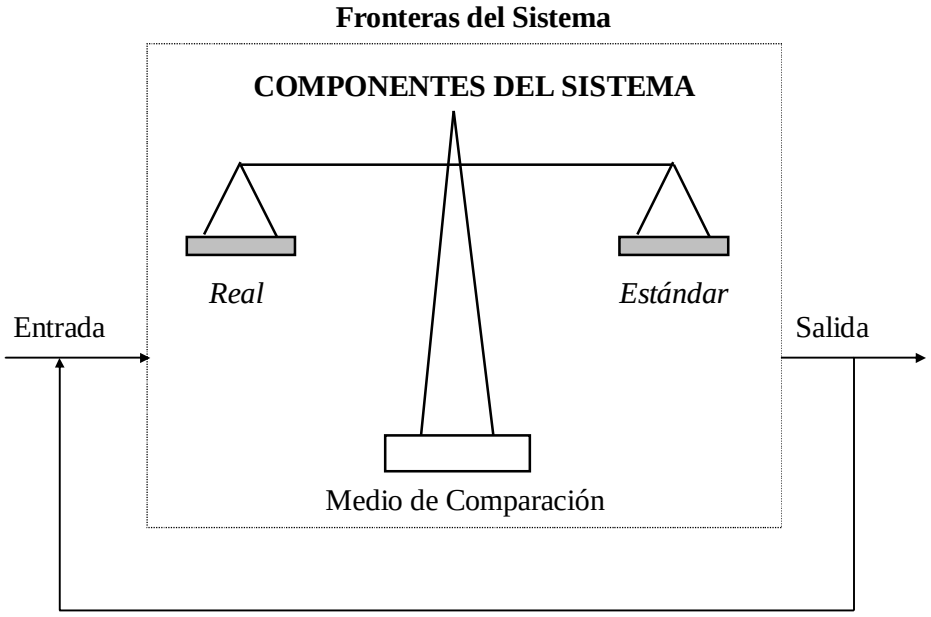
Los sistemas interactúan con sus medios ambientes, es decir, con elementos que se sitúan fuera de los límites del sistema. Las fronteras separan al sistema de su medio ambiente.

Los sistemas que interactúan con sus medios ambientes recibiendo entradas y produciendo salidas, se conocen con el nombre de sistemas abiertos en contraste con los que no interactúan con sus alrededores que se conocen como sistemas cerrados. Estos últimos existen solamente en forma conceptual.

El elemento de control se relaciona con la diferencia entre los sistemas abiertos o cerrados. Los sistemas trabajan mejor cuando operan dentro de niveles tolerables de rendimiento. Por ejemplo, la gente se siente normal cuando su temperatura corporal es de aproximadamente 36 grados. Una pequeña desviación probablemente no afectará mucho, sin embargo la diferencia se puede notar. Una gran variación, como tener 40 grados de fiebre, cambiará el funcionamiento corporal en forma drástica. El sistema orgánico tratará de corregir esas condiciones. En caso de no lograrlo, el resultado es fatal: el sistema muere. El ejemplo anterior, demuestra la importancia del control en los sistemas de todo tipo. Existen niveles aceptables de rendimiento, llamados estándares. Los rendimientos reales, se comparan contra los estándares. Las actividades que estén muy por encima o por debajo de estos estándares deben anotarse, de manera que se puedan estudiar y se hagan los ajustes necesarios. La información suministrada a través de la comparación de los resultados con los estándares, y el informe de los elementos de control sobre las diferencias, se denomina retroalimentación.

Entonces, los sistemas utilizan un modelo de control básico que consiste en:

1. Un **estándar** para rendimiento aceptable
2. Un método de medición del rendimiento **real**
3. Una forma de **comparar** el rendimiento real contra el estándar
4. Un método de **retroalimentación**



Los sistemas que pueden ajustar sus actividades a niveles aceptables continúan funcionando, los que no, se detienen.

El concepto de interacción dentro de un medio ambiente que caracteriza a los sistemas abiertos es esencial para el control. Por medio de la recepción de la entrada y la evaluación de la misma, un sistema puede determinar qué tan bien está operando. Si por ejemplo, un negocio produce objetos o servicios caros y/o bajos en calidad, la gente probablemente no los comprará. Las cifras de ventas bajas son la retroalimentación que le indica a la gerencia que necesita ajustar los productos y la manera de producción para mejorar el rendimiento y ajustarse a las expectativas.

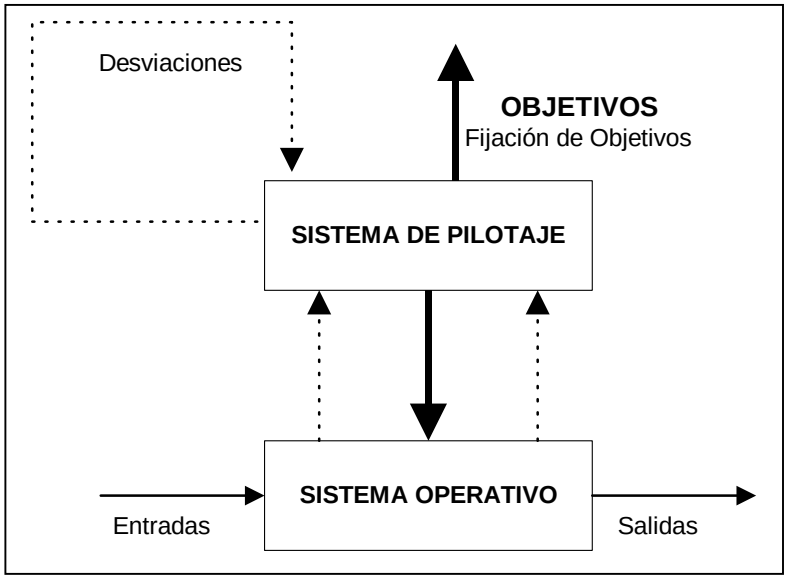
En contraste, los sistemas cerrados que no interactúan con el medio ambiente, sostienen su operación solamente durante el tiempo que tengan información adecuada y no necesiten nada del medio ambiente. Dado que esta condición no puede existir por mucho tiempo, no hay sistemas cerrados, sin embargo, el concepto es importante, porque demuestra un objetivo del diseño de un sistema y es que deben necesitar tan poca intervención externa como sea posible para mantener un rendimiento aceptable. La autorregulación y el autoajuste, por lo tanto, son objetivos del diseño en todos los medioambientes de sistemas.

1.2 El pilotaje

Un sistema se puede controlar por otro sistema que se denomina sistema de pilotaje. Por ejemplo:

> ***Se obtendrá más o menos calor según las regulaciones que se efectúen sobre la caldera, y durante más o menos tiempo según la cantidad de carbón. El operador que regula y controla el flujo de carbón que entra, constituye un sistema de pilotaje que, a través de sus acciones sobre el sistema físico (la caldera), busca satisfacer un objetivo (nivel de calor).***

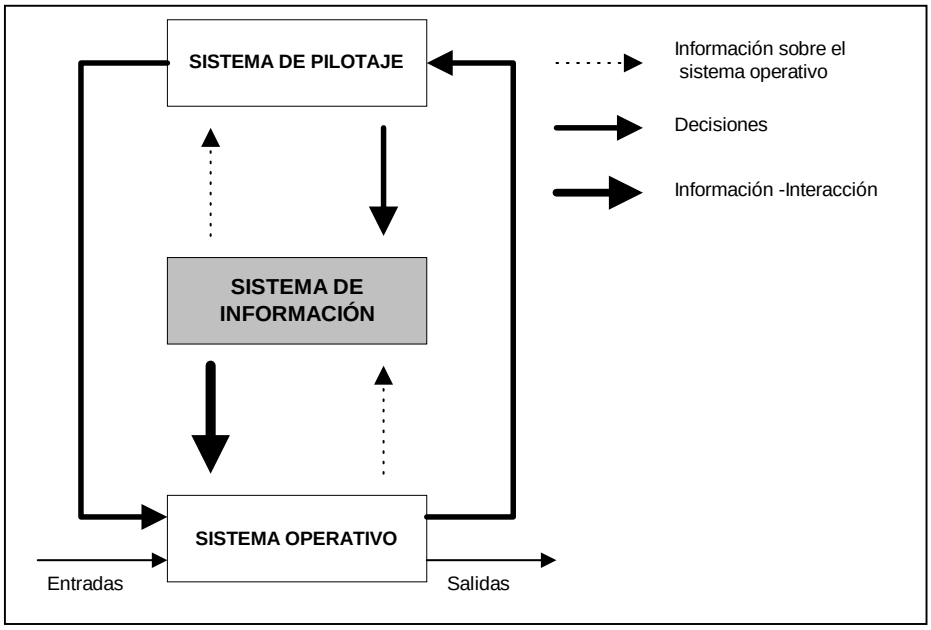
Además un sistema de gestión o sistema de pilotaje precede al propio pilotaje (a la regulación y al control) del sistema operativo, decidiendo el comportamiento de éste en función de los objetivos fijados. Este sistema se compone, por ejemplo de la dirección financiera, de la dirección comercial, de la dirección de producción, etc. Recibe del sistema operativo las informaciones sobre el estado del sistema las que le permiten medir las desviaciones con relación a los objetivos y reaccionar mediante decisiones o acciones sobre los procesos del sistema operativo o mediante la regulación de los flujos (por ejemplo fijación de cadencias de producción, decisión de lanzar una nueva gama de productos o modificar los precios de venta de determinados artículos, etc.).



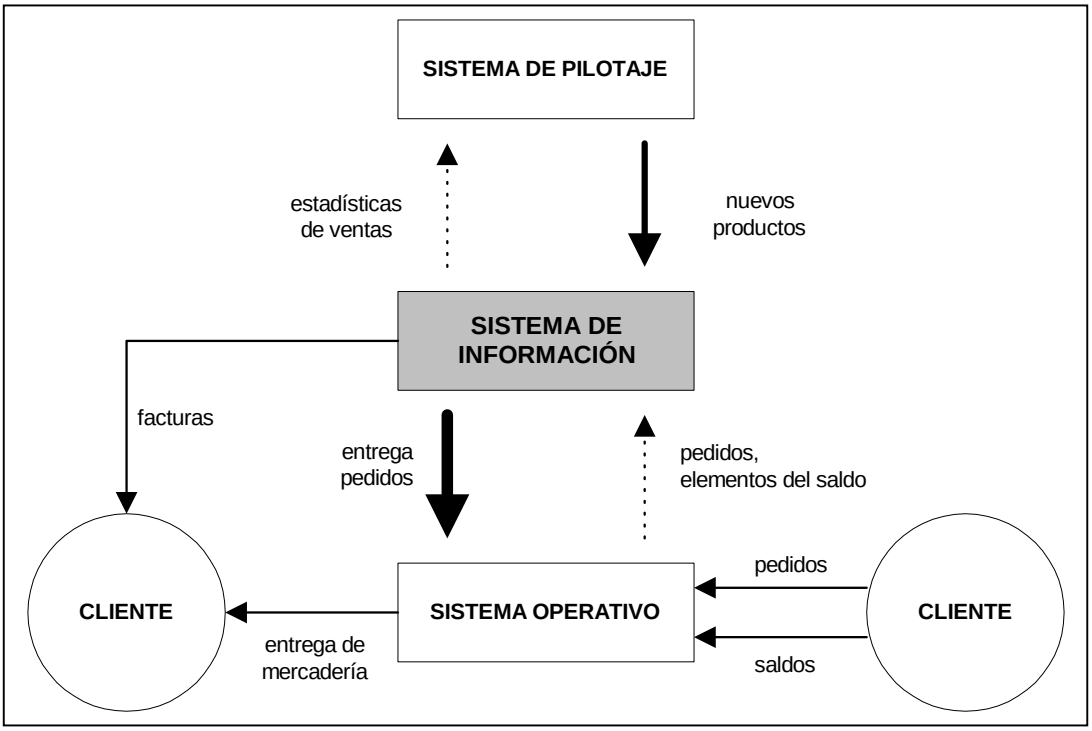
Como interfaz entre el sistema operativo y el sistema de pilotaje, toda organización evolucionada debe dotarse de un sistema de información.

1.3 Noción de sistema de información de una organización (SI)

El sistema de información se compone de diversos elementos (empleados, hardware, software, reglas de negocios, métodos etc.) encargados de almacenar y tratar las informaciones relativas al sistema operativo para ponerlas a disposición del sistema de pilotaje. Puede también recibir de éste decisiones destinadas a su propio pilotaje. Por último, puede emitir informaciones de interacción hacia el sistema operativo, es decir, puede ejercer su acción sobre el sistema operativo (por ejemplo, el sistema operativo sólo podrá remitir artículos al cliente si obtiene del sistema de información el dato de que existe stock del producto en el depósito).



Por Ejemplo:



El sistema de información contendrá imágenes formalizadas de los flujos del sistema operativo (pedidos, entregas, facturas, etc.) y datos contables utilizados para el control de la gestión. Existe, por una parte, una relación con el entorno interno (sistema operativo y sistema de pilotaje) y por otra con el entorno externo (clientes, proveedores, etc.). Estos dos entornos constituyen el Universo Exterior del sistema de información.

Otro aspecto del sistema de información es que se comporta como la memoria de la organización. Este tratamiento implica un aspecto estático:

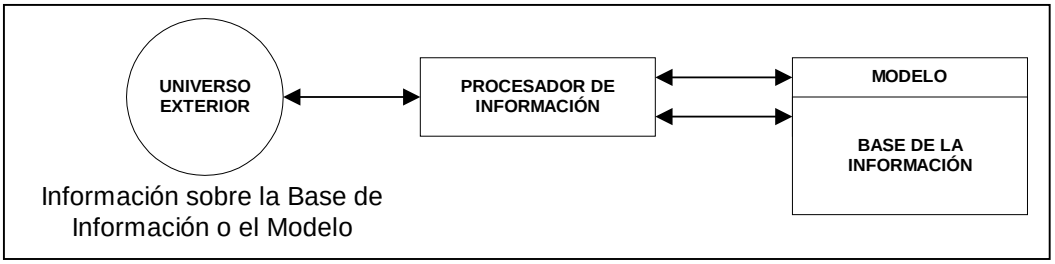
- Registrando hechos acontecidos en el universo exterior en un conjunto memorizado que se podría calificar como base de información.
- Registrando estructuras de datos, reglas y limitaciones del universo exterior, de manera formalizada en un conjunto memorizado que se podría calificar como modelo de dominio.

Implica igualmente un aspecto dinámico:

- Posibilidad de actualizar los datos memorizados en la base de información.
- Posibilidad (para un sistema adaptable) de cambiar las estructuras, reglas y limitaciones del modelo de dominio como consecuencia de los cambios acontecidos en el universo exterior y reflejo de aquellos.

Esta parte activa del sistema de información constituye el *procesador de información* (o subsistema que trata la información).

Cada hecho o evento, que surge en el universo exterior, constituye un mensaje para el procesador de información, mensaje que contiene una acción e informaciones. Con ayuda de las reglas que encuentra en el modelo, el procesador de información, interpreta el mensaje y procede a realizar las modificaciones en la base de información (o en el propio modelo) y/o devuelve un mensaje que da informaciones sobre la base del modelo. El procesador de información puede estar constituido por hombres y/o máquinas.



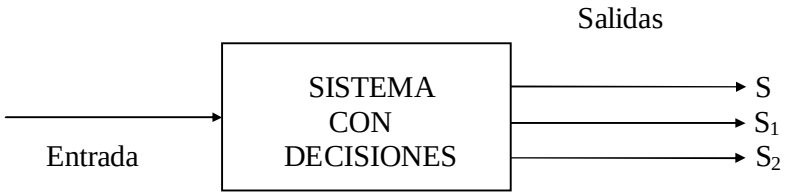
Todo sistema, contiene dos tipos de acciones, las conocidas con el nombre de programadas, y las decisiones. En un sistema, las acciones ***programadas*** son aquellas que determinan de manera única las salidas a partir de las entradas. Se dice entonces que el sistema está determinado.

En este caso, las entradas E determinan las salidas S en forma única, o sea que:

$$S = f(E)$$

Pero un sistema puede encontrarse en situación de información incompleta. En este caso, una misma entrada E puede conducir a varias salidas posibles S, S_1, S_2, etc.

La elección de la salida realmente realizada, se efectúa mediante una decisión. Por ejemplo, el conocimiento de la cantidad de un artículo en un almacén, no determina las cantidades a pedir al proveedor; entonces, quien se encarga de las compras deberá tomar una decisión.



Pueden intervenir en la decisión elementos no formalizables tales como la intuición, experiencia profesional, intereses personales, hábitos, etc.

Entonces, los procesos que en un sistema transforman las entradas en salidas, pueden incluir:

- Acciones programadas
- Opciones (decisiones)

Para que un sistema de información sea automatizable sus acciones deben ser programadas y deben existir reglas explicitables que permitan determinar sin equivocación (mediante reglas de transformación) de manera única las salidas a partir de las entradas.

Las decisiones no son formalizables y por consiguiente no son automatizables. La decisión pertenece al hombre o bien éste puede construir un modelo que haga el tratamiento de la decisión; por ejemplo, la decisión de reaprovisionamiento se puede efectuar con un modelo de gestión de almacén tal como:

Si el Stock ES MENOR QUE 500 ENTONCES pedir 2000

En este caso, la decisión se predefine con antelación y cada vez que el stock se sitúe por debajo de 500 se aplicará el modelo pidiendo 2000, que no es otra cosa que una acción programada puesto que el conocimiento del stock determina de manera única la cantidad a pedir.

1.4 Sistema automatizado de información (SAI)

Un SAI es un subsistema de un sistema de información en el que todas las transformaciones significativas de información se efectúan mediante máquinas de tratamiento automático de la información (hardware/software).

Permite la conservación y el tratamiento automático de los datos. Muchas pueden ser las razones que justifican la automatización de un SI, algunas de ellas pueden ser:

- Simplificación y mejora del trabajo administrativo (contabilidad, facturación, liquidación de sueldos, etc.) por la automatización de procedimientos repetitivos y tediosos de simple ejecución.

- Ayuda a la decisión: Si bien la decisión pertenece al hombre y no al sistema, le permite a aquel tomar decisiones con el máximo de información posible, ya que un sistema automatizado de información puede seleccionar a gran velocidad entre la masa de datos memorizados, las informaciones útiles para la toma de decisiones (ayuda al pilotaje).

1.4.1 Subsistemas funcionales del SAI

En un **SAI** el sistema de información, está constituido por:

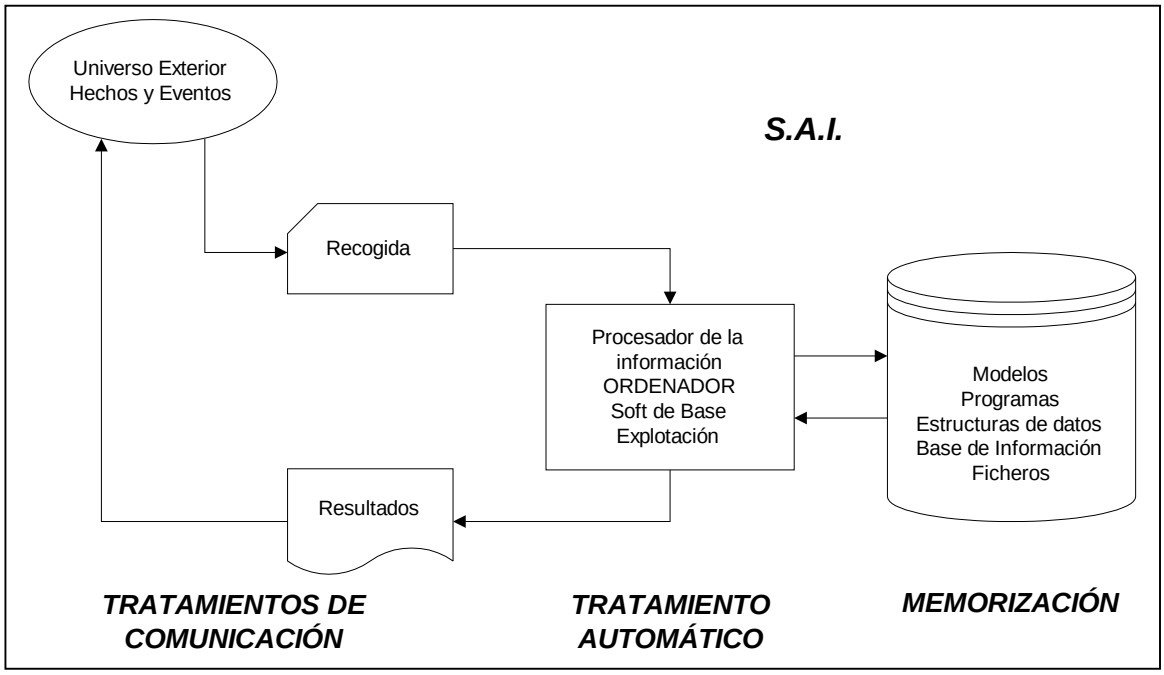
- el conjunto hardware/software y el personal de explotación
- unidades periféricas de comunicación, soportes de toma de datos, personal de recolección;
- el modelo y la base de información por medio de unidades periféricas de almacenamiento.

Los dos últimos elementos corresponden al universo exterior del procesador de información. Su trabajo consiste en buscar en el modelo los programas a ejecutar y las estructuras de reglas de negocio a cumplir y procede a la consulta o actualización de la base de información estableciendo el nexo entre ambos componentes exteriores. La respuesta o salida del procesador también va hacia el universo exterior.

Se puede descomponer en cuatro subsistemas funcionales:

- Dos subsistemas internos al SAI:
    - El tratamiento automático
    - La memorización
- Dos subsistemas de interfaz con el universo exterior (tratamiento de comunicación):
    - Recogida de datos
    - Salidas

En el diagrama pueden observarse los elementos de un SAI en relación con el Universo Exterior.



La memorización es la función de almacenamiento de las informaciones. Se memorizan programas y estructuras, y se almacena la información propiamente dicha. El almacenamiento se realiza sobre memorias externas. Es realizada por el hardware/software.

El tratamiento automático es la función consistente en manipular los datos memorizados o procedentes del exterior (recogida). Es realizada por el hardware/software. Se pone en marcha por los hechos acontecidos en el universo exterior que se interpretan como eventos que pueden ser portadores de informaciones por las que se procede a la recolección de datos.

1.4.2 Posición de un SAI en un SI

No hay que olvidarse que un SAI no es más que un subsistema del sistema de información. Este contiene partes manuales y partes automáticas o más exactamente superposición de partes manuales y automáticas. El procesador de información estará pues compuesto de hardware/software para sus componentes automáticos y de hombres y otros elementos que complementan la actividad.

La base de información puede estar almacenada parcialmente en memorias externas, quedando determinados ficheros en forma manual. Mucha información se guarda aún de manera impresa o documental. De igual forma, en el modelo, determinadas reglas y limitaciones estarán fijadas fuera de cualquier soporte informático.

Ciertos tratamientos del SI podrán incluir a la vez acciones automáticas y acciones manuales. Se dirá que se trata de tratamientos automatizados pero no enteramente automáticos. Así, la recolección de datos es un tratamiento automatizado en razón con su estrecha superposición con el TA pero no automático a causa de su importante componente manual (tecleado de los datos por el operador). Lógicamente, un tratamiento automático está

automatizado.

1.4.3 SAI integrados

El SAI de una organización se puede descomponer en subsistemas automatizados de información según sus dominios de actividad. Por ejemplo subsistemas de ventas, compras, contabilidad, personal, etc.

Cada subsistema comporta cuatro funciones:

- Recolección de datos
- Tratamiento automático
- Memorización
- Salida

Es importante estudiar las uniones entre subsistemas automatizados a los efectos de no efectuar repetición de tareas que le son comunes a dos o más subsistemas, compartiendo por ejemplo archivos, y explotándolo cada uno según sus requerimientos.

En algún momento, las entradas de datos deberán provenir del universo exterior. Otro subsistema puede utilizar a esos datos pero tomándolos como entrada interna.

Un SAI está integrado si una misma información no se recoge más que una vez en un punto del sistema y ésta queda disponible para todos los requerimientos en cualquier otra parte de la organización. La unión entre dos subsistemas del SAI integrado, se realiza mediante la memorización común, que permite salidas internas de uno y entradas internas en el otro.

1.5 Parametrización

Como todo sistema un SAI para subsistir debe ser adaptable. Un sistema de información automatizado no es la excepción y para poder resistir los cambios debe ser lo más adaptable posible a ciertas situaciones sin tener que modificarlo internamente. Ello es posible mediante la parametrización. A manera de ejemplo se presenta el caso anterior del pedido de stock (en el que se recurrió a un modelo para la decisión), por lo cual en vez de:

*Si STOCK < **50** ENTONCES pedir **2000***

Debería ser:

*Si STOCK < **x** ENTONCES pedir **y***

En donde para su ejecución deben instanciarse los valores de los parámetros ***x*** e ***y***.

La parametrización permite hacer el SAI más adaptable a distintos escenarios brindando de esta manera flexibilidad y facilidad a la evolución.

Universidad Nacional del Litoral

FACULTAD DE INGENIERÍA
Y CIENCIAS HÍDRICAS

Ingeniería en Informática

Ingeniería de Software I

TEMA II – La ingeniería de software

Introducción

Es imposible operar en el mundo moderno sin software. Las infraestructuras nacionales y los servicios públicos se controlan mediante sistemas basados en computadoras y la mayoría de los productos eléctricos y electrónicos incluyen una computadora y un software de control. La fabricación y la distribución industrial está completamente computarizada como el sistema financiero, los entretenimientos, la industria musical, los juegos, el cine, la televisión, todos usan software de manera intensiva. Esto da una idea de lo esencial que es el software para el funcionamiento de la sociedad.

Los sistemas de software son abstractos e intangibles. No están restringidos por las propiedades de los materiales ni regidos por leyes físicas ni por procesos de fabricación. Esto simplifica al software porque no tiene límites naturales a su potencial, pero precisamente debido a la falta de restricciones físicas los sistemas de software se vuelven rápidamente complejos, difíciles de entender y costosos de cambiar.

Hay muchos tipos diferentes de sistemas de software por lo que no tiene sentido buscar notaciones, métodos o técnicas universales ya que estos distintos tipos requieren de diferentes enfoques y no todas requieren de las mismas técnicas.

Algunos sistemas de software no son complejos, sino que son las aplicaciones altamente intrascendentes que son especificadas, construidas, mantenidas y utilizadas por la misma persona, habitualmente el programador aficionado o el desarrollador profesional que trabaja en solitario. Esto no significa que todos estos sistemas sean toscos o poco elegantes, ni se pretende quitar mérito a sus creadores. Tales sistemas tienden a tener un propósito limitado y un ciclo de vida muy corto. Uno puede permitirse tirarlos a la basura y reemplazarlos por software completamente nuevo en lugar de intentar reutilizarlos, repararlos o extender su funcionalidad. El desarrollo de estas aplicaciones es generalmente más tedioso que difícil y aprender a diseñarlas no es algo que resulte de interés. En lugar de esto, interesan mucho más los desafíos que plantea el desarrollo del ***software de dimensión industrial***. Entre estos se encuentran aplicaciones que exhiben un conjunto muy rico de comportamientos, como ocurre por ejemplo en sistemas que dirigen o son dirigidos por eventos del mundo físico, aplicaciones que mantienen la integridad de cientos o miles de registros de información mientras permiten actualizaciones y consultas concurrentes y sistemas para la gestión y control de entidades del mundo real, tales como controladores de tráfico aéreo, ferroviario o bien aparatos de bioingeniería. Los sistemas de software de esta clase tienden a tener un ciclo de vida largo y a lo largo del tiempo muchos usuarios llegan a depender de su funcionamiento correcto. La característica distintiva del software de dimensión industrial es que resulta sumamente difícil, sino imposible, para el desarrollador individual comprender todas las sutilidades de su diseño; dicho de otra manera, la complejidad de tales sistemas excede la capacidad intelectual humana. La complejidad es una propiedad esencial de todos los sistemas de software de gran tamaño: se puede dominar pero no eliminar. Existen personas de genio con habilidades extraordinarias que pueden hacer el trabajo de varias otras personas, pero no son el común; el mundo está poblado de genios solamente en forma dispersa y no hay razón para creer que la comunidad de la ingeniería de software posee una proporción extraordinariamente grande de ellos. Aunque hay un toque de genialidad en todos nosotros, en el dominio del software de dimensión industrial no se puede confiar siempre en la inspiración divina, por lo tanto, hay que considerar vías de mayor disciplina para dominar la complejidad.

2.1 LA COMPLEJIDAD DEL SOFTWARE

La complejidad del software se deriva de cuatro problemas:

a. Complejidad del dominio del problema. Los problemas que intentan resolver con el software conllevan a menudo elementos de complejidad ineludible en los que se encuentran innumerables requisitos que compiten entre sí e incluso se contradicen. Existe un desacoplamiento entre los usuarios de un sistema y sus desarrolladores: los usuarios suelen encontrar grandes dificultades al intentar expresar con precisión sus necesidades en una forma que los desarrolladores puedan comprender. Se le suma a esto la perspectiva distinta del mismo problema que tienen estos actores. Otra complicación es que los requisitos de un sistema de software cambian frecuentemente durante su desarrollo especialmente porque la mera existencia de un proyecto de desarrollo de software altera las reglas del problema original.

b. Dificultad de gestionar el desarrollo. La tarea fundamental del equipo de desarrollo es la de dar vida a una ilusión de simplicidad de esa complejidad externa que tienen los usuarios. Esta cantidad de trabajo exige la utilización de equipos de desarrolladores y de forma ideal se utilizará un equipo lo más pequeño posible. Un mayor número de miembros implica una comunicación más compleja y por lo tanto una coordinación más difícil, particularmente si el equipo está disperso geográficamente.

c. Flexibilidad del software. Una empresa de construcción de edificios normalmente no gestiona su propia explotación forestal para cosechar árboles de los que obtendría la madera o bien tener una acería en la obra para hacer componentes a medida para el nuevo edificio. El software ofrece la flexibilidad máxima por lo que un desarrollador puede expresar casi cualquier clase de abstracción. Esta flexibilidad resulta ser una propiedad que seduce increíblemente y suele empujar al desarrollador a construir por sí mismo prácticamente todos los bloques fundamentales sobre los que se apoyan estas abstracciones de más alto nivel.

d. Caracterización de problemas discretos. En una aplicación de gran tamaño puede haber ciento o miles de variables así como más de un flujo posible de control. El conjunto de todas estas variables, sus valores actuales y la dirección de ejecución y pila actual de cada proceso del sistema constituyen el estado actual de la aplicación. En contraste, los sistema analógicos son sistemas continuos, mientras que en el software se maneja de manera determinista, o sea que se asume como un sistema discreto. Consecuentemente esta es la razón primaria para probar a fondo los sistemas para precisamente encontrar las interacciones que pueden existir entre eventos que no fueron tenidos en cuenta al discretizar.

Cuanto más complejo es un sistema, más abierto está al derrumbamiento total. Un constructor pensaría raramente en añadir un nuevo nivel de sótano para cocheras a un edificio ya construido de cien pisos; hacer tal cosa sería muy costoso e indudablemente sería un invitación al fracaso. Contrastando con esta situación, los usuarios de sistema de software casi nunca piensan dos veces a la hora de solicitar cambios equivalentes aduciendo que *simplemente es cuestión de programar*. El fracaso en dominar la complejidad del software lleva a proyectos retrasados, que exceden el presupuesto y que son deficientes respecto a los requerimientos fijados. Esta situación

es una parte de lo denominado ***Crisis del software***, pero en realidad es una enfermedad que ha existido siempre que debe considerarse crónica y casi normal.

La ingeniería de software busca apoyar el desarrollo de software profesional o de dimensión industrial en lugar de la programación individual. Incluye técnicas que apoyan la especificación, el diseño y la evolución del programa, ninguno de los cuales son normalmente relevantes para el desarrollo de software personal. Este conjunto de técnicas, métodos y herramientas posibilita precisamente "dominar la complejidad".

El término ingeniería de software no se refiere solamente al software propiamente dicho (programa de cómputo), sino también a toda la documentación asociada y los datos de configuración requeridos para hacer que estos programas operen de manera correcta. Esta es una de las principales diferencias entre el desarrollo de software profesional y el del aficionado.

2.2 TIPOS DE PRODUCTOS DE SOFTWARE

Los ingenieros de software desarrollan productos (software que se puede vender a un cliente) de alguno de los siguientes tipos:

1. Productos genéricos: sistemas independientes que se producen por una organización de desarrollo y se venden en el mercado abierto a cualquier cliente que desee comprarlo, por ejemplo software para PC de tipo bases de datos, procesadores de texto, paquetes de dibujo, etc. También aplicaciones de propósitos específicos tales como sistemas de contabilidad, sistemas de manejo de clínicas, etc. En éstos, la organización que desarrolla el software controla la especificación del mismo.
2. Productos personalizados (a medida): son sistemas que están destinados para un cliente en particular. Un contratista desarrolla especialmente para ese cliente. En éstos, la organización compradora controla la especificación por lo que los desarrolladores de software trabajan siguiendo esa especificación.

Sin embargo, esta distinción suele hacerse difusa en la actualidad ya que cada vez más sistemas se construyen con un producto genérico como base que luego se adapta para ajustarse a los requerimientos de un cliente. Los sistemas ERP (Enterprise Resource Planning) como el sistema SAP son ejemplos de este caso. Un sistema grande y complejo se adapta a una compañía al incorporar la información acerca de las reglas y los procesos empresariales, los reportes, etc.

2.3 LA INGENIERÍA DE SOFTWARE

2.3.1 Evolución de la industria del software

El software es el factor decisivo a la hora de elegir entre varias soluciones informáticas disponibles para un problema dado, pero esto no ha sido siempre así. En los primeros años de la informática, el hardware tenía una importancia mucho mayor que en la actualidad. Su costo era comparativamente mucho más alto y su capacidad de almacenamiento y procesamiento, junto con su fiabilidad, era lo que limitaba las prestaciones de un determinado producto. El software

se consideraba como un simple añadido, a cuyo desarrollo se dedicaba poco esfuerzo y no se aplicaba ningún método sistemático. La programación era un arte de entrecasa y el desarrollo de software se realizaba sin ninguna planificación. La mayoría del software se desarrollaba y era utilizado por la misma persona u organización. La misma persona lo escribía, lo ejecutaba y, si fallaba, lo depuraba. Debido a que la movilidad en el trabajo era baja, los ejecutivos estaban seguros de que esa persona estaría allí cuando se encontrara algún error. Debido a este entorno personalizado del software, el diseño era un proceso implícito, realizado en la mente de alguien y la documentación normalmente no existía.

En este contexto, las empresas de informática se dedicaron a mejorar las prestaciones del hardware, reduciendo los costos y el consumo de los equipos, y aumentando la velocidad de cálculo y la capacidad de almacenamiento. Para ello, tuvieron que dedicar grandes esfuerzos a investigación y aplicaron métodos de ingeniería industrial al análisis, diseño, fabricación y control de calidad de los componentes de hardware. Como consecuencia de esto, el hardware se desarrolló rápidamente y la complejidad de los sistemas informáticos aumentó notablemente, necesitando de un software cada vez más complejo para su funcionamiento. Surgieron entonces las primeras casas de software, y el mercado se amplió considerablemente. Con ello aumentó la movilidad laboral, y con la marcha de un trabajador desaparecían las posibilidades de mantener o modificar los programas que éste había desarrollado. Al no utilizarse metodología alguna en el desarrollo del software, los programas contenían numerosos errores e inconsistencias, lo que obligaba a una depuración continua, incluso mucho después de haber sido entregados al cliente. Estas continuas modificaciones no hacían sino aumentar la inconsistencia de los programas, que se alejaban cada vez más de la corrección y se hacían prácticamente ininteligibles. Los costos se disparaban y frecuentemente era más rápido (y por tanto más barato) empezar de nuevo desde cero, desechando todo el trabajo anterior, que intentar adaptar un programa preexistente a un cambio de los requisitos. Sin embargo, los nuevos programas no estaban exentos de errores ni de futuras modificaciones, con lo que la situación volvía a ser la misma. Había comenzado la crisis del software.

Hoy en día, la distribución de costos en el desarrollo de sistemas informáticos ha cambiado drásticamente. El software, en lugar del hardware, es el elemento principal del costo. Esto ha hecho que las miradas de los ejecutivos de las empresas se centren en el software y a que se formulen las siguientes preguntas:

¿Por qué lleva tanto tiempo terminar los desarrollos?
¿Por qué es tan elevado el costo?
¿Por qué no es posible encontrar todos los errores antes de entregar el software al cliente?
¿Por qué resulta tan difícil constatar el progreso conforme se desarrolla el software?

Estas y otras muchas preguntas son una manifestación del carácter del software y de la forma en que se desarrolla y han llevado a la aparición y la adopción paulatina de la ingeniería del software.

2.3.2 Características del software

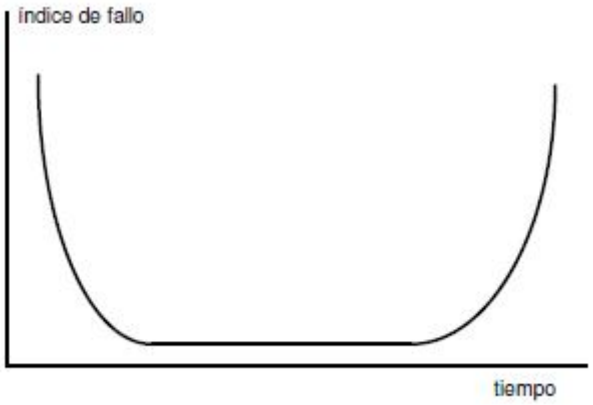
Algunas definiciones de software podrían ser:

- Instrucciones de computadora que cuando se ejecutan proporcionan la función y el comportamiento deseado,
- Estructuras de datos que facilitan a los programas manipular adecuadamente la información, y
- Documentos que describen la operación y el uso de los programas.

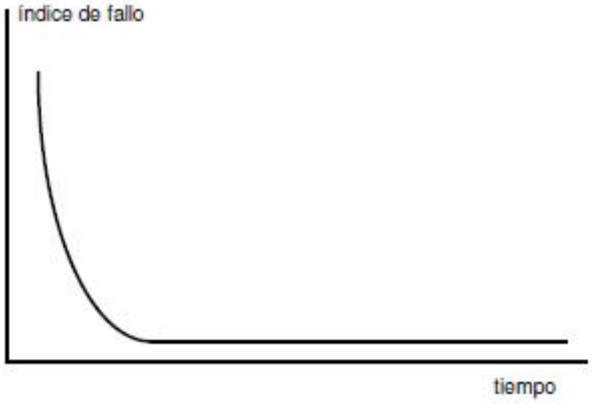
Por tanto, el software incluye no sólo los programas de computadora, sino también las estructuras de datos que manejan esos programas y toda la documentación que debe acompañar al proceso de desarrollo, mantenimiento y uso de dichos programas. Según esto, el software se diferencia de otros productos que los hombres puedan construir por su propia naturaleza lógica. En el desarrollo del hardware, el proceso creativo (análisis, diseño, construcción, prueba) se traduce finalmente en una forma material, en algo físico. Por el contrario, el software es inmaterial y por ello tiene unas características completamente distintas al hardware. Entre ellas podemos citar:

1. El software se desarrolla, no se fabrica en sentido estricto. Existen similitudes entre el proceso de desarrollo del software y el de otros productos industriales, como el hardware. En ambos casos existen fases de análisis, diseño y desarrollo o construcción, y la buena calidad del producto final se obtiene mediante un buen diseño. Sin embargo, en la fase de producción del software pueden producirse problemas que afecten a la calidad y que no existen, o son fácilmente evitables, en el caso del hardware. Por otro lado, en el caso de producción de hardware a gran escala, el costo del producto acaba dependiendo exclusivamente del costo de los materiales empleados y del propio proceso de producción, reduciéndose la repercusión en el costo de las fases de ingeniería previas. En el software, el desarrollo es una más de las labores de ingeniería, y la producción a gran o pequeña escala no influye en el impacto que tiene la ingeniería en el costo, al ser el producto inmaterial. Por otro lado, la replicación del producto (lo que sería equivalente a la producción del producto hardware) no presenta problemas técnicos, y no se requiere un control de calidad individualizado. Los costos del software se encuentran en la ingeniería (incluyendo en ésta el desarrollo), y no en la producción, y es ahí donde hay que incidir para reducir el costo final del producto.

2. El software no se estropea. Se pueden comparar las curvas de índices de fallos del hardware y el software en función del tiempo. En el caso del hardware (figura siguiente), se tiene la llamada curva de bañera, que indica que el hardware presenta relativamente muchos fallos al principio de su vida.

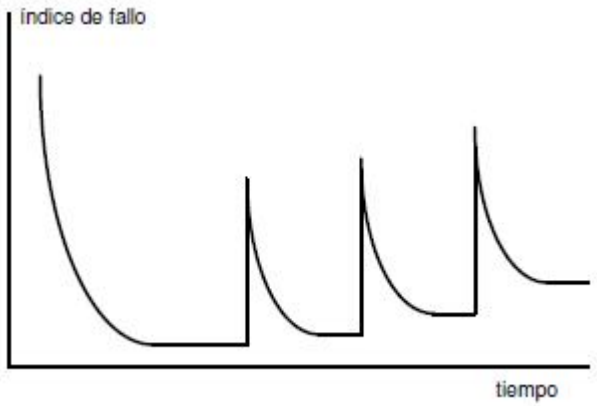


Estos fallos son debidos fundamentalmente a defectos de diseño o a la baja calidad inicial de la fase de producción. Una vez corregidos estos defectos, la tasa de fallos cae hasta un nivel estacionario y se mantiene así durante un cierto periodo de tiempo. Posteriormente, la tasa de fallos vuelve a incrementarse debido al deterioro de los componentes, que van siendo afectados por la suciedad, vibraciones y la influencia de muchos otros factores externos. Llegados a este punto, podemos sustituir los componentes defectuosos o todo el sistema por otros nuevos, y la tasa de fallos vuelve a situarse en el nivel estacionario. El software no es susceptible a los factores externos que hacen que el hardware se estropee. Por tanto la curva debería seguir la forma de la figura siguiente:



Inicialmente la tasa de fallos es alta, debido a la presencia de errores no detectados durante el desarrollo (los denominados errores latentes). Una vez corregidos estos errores, la tasa de fallos debería alcanzar el nivel estacionario y mantenerse ahí indefinidamente. Pero esto no es más que una simplificación del modelo real de fallos de un producto software. Durante su vida, el software sufre cambios, debidos al mantenimiento. El mantenimiento puede deberse a la corrección de errores latentes o a cambios en los requisitos iniciales del producto. Conforme se hacen cambios es bastante probable que se introduzcan nuevos errores, con lo que se producen picos en la curva de fallos. Estos errores pueden corregirse, pero el efecto de los sucesivos cambios hace que el producto se aleje cada vez más de las especificaciones iniciales de acuerdo a las cuales fue desarrollado, conteniendo cada vez más errores latentes. Además, con mucha frecuencia se solicita - y se emprende - un nuevo cambio antes de haber conseguido corregir todos los errores producidos por el cambio anterior. Por estas razones, el nivel

estacionario que se consigue después de un cambio es algo superior al que el que había antes de efectuarlo (figura siguiente), degradándose poco a poco el funcionamiento del sistema. Se puede decir entonces que el software no se estropea, pero se **deteriora**.



Además, cuando un componente software se deteriora, no se puede sustituir por otro, como en el caso del hardware ya que no existen piezas de repuesto. Cada fallo del software indica un fallo en el diseño o en el proceso mediante el cual se transformó el diseño en programación. La solución no es sustituir el componente defectuoso por otro (que sería idéntico y contendría los mismos errores) sino un nuevo diseño y desarrollo del producto. Por tanto, el mantenimiento del software tiene una complejidad mucho mayor que el mantenimiento del hardware.

3. La mayoría del software se construye a medida. El diseño de hardware se realiza, en gran medida, en base de componentes digitales existentes, cuyas características se comprueban en un catálogo y que han sido exhaustivamente probados por el fabricante y los usuarios anteriores. Estos componentes cumplen unas especificaciones claras y tienen unas interfaces definidas. El caso del software es totalmente distinto. No existen catálogos de componentes, y aunque determinados productos como sistemas operativos, editores, entornos de ventanas y bases de datos se venden en grandes ediciones, la mayoría del software se fabrica a medida, siendo la reutilización muy baja. Se puede comprar software ya desarrollado, pero sólo como unidades completas, no como componentes que pueden ser reensamblados para construir nuevos programas. Esto hace que el impacto de los costos de ingeniería sobre el producto final sea muy elevado, al dividirse entre un número de unidades producidas muy pequeño. Se ha escrito mucho sobre reutilización del software, y han sido muchos los intentos para conseguir aumentar el nivel de reutilización, normalmente con poco éxito. Uno de los ejemplos de reutilización más típicos, que ha venido usándose desde hace tiempo, son las bibliotecas. Durante los años sesenta se empezaron a desarrollar bibliotecas de subrutinas científicas, reutilizables en una amplia gama de aplicaciones científicas y de ingeniería. La mayor parte de los lenguajes modernos incluyen bibliotecas de este tipo, así como otras para facilitar la entrada/salida en los entornos de ventanas. Sin embargo esta aproximación no puede aplicarse fácilmente a otros problemas también de uso muy frecuente, como puede ser la búsqueda de un elemento en una estructura de datos, debido a la gran variación que existe en cuanto a la organización interna de estas estructuras y en la composición de los datos que contienen. Existen algoritmos para resolver estos problemas, pero no queda más

remedio que programarlos una y otra vez, adaptándolos a cada situación particular. Un nuevo intento de conseguir la reutilización se produjo con la utilización de técnicas de programación estructurada y modular. Sin embargo, se dedica por lo general poco esfuerzo al diseño de módulos lo suficientemente generales para ser reutilizables, y en todo caso, no se documentan ni se difunden todo lo que sería necesario para extender su uso, con lo que la tendencia habitual es diseñar y programar módulos muy semejantes una y otra vez. La programación estructurada permite diseñar programas con una estructura más clara, y que, por tanto, sean más fáciles de entender. Esta estructura interna más clara y estudiada, permite la reutilización de módulos dentro de los programas, o incluso dentro del proyecto que se está desarrollando, pero la reutilización de código en proyectos diferentes es muy baja. La tendencia actual para conseguir la reutilización es el uso de técnicas orientadas a objetos, que permiten la programación por especialización. Los objetos, que encapsulan tanto datos como los procedimientos que los manejan - los métodos -, haciendo los detalles de implementación invisibles e irrelevantes a quien los usa, disponen de interfaces claras, los errores cometidos en su desarrollo pueden ser depurados sin que esto afecte a la corrección de otras partes del código y pueden ser heredados y reescritos parcialmente, haciendo posible su reutilización aún en situaciones no contempladas en el diseño inicial. El software permite aplicaciones muy diversas, pero en todas ellas podemos encontrar algo en común: el objetivo es que el software desempeñe una determinada función, y además, debe hacerlo cumpliendo una serie de requisitos. Esos pueden ser muy variados: corrección, fiabilidad, respuesta en un tiempo determinado, facilidad de uso, bajo costo, etc., pero siempre existen y no podemos olvidarnos de ellos a la hora de desarrollar el software.

2.3.3 Problemas del software

Se mencionó la crisis del software. Sin embargo, por crisis se entiende normalmente un estado pasajero de inestabilidad, que tiene como resultado un cambio de estado del sistema o una vuelta al estado inicial, en caso de que se tomen las medidas para superarla. Teniendo en cuenta esto, el software, más que padecer una crisis podríamos decir que padece una enfermedad crónica. Los problemas que surgieron cuando se empezó a desarrollar software de una cierta complejidad siguen existiendo actualmente, sin que se haya avanzado mucho en los intentos de solucionarlos. Estos problemas son causados por las propias características del software y por los errores cometidos por quienes intervienen en su producción. Entre ellos podemos citar:

- <u>La planificación y la estimación de costos son muy imprecisas</u>
  A la hora de abordar un proyecto de una cierta complejidad, sea en el ámbito que sea, es frecuente que surjan imprevistos que no estaban recogidos en la planificación inicial, y como consecuencia de estos imprevistos se producirá una desviación en los costos del proyecto. Sin embargo, en el desarrollo de software lo más frecuente es que la planificación sea prácticamente inexistente, y que nunca se revise durante el desarrollo del proyecto. Sin una planificación detallada es totalmente imposible hacer una estimación de costos que tenga alguna posibilidad de cumplirse, y tampoco se pueden identificar las tareas conflictivas que pueden desviarnos de los costos previstos. Entre las causas de este problema se puede citar:

- No se recogen datos sobre el desarrollo de proyectos anteriores, con lo que no se acumula experiencia que pueda ser utilizada en la planificación de nuevos proyectos.
- Los administradores de proyectos no están especializados en la producción de software. Tradicionalmente, los responsables del desarrollo del software han sido ejecutivos de nivel medio y alto sin conocimientos de informática, siguiendo el principio de que *"un buen gestor (administrador, líder de proyecto, Project Manager) puede gestionar cualquier proyecto"*. Esto es cierto, pero no cabe duda de que también es necesario conocer las características específicas del software, aprender las técnicas que se aplican en su desarrollo y conocer una tecnología que evoluciona continuamente.

- La productividad es baja
  Los proyectos de software tienen, por lo general, una duración mucho mayor a la esperada. Como consecuencia de esto los costos se disparan y la productividad y los beneficios disminuyen. Uno de los factores que influyen en esto, es la falta de unos propósitos claros o realistas a la hora de comenzar el proyecto. La mayoría del software se desarrolla a partir de especificaciones ambiguas o incorrectas, y no existe apenas comunicación con el cliente hasta la entrega del producto. Debido a esto son muy frecuentes las modificaciones de las especificaciones sobre la marcha o los cambios de última hora, después de la entrega al cliente. No se realiza un estudio detallado del impacto de estos cambios y la complejidad interna de las aplicaciones crece hasta que se hacen virtualmente imposibles de mantener y cada nueva modificación, por pequeña que sea, es más costosa, y puede provocar el fallo de todo el sistema. Debido a la falta de documentación sobre cómo se ha desarrollado el producto o a que las sucesivas modificaciones - también indocumentadas - han desvirtuado totalmente el diseño inicial, el mantenimiento de software puede llegar a ser una tarea imposible de realizar, pudiendo llevar más tiempo el realizar una modificación sobre el programa ya escrito que analizarlo y desarrollarlo entero de nuevo.

- La calidad es mala
  Como consecuencia de que las especificaciones son ambiguas o incluso incorrectas, y de que no se realizan pruebas exhaustivas, el software contiene numerosos errores cuando se entrega al cliente. Estos errores producen un fuerte incremento de costos durante el mantenimiento del producto, cuando ya se esperaba que el proyecto estuviese acabado. Sólo recientemente se ha empezado a tener en cuenta la importancia de la prueba sistemática y completa, y han empezado a surgir conceptos como la fiabilidad y la garantía de calidad.

- El cliente queda insatisfecho
  Debido al poco tiempo e interés que se dedican al análisis de requisitos y a la especificación del proyecto, a la falta de comunicación durante el desarrollo y a la existencia de numerosos errores en el producto que se entrega, los clientes suelen quedar muy poco satisfechos de los resultados. Consecuencia de esto es que las aplicaciones tengan que ser diseñadas y desarrolladas de nuevo, que nunca lleguen a utilizarse o que se produzca con frecuencia un cambio de proveedor a la hora de abordar un nuevo proyecto.

2.3.4 Definición de ingeniería de software

El desarrollo de sistemas de software complejos no es una actividad trivial, que pueda abordarse sin una preparación previa. El considerar que un proyecto de desarrollo de software podía abordarse como cualquier otro ha llevado a una serie de problemas que limitan nuestra capacidad de aprovechar los recursos que el hardware pone a nuestra disposición. Los problemas tradicionales en el desarrollo de software no van a desaparecer de la noche a la mañana, pero identificarlos y conocer sus causas es el único método que nos puede llevar hacia su solución. No existe una fórmula mágica para solucionar estos problemas, pero combinando métodos aplicables a cada una de las fases del desarrollo de software, construyendo herramientas para automatizar estos métodos, utilizando técnicas para garantizar la calidad de los productos desarrollados y coordinando todas las personas involucradas en el desarrollo de un proyecto, podremos avanzar mucho en la solución de estos problemas. De ello se encarga la disciplina llamada Ingeniería del Software.

Ésta disciplina de la ingeniería se interesa por todos los aspectos de la producción de software, desde las primeras etapas de la especificación del sistema hasta el mantenimiento del sistema después que se pone en operación.

La ingeniería busca seleccionar el método más adecuado para un conjunto de circunstancias y de esta manera, un acercamiento al desarrollo más creativo y menos formal no se considera muy efectivo (salvo en ciertas situaciones). El enfoque sistemático que usa la ingeniería de software se conoce como ***Proceso de software***. Un proceso de software es una secuencia de actividades que conducen a la elaboración de un producto de software. Existen cuatro actividades fundamentales comunes a todos los procesos y son:

1. **Especificación** del software, donde clientes e ingenieros definen las funcionalidades del software que se producirá y las restricciones de su operación.
2. **Diseño e implementación** del software donde se diseña y programa cumpliendo con las especificaciones.
3. **Validación** del software, donde se verifica el software para asegurar que sea lo que el cliente requiere.
4. **Evolución** del software, donde se modifica el software para reflejar los requerimientos cambiantes del cliente y el mercado.

Diferentes tipos de sistemas necesitan distintos procesos de desarrollo. Por ejemplo, un software en tiempo real de una aeronave se debe especificar por completo antes de comenzar el desarrollo, mientras que en un sistema de comercio electrónico, la especificación y el programa por lo general se desarrollan en conjunto.

Comparación con otras ciencias afines

Las ciencias de la computación se interesan por las teorías y los métodos que subyacen en las computadoras y los sistemas de software, en tanto que la ingeniería de software se preocupa por los asuntos prácticos de la producción de software. El conocimiento de las ciencias de la computación es esencial para los ingenieros de software (como la física para un ingeniero

electrónico). La ingeniería en sistemas se interesa por todos los aspectos del desarrollo y la evolución de complejos sistemas donde el software tiene un papel principal. Por lo tanto la ingeniería en sistemas se preocupa por el desarrollo del hardware, el diseño de políticas y procesos, la implantación del sistema así como por la ingeniería de software. Los ingenieros en sistemas intervienen en la especificación del sistema definiendo su arquitectura global y están menos preocupados por la ingeniería de los componentes de ese sistema.

La ingeniería de software es un enfoque sistemático para la producción de software considerando elementos prácticos tales como costos y planificación. Éste enfoque sistemático varía mucho dependiendo de la organización que desarrolla el software y el tipo de software por lo que no existen métodos y técnicas universales de ingeniería de software que sean adecuados para todos los sistemas y organizaciones. Uno de los factores más significativos en la determinación de qué método o técnica aplicar es el tipo de aplicación. Estos tipos pueden ser:

1. **Aplicaciones independientes**. Se trata de sistemas de aplicación que corren en una PC local e incluyen toda la funcionalidad necesaria sin necesidad de conectarse a una red.
2. **Aplicaciones interactivas basadas en transacción**. Son las aplicaciones que se ejecutan remotamente y los usuarios acceden desde su propia PC o terminal. Se incluyen las aplicaciones WEB como comercio electrónico o sistemas empresariales.
3. **Sistemas de control embebido**. Sistemas de control de software que regulan y gestionan dispositivos de hardware. Son los más numerosos. Por ejemplo los software de celulares, los frenos ABS, una video, etc.
4. **Sistema de procesamiento por lotes**. Batch. Procesan gran cantidad de entradas individuales para crear salidas correspondientes. Son por ejemplo los sistemas de facturación periódica como los de facturas telefónicas, impuestos, sueldos, etc.
5. **Sistemas de entretenimiento**. Sistemas de uso personal. Juegos.
6. **Sistemas para modelado y simulación**. Sistemas que desarrollan científicos o ingenieros para modelar procesos o situaciones físicas que incluyen muchos objetos separados interactuantes. Son computacionalmente intensivos y requieren muchos recursos de hardware.
7. **Sistemas de adquisición de datos**. Son sistemas que desde su entorno recopilan datos mediante sensores y los envían para su procesamiento a otro sistema. Interactúan con sensores y se instalan en ambientes hostiles.

Si bien cada situación tal vez amerite una metodología distinta, existen fundamentos de la ingeniería de software que se aplican a todos los sistemas de software:

- Deben llevarse a cabo usando un proceso de desarrollo administrado y comprendido. La organización que diseña el software necesita planear el proceso de desarrollo así como tener ideas claras acerca de lo que se producirá y el tiempo en que estará terminado. Se usan diferentes procesos para distintos tipos de software.
- La confiabilidad y el desempeño son importantes para todos los tipos de sistemas. El software tiene que comportarse como se espera, sin fallas y cando se requiera estar disponible. Debe ser seguro en su operación y contra ataques externos. Debe desempeñarse de manera eficiente y no desperdiciar recursos.
- Es importante comprender y gestionar la especificación y los requerimientos del

software (lo que debe hacer). Debe conocerse qué esperan de él los diferentes clientes y usuarios del sistema y gestionar sus expectativas para entregar un sistema útil dentro de la fecha y presupuesto.

- Tiene que usar de manera tan efectiva como sea posible los recursos existentes. Donde sea adecuado se debe reutilizar el software ya desarrollado en vez de diseñar uno nuevo.

La ingeniería del software abarca un conjunto de tres elementos clave: métodos, herramientas y procedimientos, que faciliten al project manager (PM, líder de proyecto) el control del proceso de desarrollo y suministren a los implementadores bases para construir de forma productiva software de alta calidad.

Los métodos indican cómo construir técnicamente el software, y abarcan una amplia serie de tareas que incluyen la planificación y estimación de proyectos, el análisis de requisitos, el diseño de estructuras de datos, programas y procedimientos, la codificación, las pruebas y el mantenimiento. Los métodos introducen frecuentemente una notación específica para la tarea en cuestión y una serie de criterios de calidad.

Las herramientas proporcionan un soporte automático o semiautomático para utilizar los métodos. Existen herramientas automatizadas para cada una de las fases vistas anteriormente, y sistemas que integran las herramientas de cada fase de forma que sirven para todo el proceso de desarrollo. Estas herramientas se denominan CASE (Computer Assisted Software Engineering).

Los procedimientos definen la secuencia en que se aplican los métodos, los documentos que se requieren, los controles que permiten asegurar la calidad y las directrices que permiten a los PM evaluar los progresos.

2.4 MODELOS DE PROCESO DE SOFTWARE (ciclo de vida)

No existe un proceso ***ideal***. La mayoría de las organizaciones han diseñado sus propios procesos de desarrollo de software. Los procesos han evolucionado para beneficiarse de las capacidades de los integrantes de la organización y de las características específicas de los sistemas que se están desarrollando. Para algunos sistemas críticos, se requiere de un proceso de desarrollo muy estructurado. Para los sistemas empresariales con requerimientos rápidamente cambiantes, es probable que sea más efectivo un proceso menos formal y flexible. En ocasiones los procesos de software se clasifican como dirigidos por un plan o como procesos ágiles. Los primeros son aquellos donde las actividades del proceso se planean por anticipado y el avance se mide contra dicho plan. En los ágiles la planeación es incremental y es más fácil de modificar el proceso para reflejar los requerimientos cambiantes del cliente. Cada enfoque es adecuado para diferentes tipos de software por lo que se requiere encontrar un equilibrio entre procesos dirigidos por un plan y procesos ágiles.

Por ciclo de vida, se entiende la sucesión de etapas por las que pasa el software desde que un nuevo proyecto es concebido hasta que se deja de usar. Cada una de estas etapas lleva asociada una serie de tareas o actividades que deben realizarse, y una serie de documentos (en sentido amplio: software) que serán la salida de cada una de estas fases y servirán de entrada en la

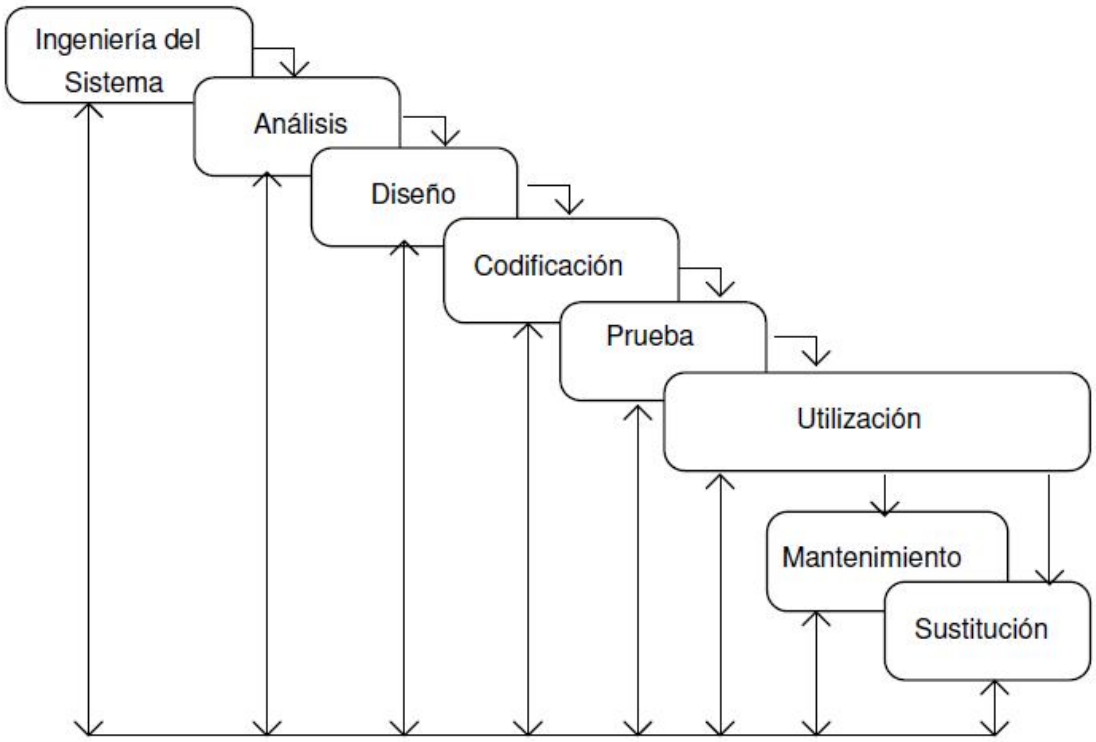
fase siguiente. Existen diversos modelos de proceso de software o de ciclo de vida, es decir, diversas formas de ver el proceso de desarrollo de software, y cada uno de ellos va asociado a un paradigma de la ingeniería del software, es decir, a una serie de métodos, herramientas y procedimientos que debemos usar a lo largo de un proyecto. Los modelos que se verán son:

- Modelo en cascada
- Desarrollo incremental o evolutivo
- Ingeniería de software orientada a la reutilización.

2.4.1 El modelo en cascada (waterfall) o ciclo de vida clásico

Este paradigma es el más antiguo de los empleados en la ingeniería de software y se desarrolló a partir del ciclo convencional de una ingeniería. No hay que olvidar que la ingeniería de software surgió como copia de otras ingenierías, especialmente de la del hardware, para dar solución a los problemas más comunes que aparecían al desarrollar sistemas de software complejos.

Es un ciclo de vida en sentido amplio, que incluye no sólo las etapas de ingeniería sino toda la vida del producto: las pruebas, el uso (la vida útil del software) y el mantenimiento, hasta que llega el momento de sustituirlo.



El ciclo de vida en cascada exige un enfoque sistemático y secuencial del desarrollo de software, que comienza en el nivel de la ingeniería de sistemas y avanza a través de fases secuenciales sucesivas. Estas fases son las siguientes:

Etapa 1: Ingeniería y análisis del sistema

El software es siempre parte de un sistema mayor, por lo que siempre va a interrelacionarse con otros elementos, ya sea hardware, otro sistema de software o personas. Por esto, el primer paso del ciclo de vida de un proyecto consiste en un análisis de las características y el comportamiento del sistema del cual el software va a formar parte. En el caso de que se desee construir un sistema nuevo, por ejemplo un sistema de control, se debe analizar cuáles son los requisitos y la función del sistema, y luego asignarle un subconjunto de estos requisitos al software. En el caso de un sistema ya existente (por ejemplo si se quiere informatizar una empresa) se debe analizar el funcionamiento de la misma, - las operaciones que se llevan a cabo en ella -, y asignarle al software aquellas funciones que se van a automatizar. La ingeniería del sistema comprende, por tanto, los requisitos globales a nivel del sistema, así como una cierta cantidad de análisis y de diseño a nivel superior, es decir sin entrar en mucho detalle.

Etapa 2: Análisis de requisitos del software

El análisis de requisitos debe ser más detallado para aquellos componentes del sistema que se van a implementar mediante software. El ingeniero del software debe comprender cuáles son los datos que se van a manejar, cuál va a ser la función que tiene que cumplir el software, cuáles son las interfaces requeridas y cuál es el rendimiento que se espera lograr. Los requisitos, tanto del sistema como del software deben documentarse y revisarse con el cliente.

Etapa 3: Diseño

El diseño se aplica a cuatro características distintas del software:

- la estructura de los datos
- la arquitectura de las aplicaciones
- la estructura interna de los programas
- las interfaces

El diseño es el proceso que traduce los requisitos en una representación del software de forma que pueda conocerse la arquitectura, funcionalidad e incluso la calidad del mismo antes de comenzar la codificación. Al igual que el análisis, el diseño debe documentarse y forma parte de la configuración del software (el control de configuraciones es lo que nos permite realizar cambios en el software de forma controlada y no traumática para el cliente).

Etapa 4: Codificación

La codificación consiste en la traducción del diseño a un formato que sea legible para la máquina. Si el diseño es lo suficientemente detallado, la codificación es relativamente sencilla, y puede hacerse - al menos en parte - de forma automática, usando generadores de código. Se puede observar que estas primeras fases del ciclo de vida consisten básicamente en una traducción: del análisis del sistema, los requisitos, la función y la estructura de este se traducen a un documento de análisis del sistema que está formado en parte por diagramas y en parte por descripciones en lenguaje natural. En el análisis de requisitos se profundiza en el estudio del componente software del sistema y esto se traduce a un documento, también formado por diagramas y descripciones en lenguaje natural. En el diseño, los requisitos del software se traducen a una serie de diagramas que representan la estructura del sistema software, de sus datos, de sus programas y de sus interfaces. Por

último, en la codificación se traducen estos diagramas de diseño a un lenguaje fuente, que luego se traduce - se compila - para obtener un programa ejecutable.

Etapa 5: Prueba

Una vez que se tiene el programa ejecutable, comienza la fase de pruebas. El objetivo es comprobar que no se hayan producido errores en alguna de las fases de traducción anteriores, especialmente en la codificación. Para ello deben probarse todas las sentencias, no sólo los casos normales y todos los módulos que forman parte del sistema.

Etapa 6: Utilización

Una vez superada la fase de pruebas, el software se entrega al cliente y comienza la vida útil del mismo. La fase de utilización se solapa con las posteriores - el mantenimiento, evolución y la sustitución - y dura hasta que el software, ya reemplazado por otro, deje de utilizarse.

Etapa 7: Mantenimiento

El software sufrirá cambios a lo largo de su vida útil. Estos cambios pueden ser debidos a tres causas:

- Que, durante la utilización, el cliente detecte errores en el software (los errores latentes).
- Que se produzcan cambios en alguno de los componentes del sistema informático: por ejemplo cambios en la máquina, en el sistema operativo o en los periféricos.
- Que el cliente requiera modificaciones funcionales (normalmente ampliaciones) no contempladas 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.
El modelo en cascada, a pesar de ser lineal, contiene flujos que permiten la vuelta atrás. Así, desde el mantenimiento se vuelve al análisis, el diseño o la codificación, y también desde cualquier fase se puede volver a la anterior si se detectan fallos. Estas vueltas atrás no son controladas, ni quedan explícitas en el modelo, y este es uno de los problemas que presenta este paradigma.

Etapa 8: Sustitución

La vida del software no es ilimitada y cualquier aplicación, por buena que sea, acaba por ser sustituida por otra más amplia, más rápida o más bonita y fácil de usar. La sustitución de un software que está funcionando por otro que acaba de ser desarrollado es una tarea que hay que planificar cuidadosamente y que hay que llevar a cabo de forma organizada. Es conveniente realizarla por fases, si esto es posible, no sustituyendo todas las aplicaciones de golpe, puesto que la sustitución conlleva normalmente un aumento de trabajo para los usuarios, que tienen que acostumbrarse a las nuevas aplicaciones, y también para los implementadores, que tienen que corregir los errores que aparecen. Es necesario hacer una migración de la información que maneja el sistema viejo a la estructura y el formato requeridos por el nuevo. Además, es conveniente mantener los dos sistemas funcionando en paralelo durante algún tiempo para comprobar que el sistema nuevo funcione correctamente y para asegurar el funcionamiento normal de la organización aún en el caso de que el sistema nuevo falle y tenga que volverse a alguna de las fases de desarrollo. La

sustitución implica el desarrollo de programas para la interconexión de ambos sistemas, el viejo y el nuevo, y para la migración y eventualmente la replicación de los datos entre ambos, evitando la duplicación del trabajo de las personas encargadas del proceso de datos durante el tiempo en que ambos sistemas funcionen en paralelo.

El ciclo de vida en cascada es el paradigma más antiguo, más conocido y más ampliamente usado en la ingeniería de software. No obstante, ha sufrido diversas críticas, debido a los problemas que se plantean al intentar aplicarlo a determinadas situaciones. Entre estos problemas están:

- En la realidad los proyectos no siguen un ciclo de vida estrictamente secuencial como propone el modelo. Siempre hay iteraciones. El ejemplo más típico es la fase de mantenimiento, que implica siempre volver a alguna de las fases anteriores, pero también es muy frecuente que en una fase, por ejemplo el diseño, se detecten errores que obliguen a volver a la fase anterior, el análisis.
- Es difícil que se puedan establecer inicialmente todos los requisitos del sistema. Normalmente los clientes no tienen conocimiento de la importancia de la fase de análisis o bien no han pensado en todo detalle qué es lo que quieren que haga el software. Los requisitos se van aclarando y refinando a lo largo de todo el proyecto, según se plantean dudas concretas en el diseño o la codificación pero el ciclo de vida clásico requiere la definición inicial de todos los requisitos y no es fácil acomodar en él las incertidumbres que suelen existir al comienzo de todos los proyectos.
- Hasta que se llega a la fase final del desarrollo: la codificación, no se dispone de una versión operativa de las aplicaciones. Como la mayor parte de los errores se detectan cuando el cliente puede probar los programas no se detectan hasta el final del proyecto, cuando son más costosos de corregir y más prisa (y más presiones) hay por que el sistema se ponga definitivamente en marcha.

Todos estos problemas son reales, pero de todas formas es mucho mejor desarrollar software siguiendo el modelo de ciclo de vida en cascada que hacerlo sin ningún tipo de guías. Además, este modelo describe una serie de pasos genéricos que son aplicables a cualquier otro paradigma, refiriéndose la mayor parte de las críticas que recibe a su carácter secuencial.

2.4.2 El modelo de desarrollo incremental o evolutivo

Dos de las críticas que se hacían al modelo de ciclo de vida en cascada eran que es difícil tener claros todos los requisitos del sistema al inicio del proyecto, y que no se dispone de una versión operativa de la o las aplicaciones hasta las fases finales del desarrollo, lo que dificulta la detección de errores y deja también para el final el descubrimiento de los requisitos inadvertidos en las fases de análisis. Para paliar estas deficiencias se ha propuesto un modelo de ciclo de vida evolutivo.

El desarrollo evolutivo se basa en la idea de desarrollar una implementación inicial exponiéndola a los comentarios del usuario y refinándola a través de las diferentes versiones hasta que se desarrolla un sistema adecuado. Las actividades de especificación, desarrollo y validación se

entrelazan en vez de separarse con una rápida retroalimentación entre éstas. Existen dos tipos de desarrollo evolutivo:

1. **Desarrollo exploratorio:** donde el objetivo del proceso es trabajar con el cliente para explorar sus requerimientos y entregar un sistema final. El desarrollo empieza con las partes del sistema que se comprenden mejor. El sistema evoluciona agregando nuevos atributos propuestos por el cliente.
2. **Prototipos desechables:** donde el objetivo del proceso de desarrollo evolutivo es comprender los requerimientos del cliente y entonces desarrollar una definición mejorada de los requerimientos del sistema. El prototipo se centra en experimentar con los requerimientos del cliente que no se comprenden del todo.

En la producción de sistemas, un enfoque evolutivo para el desarrollo de software suele ser más efectivo que el enfoque en cascada ya que satisface las necesidades inmediatas de los clientes. La ventaja de un proceso del software que se basa en un enfoque evolutivo es que la especificación se puede desarrollar en forma creciente. Tan pronto como los usuarios desarrollen un mejor entendimiento de su problema, éste se puede reflejar en el sistema software. El desarrollo de software incremental es la parte fundamental de los enfoques ágiles y es mucho mejor que el en cascada para la mayoría de los sistemas empresariales de comercio electrónico y personales. El desarrollo incremental refleja la forma en que se resuelven los problemas; rara vez se trabaja por adelantado una solución completa del problema sino que más bien se avanza en una serie de pasos hacia una solución y se retrocede cuando se detecta que se cometieron errores. Cada incremento o versión del sistema incorpora algunas de las funciones que necesita el cliente. Por lo general los primeros incrementos del sistema incluyen la función más importante o más urgente. De esta manera el cliente puede evaluar el desarrollo del sistema en una etapa relativamente temprana para constatar si se entrega lo que se requiere. En caso contrario, sólo el incremento actual se cambia y posiblemente se defina una nueva función para incrementos posteriores.

Para sistemas grandes, se recomienda un proceso mixto que incorpore las mejores características del modelo en cascada y del desarrollo evolutivo. Esto puede implicar desarrollar un prototipo desechable utilizando un enfoque evolutivo para resolver incertidumbres en la especifi-

cación del sistema. Puede entonces reimplementarse utilizando un enfoque más estructurado. Las partes del sistema bien comprendidas se pueden especificar y desarrollar utilizando un proceso basado en el modelo de cascada. Las otras partes, como la interfaz de usuario que son difíciles de especificar por adelantado, se deben desarrollar siempre utilizando un enfoque de desarrollo exploratorio.

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 un buen candidato para este tipo de modelo. No obstante, hay que tener en cuenta la complejidad: si la aplicación necesita que se desarrolle una gran cantidad de código para poder tener un prototipo para mostrar al usuario, las ventajas de la construcción de prototipos se verán superadas por el esfuerzo de desarrollar un prototipo que al final habrá que desechar o modificar mucho. También hay que tener en cuenta la predisposición del cliente para probar un prototipo y sugerir modificaciones de los requisitos. Puede ser que el cliente "no tenga tiempo para andar jugando" o "no vea las ventajas de este método de desarrollo".

También es conveniente construir prototipos para probar la eficiencia de los algoritmos que se van a implementar, o para comprobar el rendimiento de un determinado componente del sistema, por ejemplo, una base de datos o el soporte hardware, en condiciones similares a las que existirán durante la utilización del sistema. Es bastante frecuente que el producto de ingeniería desarrollado presente un buen rendimiento durante la fase de pruebas realizada por los ingenieros antes de entregarlo al cliente (pruebas que se realizarán normalmente con unos pocos registros en la base de datos o pocos clientes conectados al sistema), pero que sea muy ineficiente, o incluso inviable, a la hora de almacenar o procesar el volumen real de información que producirá su trabajo en el ambiente de producción real. En estos casos, la construcción de un prototipo de parte del sistema y la realización de pruebas de rendimiento, sirven para decidir, antes de empezar la fase de diseño, cuál es el modelo más adecuado de entre la gama disponible para el soporte hardware o cómo deben hacerse los accesos a la base de datos para obtener buenas respuestas en tiempo cuando la aplicación esté ya en funcionamiento.

En otros casos, el prototipo servirá para modelar y poder mostrar al cliente cómo va a realizarse la entrada y salida de datos en la aplicación, de forma que éste pueda hacerse una idea de como va a ser el sistema final, pudiendo entonces detectar deficiencias o errores en la especificación aunque el modelo no sea más que una cáscara vacía.

El prototipado es una técnica que sirve fundamentalmente para la fase de análisis de requisitos, pero lleva consigo la obtención de una serie de subproductos que son útiles a lo largo del desarrollo del proyecto ya que parte del trabajo realizado durante la fase de diseño rápido (especialmente la definición de pantallas e informes) puede ser utilizada durante el diseño del producto final. Además, tras realizar varias vueltas en el ciclo de construcción de prototipos, el diseño del mismo se parece cada vez más al que tendrá el producto final. Además durante la fase de construcción de prototipos será necesario codificar algunos componentes del software que también podrán ser reutilizados en la codificación del producto final, aunque deban ser optimizados en cuanto a corrección o velocidad de procesamiento.

No obstante, hay que tener en cuenta que el prototipo **no es el sistema final**, puesto que normalmente apenas es utilizable. Será demasiado lento, demasiado grande, inadecuado para el

volumen de datos necesario, contendrá errores (debido al diseño rápido), será demasiado general (sin considerar casos particulares, que debe tener en cuenta el sistema final) o estará codificado en un lenguaje o para una máquina inadecuadas, o a partir de componentes software previamente existentes. No hay que preocuparse de haber desperdiciado tiempo o esfuerzos construyendo prototipos que luego habrán de ser desechados si con ello se ha conseguido tener más clara la especificación del proyecto, puesto que el tiempo perdido se ahorrará en las fases siguientes, que podrán realizarse con menos esfuerzo y en las que se cometerán menos errores que nos obliguen a volver atrás en el ciclo de vida.

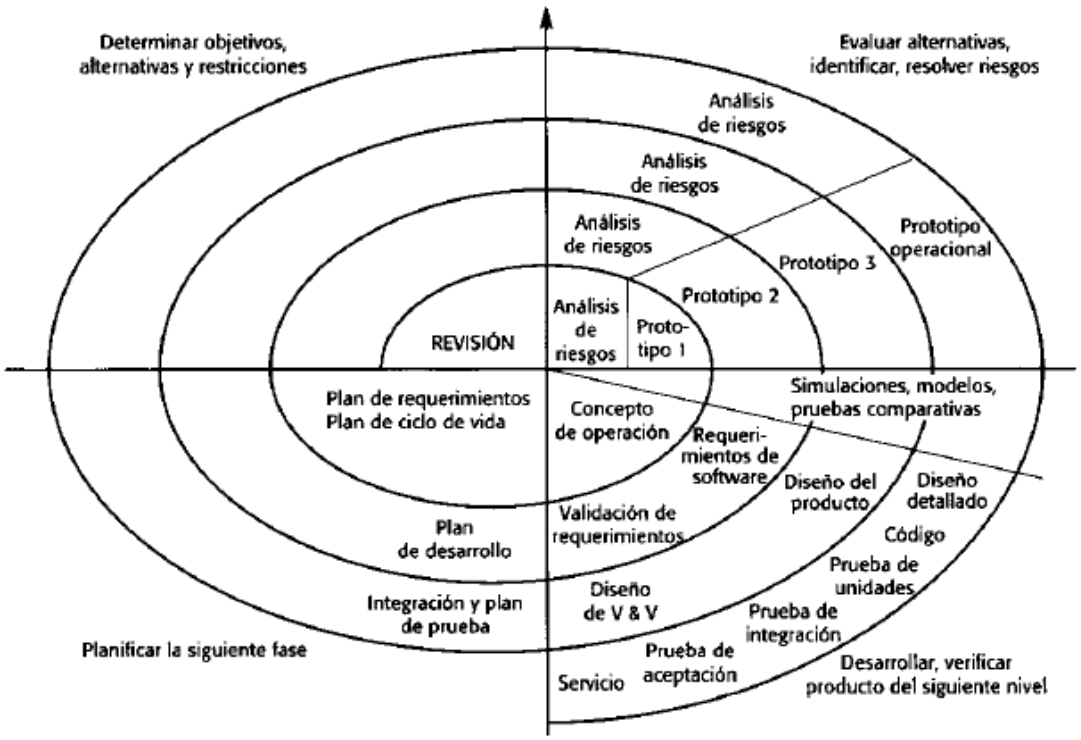
Hay que tener en cuenta que un análisis de requisitos incorrecto o incompleto, cuyos errores y deficiencias se detecten a la hora de las pruebas o tras entregar el software al cliente, obligará a repetir de nuevo las fases de análisis, diseño y codificación, que tal vez se habían realizado cuidadosamente, pensando que se estaba desarrollando el producto final. Al tener que repetir estas fases, sí que estaremos desechando una gran cantidad de trabajo, normalmente muy superior al esfuerzo de construir un prototipo basándose en un diseño rápido, en la reutilización de trozos de software preexistentes y en herramientas de generación de código para informes y manejo de ventanas.

Uno de los problemas que suelen aparecer siguiendo el paradigma de construcción de prototipos, es que con demasiada frecuencia el prototipo pasa a ser parte del sistema final, bien sea por presiones del cliente, que quiere tener el sistema funcionando lo antes posible o bien porque los técnicos se han acostumbrado a la máquina, el sistema operativo o el lenguaje con el que se desarrolló el prototipo. 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, o que las elecciones de lenguaje, sistema operativo o máquina para desarrollarlo se han hecho basándose en criterios como el mejor conocimiento de esas herramientas por parte los técnicos que en que sean adecuadas para el producto final.

2.4.3 El modelo en espiral de Boehm

El modelo en espiral combina las principales ventajas del modelo de ciclo de vida en cascada y del modelo de construcción de prototipos. Proporciona un modelo evolutivo para el desarrollo de sistemas de software complejos, mucho más realista que el ciclo de vida clásico, y permite la utilización de prototipos en cualquier etapa de la evolución del proyecto. Es un modelo de proceso dirigido por el riesgo. Aquí el proceso de software se presenta como una espiral y no como una secuencia de actividades con cierto retroceso de una actividad a otra. Cada ciclo en la espiral representa una fase del proceso de software, de esta manera, el ciclo más interno puede relacionarse con la factibilidad del sistema, el siguiente con la definición de requerimientos, el siguiente con el diseño del sistema, etc.

La reducción de riesgos se realiza mediante los prototipos, permitiendo finalizar el proyecto antes de haberse embarcado en el desarrollo del producto final (si el riesgo es demasiado grande).



El modelo en espiral define cuatro tipos de actividades, y representa cada uno de ellos en un cuadrante:

Establecimiento de objetivos y planificación

Consiste en determinar los objetivos del proyecto, las posibles alternativas y las restricciones. Esta fase equivale a la de recolección de requisitos del ciclo de vida clásico e incluye además la planificación de las actividades a realizar en cada iteración.

Análisis de riesgo

En cada uno de los riesgos identificados del proyecto, se realiza un análisis minucioso. Se dan acciones para reducir el riesgo. Por ejemplo, si existe un riesgo que los requerimientos sean inadecuados, puede desarrollarse un sistema prototipo.

Desarrollo y validación

Después de una evaluación del riesgo, se elige un modelo de desarrollo para el sistema. Por ejemplo, la creación de prototipos desechables sería el mejor enfoque de desarrollo si predominan los riesgos en la interfaz del usuario. Si el principal riesgo identificado es la integración de subsistemas, el modelo en cascada sería el mejor modelo de desarrollo a utilizar.

Planeación

El proyecto se revisa y se toma una decisión sobre si hay que continuar con otro ciclo de la espiral. Si se opta por continuar, se trazan los planes para la siguiente fase del proyecto.

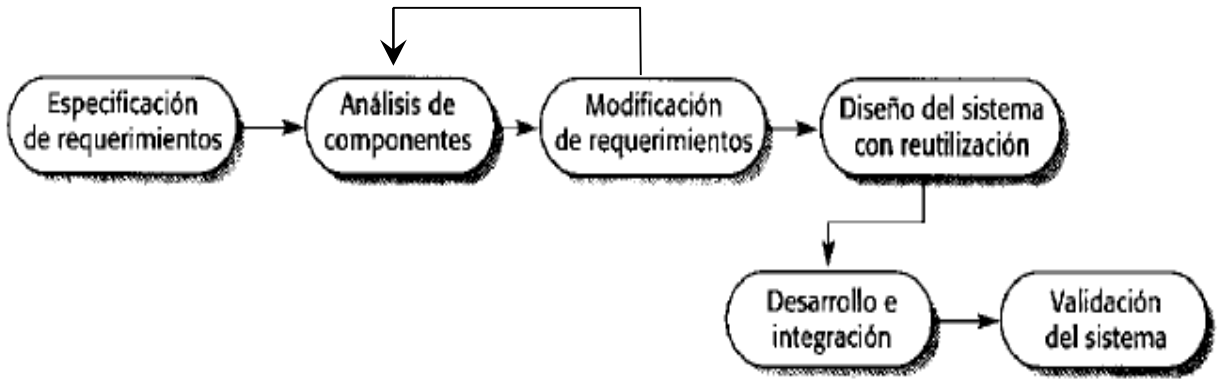
En la primera iteración se definen los requisitos del sistema y se realiza la planificación inicial del mismo. A continuación se analizan los riesgos del proyecto, basándonos en los requisitos iniciales y se procede a construir un prototipo del sistema. Entonces el cliente procede a evaluar el prototipo y con sus comentarios, se procede a refinar los requisitos y a reajustar la planificación inicial, volviendo a empezar el ciclo. En cada una de las iteraciones se realiza el análisis de riesgos, teniendo en cuenta los requisitos y la reacción del cliente ante el último prototipo. Si los riesgos son demasiado grandes se terminará el proyecto, aunque lo normal es que se siga avanzando a lo largo de la espiral.

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

La diferencia principal con el modelo de construcción de prototipos, es que en éste los prototipos se usan para perfilar y definir los requisitos. Al final, el prototipo se desecha y comienza el desarrollo del software siguiendo el ciclo clásico. En el modelo en espiral, en cambio, los prototipos son sucesivas versiones del producto, cada vez más detalladas (el último es el producto en sí) y constituyen el esqueleto del producto de ingeniería razón por la cual deben construirse siguiendo estándares de calidad.

2.4.4 Ingeniería de software orientada a la reutilización

En la mayoría de proyectos de software hay cierta reutilización de software. Sucede con frecuencia de manera informal cuando las personas que trabajan en el proyecto conocen diseños o códigos similares a los requeridos. Los buscan, modifican e incorporan en sus sistemas. Esta utilización ocurre independientemente del proceso de desarrollo que se emplee. En ocasiones tales componentes son sistemas por derecho propio (sistemas comerciales, ERP, procesadores de texto, hojas de cálculo) que pueden mejorar la funcionalidad específica.



Las etapas de especificación de requerimientos y la de validación son similares a otros procesos de software, las etapas intermedias son diferentes. Estas etapas son:

Análisis de componentes

Dada la especificación de requerimientos, se realiza una búsqueda de componentes para implementar dicha especificación. Por lo general, no hay coincidencia exacta y los componentes que se usa proporcionan sólo parte de la funcionalidad requerida.

Modificación de requerimientos

En esta etapa se analizan los requerimientos usando información de los componentes descubiertos. Luego se modifican para reflejar los componentes disponibles. Donde las modificaciones son imposibles, puede regresarse a la actividad de análisis de componentes para buscar soluciones alternativas.

Diseño de sistema con reutilización

En esta etapa se diseña el marco conceptual del sistema o se reutiliza el existente. Los creadores toman en cuenta los componentes que se reutilizan y organizan el marco de referencia para atenderlo. Es posible que deba diseñarse algo de software nuevo si no están disponibles los componentes reutilizables.

Desarrollo e integración

Se diseña el software que no puede procurarse de manera externa y se integran los componentes y sistemas comerciales para crear el nuevo sistema.

Existen tres tipos de componentes de software que pueden usarse en un proceso orientado a la reutilización:

1. Servicios Web que se desarrollan en concordancia para atender servicios estándares y que están disponibles para la invocación remota.
2. Colecciones de objetos que se desarrollan como un paquete para su integración con el marco de componentes como J2EE o .NET.
3. Sistemas de software independientes que se configuran para usar en un entorno particular.

La ingeniería de software orientada a la reutilización tiene la clara ventaja de reducir la cantidad de software a desarrollar y consecuentemente disminuir costos y riesgos. Por lo general conduce a entregas más rápidas de software, pero puede conducir hacia un sistema que no cubra las necesidades reales de los usuarios. Además se pierde el control sobre la evolución del sistema conforme las nuevas versiones de los componentes reutilizables no estén bajo el control de la organización que lo usa.

Universidad Nacional del Litoral

FACULTAD DE INGENIERÍA
Y CIENCIAS HÍDRICAS

Ingeniería en Informática

Ingeniería de Software I

TEMA III – Actividades del proceso de desarrollo

Introducción

Las cuatro actividades básicas del proceso: *especificación, desarrollo, validación y evolución* se organizan de forma distinta en diferentes procesos de desarrollo. En el enfoque en cascada están organizadas en secuencia, mientras que en el desarrollo evolutivo se entrelazan. Cómo se llevan a cabo estas actividades depende del tipo de software, de las personas y de la estructura organizacional. No hay una forma correcta o incorrecta de organizar estas actividades, y el objetivo de este tema es simplemente proporcionar una introducción de cómo se pueden organizar.

3.1 INICIO DEL PROYECTO

3.1.1 Razones de solicitudes de proyectos

Para desencadenar el proceso del desarrollo de un sistema de información de software, debe existir una petición o requerimiento ya sea de un gerente, un empleado, un área de sistemas, una repartición, autoridades políticas, etc.

Las razones por las que se hacen los requerimientos pueden apuntar a diferentes direcciones. A veces tienden a solucionar un problema como la reducción de costos, realizar ciertas tareas o mejorar el control del trabajo que se lleva a cabo. Otras veces es para mejorar la eficiencia del trabajo realizado en distintas áreas de la organización. A continuación se indicarán algunos de los motivos que se presentan con mayor frecuencia:

Mayor velocidad de proceso

Utilizar la capacidad de la computadora para calcular, clasificar y consultar datos e información cuando se desea una mayor velocidad de proceso que la del personal que efectúa las mismas tareas. Esto permite además, liberar al personal de trabajos tediosos de cálculos o comparaciones rutinarias.

Mayor exactitud y mejor consistencia

En ocasiones se solicitan los proyectos de sistemas de información para mejorar la exactitud de los datos procesados o para asegurar que siempre se siga un mismo procedimiento que describe cómo realizar una tarea específica, evitando por ejemplo, que el personal tenga que elegir datos de distintos lugares para aplicarlos a una actividad rutinaria como ser el cálculo de actualizaciones monetarias, uso de coeficientes de gran longitud, operaciones que involucran valores de tablas preestablecidas, etc.

Consulta más rápida a la información

Las organizaciones, almacenan generalmente, grandes cantidades de datos sobre sus operaciones, empleados, clientes, proveedores, contabilidad, finanzas, etc. Dos aspectos son constantes: dónde almacenar los datos, y cómo consultarlos cuando se necesitan. El almacenamiento de datos, se vuelve más complejo si los usuarios desean recuperar los datos en varias formas distintas, y en diferentes circunstancias, efectuando rastreos que demandan tiempo y personal. Las formas de almacenamiento y posterior recuperación que se pueden lograr con el uso de un sistema informático,

permiten por un lado poder almacenar toda la información en dispositivos secundarios y por el otro, hacer recuperaciones dinámicas en poco tiempo y con exactitud.

Integración de las áreas de la organización

Los sistemas de información, se utilizan para integrar las actividades que se expanden alrededor de diversas áreas de la organización. En muchas de ellas, el trabajo hecho en determinada área, es coordinado con el que se lleva a cabo en otra. Información que es capturada en un área, tal vez otra también la necesite y no debe cargarla o procesarla nuevamente. Tal vez un organismo se valga de esa información cargada en varios lugares distintos y la procese en un nivel de consolidación superior. Es importante destacar que debe existir un único punto de entrada de información. Una vez ingresada al sistema, estará disponible para cualquier área.

Reducción de costos

Algunos diseños de sistemas permiten que se realice la misma cantidad de trabajo a menor costo, es decir, si se acepta la ventaja del cálculo automatizado y de las capacidades de recuperación que se pueden incluir en procedimientos de flujo continuo en programas de computadoras. Algunas de las tareas las llevará a cabo un programa de computadora, pero habrá otras que continuarán realizándose por el hombre por el alto componente manual que requiere o bien porque es él quien debe tomar algún tipo de decisión no formalizable que permita la prosecución del circuito de información.

Mayor seguridad

A veces, el hecho de que los datos puedan almacenarse en forma legible para la máquina, provee seguridad que sería difícil de alcanzar en un ambiente no computarizado. A una computadora le es indiferente en el tratamiento automático, trabajar con codificaciones que con descripciones de esos códigos. También la manera de acceder a la información, depende de ciertos permisos existiendo obstáculos que le impiden ingresar a cualquier persona ajena a la organización o sin autorización. Esto no significa que no se pueda penetrar cualquier tipo de bloqueo impuesto, pero disminuye el riesgo consecuentemente con la cantidad de personas que tienen los conocimientos como para hacerlo.

3.1.2 Origen de las solicitudes de proyectos

Existen cuatro orígenes principales de solicitudes de proyectos divididos en dos grupos. El primero de ellos, está compuesto por elementos interiores de la organización, siendo los potenciales solicitantes:

- Directores, Gerentes, Jefes de alto rango
- Altos ejecutivos, Autoridades con cargos políticos (para las reparticiones públicas)
- Analistas de Sistemas, Sectoriales de Informática, Direcciones de Informática

Desde las influencias exteriores, éstas se ven manifestadas a través de exigencias por parte del estado o agencias gubernamentales las que pueden solicitar proyectos de sistemas de información. Los solicitantes pueden pretender aplicaciones totalmente nuevas o proponer cambios a las ya existentes dependiendo del origen y de la razón de la solicitud. Puede ocurrir que debido a los fuertes cambios en las reglas de gestión o la implantación de metodologías y técnicas nuevas o distintas a las mantenidas hasta el momento.

3.1.3 Administración de la revisión y selección de proyectos

Normalmente, se generan muchas más solicitudes de desarrollo de sistemas, de las que las organizaciones quieren o son capaces de llevar a cabo. Algunas solicitudes valen la pena, otras no. Antes de que cualquier trabajo se lleve a cabo con relación a estas solicitudes, alguien debe decidir cuáles se van a rechazar (quizás para canalizarlas por otros medios). La decisión de aceptar o rechazar una petición, puede hacerse en muchas formas diferentes y por diversos miembros de la organización. Los analistas de sistemas no son los que toman la decisión final. La metodología más difundida para revisar y elegir los proyectos de desarrollo viables, es mediante la conformación de un grupo multidisciplinario de selección. Este grupo multidisciplinario estará compuesto por:

- **Grupo directivo:** personal jerárquico de la organización (eventualmente autoridades políticas),
- **Grupo de usuarios:** personal especialista en el tema tocado por la solicitud de sistemas y que se verían afectados directamente por la solución adoptada,
- **Grupo de sistemas de información:** personal especialista en sistemas de información

No necesariamente para tomar una decisión deberán intervenir los tres grupos. Dependiendo del caso, uno de ellos bastaría. En el caso en que participen todos ellos, la adopción de una postura puede no ser compartida por los tres grupos, no obstante suelen existir fuertes condicionantes externos (caso de presiones políticas que constituiría el cuarto grupo fantasma), que obligan a aceptar una situación sin siquiera evaluarla.

3.1.4 La solicitud del proyecto

La propuesta del proyecto, sometida por el usuario o analistas del grupo de selección, es un elemento crítico para iniciar el estudio de sistemas. Aunque la forma de tal solicitud varía de una organización a otra, existe un acuerdo general sobre la clase de información que debe proporcionarse. Esta información está referida a:

- ¿Cuál es el problema?
- Detalles del problema
- ¿Qué tan significativo es el problema?
- ¿Cuál cree el usuario que es la solución?
- ¿Cómo ayudaría la realización de modificaciones o desarrollo del nuevo sistema de información?

- ¿A quién más que conozca acerca de este problema puede contactarse?

En la propuesta, quien hace la petición, identifica la situación, indica dónde se necesita ayuda y deberá proporcionar los detalles. También es útil para los miembros de grupo de selección, conocer por qué quien hace la petición piensa que el proyecto es importante. La presentación de un informe que describa el significado del problema o situación, servirá como punto de partida para la discusión. Rara vez, existe una solución clara y sencilla al problema o situación. Dado que se deberá efectuar una investigación preliminar para conocer más, es útil si la persona que hace la petición, proporciona los datos de a qué entidad y con qué individuos se puede hacer contacto para obtener más información.

Se aclara que se tratarán los sistemas de información que involucran el desarrollo de componentes de software.

3.2 ACTIVIDADES DEL PROCESO DE DESARROLLO

Como se viera en el tema anterior, existen diferentes procesos de software pero todos incluyen cuatro actividades que son fundamentales para la ingeniería de software:

- **Especificación del software**. Tienen que definirse tanto la funcionalidad del software como las restricciones de su operación.
- **Diseño e implementación**. Debe desarrollarse el software para cumplir con las especificaciones.
- **Validación del software**. Hay que validar el software para asegurarse de que cumple con lo que el cliente quiere.
- **Evolución del software**. El software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente.

Cuando la evolución del entorno es muy fuerte - técnicas obsoletas, software obsoleto, modificaciones profundas a las reglas de gestión, etc. -, puede dar origen a otro ciclo de vida.

Estas etapas están íntimamente ligadas y son inseparables. Si bien, las etapas, se exponen en una secuencia, no debe interpretarse que una etapa comienza exactamente con la terminación de la anterior, por lo contrario, algunas fases se superponen. Por otra parte, se puede presentar un comportamiento iterativo, ya que en muchos casos, es necesario volver a una etapa anterior, a raíz de una información incorrecta, una interpretación incorrecta por parte de quien lleva adelante el análisis, etc.

3.2.1 Requerimientos - especificación del software

Considérese por ejemplo el depósito de un comercio que vende determinadas líneas de productos. Con el objeto de controlar mejor sus stocks de mercaderías y tener información más actualizada sobre esos niveles y los puntos de pedido, el comercio necesita agilizar la operación del depósito. Antes de diseñar un sistema para la captación de datos, actualización de archivos y producción de informes (esto último es lo que más le interesa al encargado del depósito), se debe conocer más acerca de cómo maneja la

empresa sus operaciones como ser:

- Saber qué formularios o documentos se utilizan para respaldar las gestiones que se realizan tales como: órdenes de pedido, órdenes de compra, órdenes de pago, facturaciones, etc.

- Saber qué informes - si existen algunos- se producen ahora y para qué se utilizan; por lo tanto se debe buscar la información acerca de dichos informes: listas de avisos de pedidos, solicitudes de compra, listado de mercaderías en stock, etc.

- Es necesario precisar en dónde se origina esta información; por ejemplo: el departamento de compras, el depósito o los departamentos contables.

En otras palabras, se debe comprender la forma en que trabaja el sistema actual, y más específicamente, cuál es el flujo de información que atraviesa al sistema.

Resulta importante también comprender por qué motivo el comercio desea cambiar sus operaciones actuales:

- ¿Tiene problemas al dar seguimiento a sus pedidos, a la mercadería o al dinero?
- ¿Utiliza demasiados formularios en papel innecesariamente?
- ¿Necesita un sistema más eficiente antes de que amplíe sus operaciones?
- ¿Necesita brindar una imagen distinta a sus clientes?

Sólo después de conocer todos estos datos se puede comenzar a definir cómo y dónde se puede beneficiar un sistema de información basado en software y que sirva a todos los usuarios del sistema. Además se deben relevar las expectativas nuevas y funcionalidades que se desea que tenga el nuevo sistema. Esta acumulación de información forma parte de lo denominado **Requerimientos** y obviamente debe preceder a todas las etapas de cualquier proceso de desarrollo de software.

La especificación del software o la ingeniería de requerimiento, consiste en el proceso de comprender y definir qué servicios se requieren del sistema así como la identificación de restricciones sobre la operación y el desarrollo. Es una etapa particularmente crítica del proceso de software ya que los errores aquí, conducen de manera inevitable a problemas posteriores tanto en el diseño como en la implementación del sistema y pueden concluir con el fracaso del proyecto.

El proceso de ingeniería de requerimientos se enfoca en producir un documento de requerimientos convenido que especifique los requerimientos de los interesados que cumplirá el sistema. Por lo general los requerimientos se presentan en dos niveles de detalle. Los usuarios finales y clientes necesitan un informe de requerimientos de alto nivel, mientras que los desarrolladores de sistemas precisan una descripción más detallada del sistema.

En esta etapa el trabajo conjunto entre usuarios y equipo de desarrollo, es fundamental. El analista debe considerar cuáles son y serán las necesidades de la organización. Al trabajar con gerentes y empleados, el analista además de conocer el

dominio, recomienda qué medidas podrían adoptarse para una solución o una mejora al utilizar un sistema de software. Obviamente estas recomendaciones tienen límites impuestos por el sistema en sí mismo, ya que en caso de tratarse de disciplinas científicas o técnicas, primeramente es necesario que quienes utilicen el sistema, tengan claramente establecidos sus objetivos, ideas y metodologías de trabajo. De todas manera, las medidas a tomar deben basarse en aspectos como

- la adaptabilidad de la solución a la estructura de la organización actual
- el apoyo que deberá tener por parte de los usuarios finales

Si los usuarios que emplearán el sistema no se sienten a gusto con éste, fallará en su propósito por mejorar la situación.

El grado de participación de gerentes y empleados dentro de una organización con el sistema de información, puede variar dependiendo del tipo de usuario. Los tipos de usuarios serán clasificados como:

- **Usuarios directos:** son quienes realmente interactúan con el sistema. Ellos ingresan datos o reciben salidas (consultas).

- **Usuarios indirectos:** son quienes se benefician de los resultados o informes producidos por el sistema, pero no interactúan directamente con el hardware o software.

- **Usuarios administrativos:** son los que administran los sistemas de aplicación. Son quienes tienen la autoridad para aprobar o desaprobar la inversión en el desarrollo de aplicaciones. Tienen además la responsabilidad de la organización para la efectividad de los sistemas. Estos usuarios participan activamente en el desarrollo del sistema.

Con frecuencia, y como una primera aproximación al conocimiento de una organización, son utilizados los organigramas para describir las relaciones entre sus componentes, tales como divisiones, departamentos, secciones, oficinas, etc. Aunque los organigramas puedan mostrar las relaciones formales entre los componentes con cierta exactitud, no dicen cómo opera el sistema de la organización, dado que muchos detalles de importancia no se pueden (además no corresponde) describir en las cajas del diagrama. Asimismo, una estructura no siempre refleja las dependencias funcionales reales existentes entre sus distintos miembros.

Existen cinco actividades principales en el proceso de ingeniería de requerimientos:

3.2.1.1 Actividades de la ingeniería de requerimientos

3.2.1.1.1 Investigación preliminar

El propósito de la investigación preliminar, radica en evaluar las peticiones de proyecto. No es un estudio de diseño ni tampoco incluye la recopilación de datos para describir completamente el sistema de la organización. En lugar de esto, los analistas

recopilan la información que permite a los miembros del grupo de selección, evaluar las ventajas de la petición del proyecto y dar su juicio bien fundamentado sobre la factibilidad del proyecto propuesto. Los analistas deben en esta etapa:

- Aclarar y entender la petición del proyecto
- Determinar el tamaño del proyecto
- Señalar los costos y beneficios de las alternativas apropiadas
- Determinar la factibilidad técnica y operativa de los enfoques alternativos
- Informar los hallazgos a la gerencia con recomendaciones y subrayando la aceptación o rechazo de la propuesta

Los datos que el analista recaba durante las investigaciones preliminares, se recopilan por medio de dos métodos principales:

- **la revisión de documentos**
- **la entrevista al personal escogido de la organización**.

Los analistas que llevan a cabo la investigación, primero inspeccionan la sección de la organización afectada por el proyecto. Normalmente, los analistas pueden reconocer estos datos, si examinan los organigramas de la organización y estudian los procedimientos de operación descritos. Los procedimientos describen cómo opera el proceso.

Los documentos escritos, señalan a los analistas cómo deben operar los sistemas, pero pueden no incluir suficientes datos para decidir sobre las ventajas de una propuesta de sistemas, ni presentar aspectos sobre las operaciones corrientes. Para conocer estos datos, los analistas realizan entrevistas. En las entrevistas, los analistas estudian las características de los sistemas con el objeto de conocer más hechos sobre la naturaleza del proyecto requerido y la razón para analizarlo; por lo tanto, deben estar seguros de remarcar el requerimiento y el problema que los guíen en sus entrevistas; en otras palabras, deben recabar datos que expliquen más adelante la naturaleza del proyecto solicitado y muestren si la asistencia se justifica en términos económicos, operativos y técnicos. Éste no es el momento para trabajar sobre aspectos posteriores de la investigación detallada. Normalmente, las entrevistas de la investigación preliminar, sólo incluye a la gerencia y al personal de supervisión.

3.2.1.1.2 Estudio de la factibilidad del proyecto

Se realiza una estimación sobre si las necesidades identificadas del usuario se cubren con las actuales tecnologías de software y hardware. El estudio considera si el sistema propuesto tendrá un costo/beneficio desde un punto de vista empresarial y si éste puede desarrollarse dentro de las restricciones presupuestarias existentes. Un estudio de factibilidad debe ser rápido y relativamente barato. El resultado debe informar la decisión respecto a si se continúa o no con un análisis más detallado.

La recopilación de elementos que se lleva a cabo durante la investigación preliminar, examina la factibilidad del proyecto; es decir, la posibilidad de que el sistema sea beneficioso a la organización. Se estudian tres pruebas de factibilidad: operativa,

técnica y financiera. Todas son de la misma importancia.

Factibilidad Operativa

Los proyectos propuestos son beneficiosos sólo si pueden convertirse en sistemas de información que cumplan los requerimientos operativos de la organización y sea consecuente con los objetivos de ella. Dicho de otra forma, esta prueba de factibilidad, cuestiona si el sistema trabajará cuando se desarrolle e instale. Tiene que ver fundamentalmente con el hecho que existan obstáculos para ponerlo en marcha; por ejemplo si tiene suficiente apoyo por parte de la gerencia; si los usuarios aceptarán un cambio que traiga un sistema operativo distinto aunque sea más útil, etc.

Los aspectos que son relativamente pequeños y parecen cuestiones de menor importancia al principio, encuentran siempre maneras de crecer y convertirse en problemas mayores después de la puesta en marcha; por lo tanto, todos los aspectos operativos deben considerarse con cuidado.

Factibilidad Técnica

Los aspectos técnicos, que normalmente surgen durante la etapa de la factibilidad de investigación son:

- ¿Existe la tecnología necesaria (o puede adquirirse) para hacer lo que se sugiere?
- ¿Tiene el equipo propuesto la capacidad técnica para almacenar los datos requeridos y utilizarlos en el nuevo sistema?
- ¿El sistema propuesto y sus componentes proporcionarán las respuestas adecuadas a las preguntas, sin importar el número y ubicación de los usuarios?
- ¿Se puede agrandar el sistema si se desarrolla?
- ¿Existen garantías técnicas de exactitud, confiabilidad, facilidad de acceso y seguridad de datos?

Por ejemplo, si la propuesta incluye el uso de una impresora que imprime a una velocidad de 1000 páginas por minuto, una breve investigación muestra que esto es técnicamente factible. Si debe o no incluirse en la configuración debido a su costo, es una decisión de tipo económico.

Factibilidad financiera y económica

Un sistema que puede desarrollarse técnicamente y que se utilizará si se instala, debe considerarse como una buena inversión para la organización, es decir, los beneficios financieros, deberán igualar o exceder los costos financieros. Las preguntas económicas y financieras que se plantean los analistas durante la investigación preliminar, buscan estimaciones de:

- El costo de llevar a cabo una investigación completa de sistemas
- El costo de hardware y el software para el tipo de aplicación considerado
- Los beneficios en forma de reducción de costos, o menos errores costosos
- El costo si nada cambia (si el sistema no se desarrolla)

Para ser considerada factible, una propuesta de proyecto debe pasar todas estas pruebas; de otra forma, no es un proyecto factible. Muchas solicitudes de proyectos, mueren en esta etapa.

3.2.1.1.3 Obtención y análisis de requerimientos

Este es el proceso de derivar los requerimientos del sistema mediante la observación de los sistemas existentes, discusiones con los usuarios y proveedores potenciales, análisis de tareas, etc. Esto puede incluir el desarrollo de uno o más modelos de sistemas y prototipos, lo que ayuda a entender el sistema que se va a especificar.

El punto clave del análisis de sistemas, se consigue al adquirir un conocimiento detallado de todas las facetas importantes dentro del área de la organización que se investiga. Las preguntas claves que el analista debe conocer son:

- ¿Qué se está haciendo?
- ¿Cómo se está haciendo?
- ¿Qué tan frecuentemente ocurre?
- ¿Qué tan grande es la cantidad de transacciones o decisiones?
- ¿Qué tan bien se lleva a cabo la tarea?
- ¿Existe algún problema?
- Si el problema existe, ¿qué tan serio es?
- Si el problema existe, ¿cuál es la causa principal?

Para contestar a estas preguntas, los analistas deberán tomar contacto con diferentes personas para recabar los detalles con relación al proceso, así como sus opiniones sobre las causas por las cuales suceden las cosas de esa manera y que expongan algunas ideas para modificarlas.

Existen varias técnicas para la recopilación de información, cada una de ellas con ventajas y desventajas sobre las otras, por lo que solamente la experiencia del analista es la que le permitirá elegir la más adecuada o la combinación de algunas de ellas, lógicamente, dependiendo del caso de estudio y de la magnitud de la organización objeto del trabajo.

Entrevistas

Las entrevistas, se utilizan para recopilar información en forma verbal, a través del mantenimiento del diálogo entre el analista y los interlocutores acerca de temas previamente determinados por aquel. Los interlocutores pueden ser de niveles gerenciales o simplemente empleados, los cuales son usuarios actuales del sistema existente, usuarios potenciales del sistema propuesto o aquellos que proporcionarán datos o serán afectados por la aplicación propuesta. Es de destacar que la entrevista es una forma de conversación y NO de interrogación.

En las investigaciones de sistemas, las formas cualitativas y cuantitativas de información son importantes. La información cualitativa, está relacionada con opiniones, políticas y descripciones narrativas de actividades o problemas, mientras que las cuantitativas tratan con números, frecuencias o

cantidades. A menudo las entrevistas son la mejor fuente de información cualitativa; los otros métodos, tienden a ser útiles en la recopilación de datos cuantitativos. Son valiosas las opiniones, comentarios, ideas o sugerencias con relación a cómo se podría hacer el trabajo. Mucha gente incapaz de expresarse por escrito, puede discutir sus ideas en forma verbal. Como resultado de esto, las entrevistas pueden descubrir rápidamente malos entendidos, falsas expectativas o incluso, resistencia potencial para las aplicaciones de desarrollo.

Existen dos tipos de entrevistas:

- **Entrevistas Estructuradas**
- **Entrevistas sin Estructura**

Si el objetivo de la entrevista radica en adquirir información general, es conveniente elaborar una serie de preguntas (o mejor dicho, una forma de llevar la conversación hacia la parte que al analista le interese) sin estructura, con una sesión de preguntas y respuestas libres. La atmósfera abierta y de fácil flujo de esta modalidad, proporciona una mayor oportunidad para conocer las actitudes, ideas y creencias de quien responde.

Las entrevistas estructuradas, utilizan preguntas estandarizadas. El formato de respuestas para las preguntas puede ser abierto o cerrado.

Las preguntas para respuestas abiertas les permiten a los entrevistados dar cualquier respuesta que parezca apropiada. Pueden contestar por completo con sus propias palabras y exponiendo sus ideas particulares sobre el tema tratado.

Con las preguntas para respuestas cerradas, se proporciona al usuario un conjunto de respuestas alternativas que se pueden seleccionar. Todas las personas que responden se basan en un mismo conjunto de respuestas posibles.

Las entrevistas no estructuradas, requieren menos tiempo de preparación, porque no se necesita tener por anticipado las palabras precisas de las preguntas y de las posibles respuestas que se le darán como opciones al interlocutor. Sin embargo, analizar las respuestas después de las entrevistas, lleva más tiempo que con las estructuradas.

Dado que un limitado grupo de personas se seleccionarán para las entrevistas, se deberá ser cuidadoso en incluir aquellas personas que tienen información que no se podrá conseguir de otra forma. Durante las primeras etapas de un estudio de sistemas, cuando los analistas determinan la factibilidad del proyecto, con frecuencia las entrevistas sólo se aplican a la gerencia o personal de supervisión. Sin embargo, durante la investigación detallada en donde el objetivo es descubrir hechos específicos, opiniones y conocer cómo se manejan las operaciones desempeñadas actualmente, las entrevistas se aplican a todos los niveles gerenciales y a los empleados, dependiendo de quien pueda proporcionar la mayor parte de la información útil para el

estudio. Unos pueden conocer cuestiones generales de los problemas tratados (niveles gerenciales), mientras que otros los conocen con profundidad (empleados).

Demás está decir que la habilidad del analista es vital para el éxito en la búsqueda de hechos por medio de la entrevista ("no es fácil el tema de las relaciones humanas"). Las buenas entrevistas, dependen del conocimiento del analista tanto de la preparación del objetivo de una entrevista específica, como de las preguntas que se le van a realizar a una persona determinada.

Cuestionarios

Para los analistas, los cuestionarios, pueden ser la única manera posible de relacionarse con un gran número de personas para conocer varios aspectos del sistema. Por supuesto, no es posible observar las expresiones o reacciones de quienes responden a los cuestionarios. En la mayor parte de los casos, el analista, no verá a los que responden, y esto es una ventaja para quien contesta pues al realizarse muchas entrevistas, ayuda a asegurar que el interpelado cuenta con mayor anonimato y pueden darse respuestas más honestas (y menos respuestas predefinidas o estereotipadas). Las preguntas estandarizadas, pueden proporcionar datos más confiables. Asimismo, las características anteriores, también son desventajas de los cuestionarios. Aunque su aplicación puede realizarse con un mayor número de individuos, es muy rara una respuesta total. Puede necesitarse algún seguimiento de los cuestionarios para motivar al personal a que responda.

Existen dos formas de cuestionarios para recopilar datos: cuestionarios abiertos y cuestionarios cerrados, y su aplicación depende del grado de conocimiento que los analistas posean acerca de todas las posibles respuestas a las eventuales preguntas y de esa manera poder incluirlas. Con frecuencia, se utilizan ambas formas en los estudios de sistemas.

Los cuestionarios abiertos, al igual que las entrevistas, son aplicados cuando se quieren conocer los sentimientos, opiniones y experiencias generales. También son útiles al explorar el problema básico. El formato abierto, proporciona una amplia oportunidad para que quienes respondan, describan y expliciten las razones de sus ideas.

Los cuestionarios cerrados, limitan las respuestas posibles del interrogado. Por medio de un cuidadoso estilo en la pregunta, el analista puede controlar el marco de referencia. Este formato es el mejor método para obtener información sobre los hechos. También fuerza a los individuos para que tomen una posición y forma su opinión sobre los aspectos importantes. Las formas más comunes de respuestas a cuestionarios cerrados son:

1 - Si/No
2 - De acuerdo/en desacuerdo
3 - Selección de respuestas por escala
4 - Selección de puntajes o número de ocurrencias
5 - Rangos
6 - Selección limitada de respuestas

Los cuestionarios bien hechos, no se desarrollan rápidamente. Su confección lleva tiempo y mucho trabajo de preparación. La primera consideración se encuentra en determinar el objetivo del mismo. ¿Qué datos quiere conocer el analista a través de su uso?

Aquellas personas que reciban el cuestionario, deben seleccionarse de acuerdo con la información que puedan proporcionar. Escribir o imprimir un cuestionario, no significa que se pueda distribuir amplia y libremente sin un análisis previo. En caso que sean contestados por personas no calificadas y si el cuestionario es anónimo, ya no será posible retirar sus respuestas de la muestra, distorsionándose el resultado de la investigación en ese aspecto.

Revisión de registros

Con frecuencia, en muchas organizaciones la información ya se encuentra disponible para que el analista conozca las actividades y operaciones con las cuales no está familiarizado. Muchos tipos de registros e informes son accesibles si el analista sabe donde buscar. En la revisión de registros, los analistas examinan datos y descripciones que ya están escritos o registrados en relación con el sistema y las dependencias donde desenvuelven sus actividades los usuarios. Esta forma de encontrar datos, puede servir como presentación del analista si se realiza al iniciar el estudio, o bien como un término de comparación de lo que sucede en la dependencia con lo que los registros presentan como lo que debería suceder.

El término *registros*, se refiere a los manuales escritos sobre políticas, regulaciones y procedimientos de operaciones estándares que la mayoría de las organizaciones mantienen como guía de referencia para personal directivo y empleados. Los manuales que documentan o describen las operaciones para los procesos de datos existentes o sistemas de información que entran dentro del área de investigación, también proporcionan una visión sobre la forma en la que la organización debe conducirse. Normalmente muestran los requerimientos y restricciones del sistema y características de procesamiento. Los registros permiten que los analistas se familiaricen con algunas operaciones, oficinas de la organización y relaciones formales a las que debe darse apoyo. No obstante, no muestran cómo producen de hecho las actividades, dónde se ubica el poder verdadero para las decisiones, o cómo se realizan las tareas en la actualidad. Los otros métodos con objeto de encontrar datos estudiados en esta sección, son más eficaces para proporcionar al analista este tipo de información.

En la mayor parte de las organizaciones, los manuales y estándares sobre procedimientos de operación, usualmente son obsoletos; a menudo no se mantienen actualizados lo suficiente para poder señalar los procedimientos existentes. Los organigramas, muestran cómo las diferentes unidades de la organización *deberían* relacionarse con otras, pero muchas, no reflejan las operaciones actuales.

Otro componente que aparece en estos manuales, está constituido por los formularios estándares utilizados por la organización para su manejo interno.

Estos documentos y formularios (aun los que están en blanco) utilizados en la organización, proporcionan sin lugar a dudas, una porción importante del grupo de datos que son manejados dentro del sistema. Al comparar los formularios en blanco con el procedimiento y los manuales de operación, se muestra al analista cómo deben llenarse; por lo tanto, recolectar y comparar los formularios ya completos y los informes, permite señalar cualquier variación entre el uso de hecho y el prescrito en los documentos. A veces existen diferencias. Algunos espacios de los formularios se dejan en blanco; otras se completan con diferentes datos de los solicitados y otras se distribuyen en oficinas distintas a aquellas que se había prescrito. En algunas investigaciones para encontrar la causa de un problema operativo específico, los analistas deben ser capaces de detectar estas causas y encontrar las diferencias entre los procedimientos reales y los planificados.

Puede ser muy revelador realizar un *muestreo* de algunos de los documentos o formularios utilizados en los procesos. Los analistas pueden obtener copias para observar qué hay y qué falta. Analizar un conjunto de copias seleccionadas al azar, permite el cálculo de porcentajes de error y las estadísticas que describen en forma completa una situación. Al reunirse con mucha información de este tipo, es aconsejable catalogar los documentos a fin de determinar cuáles se utilizan y cuándo, y cuáles no se utilizan después de llenados. Este esfuerzo extra, puede ser difícil e insumir mucho tiempo si el número de documentos es elevado, pero establecer un diagrama de procedimientos y flujos, puede resultar de gran utilidad para lograr una visión integral de todas las tareas.

También se deben buscar los formularios y documentos que desconoce la mayoría. Son aquellos que los individuos elaboran para su propio uso, pero que no son parte de los procedimientos preestablecidos. Pueden ser señales de ineficacia de los métodos estándares o causas de problemas.

Observación

Leer en relación con una actividad de una organización, le proporciona al analista una dimensión de las actividades del sistema. Entrevistar personas, ya sea directamente o a través de cuestionarios, también le ayuda y le dice algo más. Ninguno de los dos métodos da una información completa. La observación proporciona información de primera mano en relación con la forma en que se llevan a cabo las actividades. Las preguntas sobre el uso de documentos, la manera en la que se realizan las tareas y si ocurren los pasos específicos como se preestablecieron, puede contestarse rápidamente si se observan las operaciones. Un analista que desea conocer lo que hace un gerente de alto nivel cuando decide si aceptará una oferta por ejemplo, debe observar el tipo de información que requiere o solicita, qué tan pronto le llega, si le llega o no en término y de dónde viene. También es importante saber si esta información es utilizada o no. Cada pregunta que el analista se plantee no se contestará de esta manera, pero la información que puede obtenerse, no estaría disponible de otra forma.

La observación es muy útil cuando el analista necesita ver de primera mano, cómo se manejan los documentos, cómo se llevan a cabo los procesos y si

ocurren los pasos especificados. Debe tratar de captar quien utiliza los documentos y si encuentran dificultades en su uso. También deberá detectar cuáles son los documentos o registros que no se utilizan. Otro elemento que se debe identificar, está constituido por las tareas problemáticas que llevan a los empleados a cometer errores con frecuencia al completarlas, así como aquellas que tienden a retardar el procedimiento.

Uno de los problemas que puede presentarse, es que el observado puede cambiar la forma en que el trabajo se lleva a cabo; esto es normal. En la observación participativa, los analistas se convierten en parte de la situación y hacen el trabajo por sí mismos. La participación les proporcionará impresiones de primera mano del trabajo diario del personal del mostrador. Será informativo en la medida que los analistas puedan posteriormente retroceder y ser objetivos en cuanto a lo que han experimentado. Cuando terminan, los analistas poseerán información adicional que no estaría disponible por medio de ninguno de los otros métodos de recopilación de datos ya vistos.

3.2.1.1.4 Especificación de requerimientos

Consiste en la actividad de transcribir la información recopilada durante la actividad del análisis en un documento que define un conjunto de requerimientos. En este documento se incluyen dos clases de requerimientos. Los requerimientos de usuario son informes abstractos de requerimientos del sistema para el cliente y el usuario final del sistema; y los requerimientos de sistema son una descripción detallada de la funcionalidad a ofrecer.

3.2.1.1.5 Validación de requerimientos

Esta actividad verifica que los requerimientos sean realistas, coherentes y completos. Durante este proceso es inevitable descubrir errores en el documento de requerimientos y en consecuencia deben ser modificados y corregidos.

Las actividades del proceso de requerimientos no se realizan simplemente en una secuencia estricta. El análisis de requerimientos continúa durante la definición y especificación y a lo largo del proceso salen a la luz nuevos requerimientos; por lo tanto las actividades de análisis, definición y especificación están vinculadas. En los métodos ágiles, los requerimientos se desarrollan de manera incremental según las prioridades del usuario, en tanto que la obtención de requerimientos proviene de los usuarios que son parte del equipo de desarrollo.

3.2.1.2 Uso de modelos

En toda esta etapa suelen utilizarse modelos de sistemas ya que ayudan a aclarar lo que hace el sistema existente y pueden utilizarse como base para discutir sus fortalezas y debilidades. Los modelos del sistema nuevo se emplean durante toda la etapa de

ingeniería de requerimientos para ayudar a explicar los requerimientos propuestos a otros participantes del sistema. Los ingenieros usan tales modelos para discutir las propuestas de diseño y documentar el sistema para la implementación. El aspecto más importante de un modelo del sistema es que deja fuera los detalles ya que un modelo es una abstracción del sistema a estudiar y no una representación de dicho sistema. De manera ideal, una representación de un sistema debe mantener toda la información sobre la entidad a representar. Una abstracción simplifica y recoge deliberadamente las características más destacadas dejando de lado las que no lo son o no resultan pertinentes. Existe una gran variedad de modelos que pueden utilizarse y que dependen del aspecto que se quiere representar como ser:

- Una perspectiva externa (modelo de contexto o entorno del sistema)
- Una perspectiva de interacción donde se modele la interacción entre un sistema y su entorno o entre los componentes de un sistema
- Una perspectiva estructural donde se modelen la organización de un sistema o la estructura de datos que procese el sistema
- Una perspectiva de comportamiento donde se modele el comportamiento dinámico del sistema y cómo responde ante ciertos eventos.

Cuando se desarrollen modelos de sistema no es necesario apegarse rigurosamente a los detalles de una notación. El detalle y el rigor de un modelo dependen de cómo lo use. Los modelos gráficos se utilizan con muchísima frecuencia:

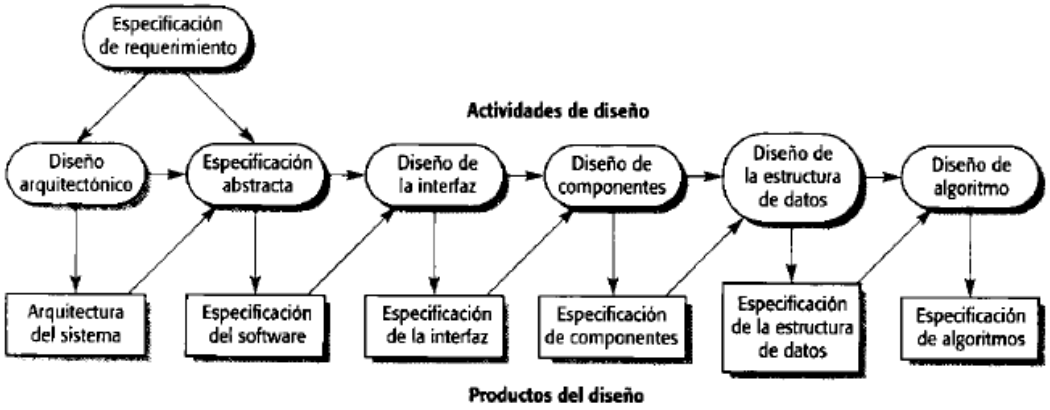
- Como medio para facilitar la discusión sobre un sistema existente o propuesto.
- Como una forma de documentar el sistema (planos del sistema)
- Como una descripción detallada del sistema que sirve para generar una implementación del sistema.

Distintos tipos de modelos se verán en los temas subsiguientes. La mayoría de los modelos existentes cuentan con herramientas de software que permiten construirlos y mantenerlos de una manera integrada (herramientas CASE - *Computer Aided Software Engineering*).

3.2.2 Diseño e implementación del software

La etapa de implementación de desarrollo de software corresponde al proceso de convertir una especificación del sistema en un sistema ejecutable. Siempre incluye procesos de diseño y programación de software aunque también puede involucrar la corrección de una especificación del software si se utiliza un enfoque incremental de desarrollo.

Un diseño de software se entiende como una descripción de la estructura del software que se va a implementar, los modelos y las estructuras de datos utilizados por el sistema, las interfaces entre componentes del sistema y, en ocasiones, los algoritmos usados. Los diseñadores no llegan inmediatamente a una creación terminada, sino que desarrollan el diseño de manera iterativa. Agregan formalidad y detalle conforme realizan su diseño con vueltas atrás constantemente para corregir diseños anteriores.



La figura es un modelo abstracto de este proceso que ilustra las entradas al proceso de diseño (actividades de diseño) y los documentos generados como salidas de este proceso. Sugiere que las etapas del proceso de diseño son secuenciales. De hecho, las actividades del proceso de diseño están vinculadas. En todos los procesos de diseño es inevitable la retroalimentación de una etapa a otra y la consecuente reelaboración del diseño.

El proceso de diseño puede implicar el desarrollo de varios modelos del sistema con diferentes niveles de abstracción. Mientras se descompone un diseño, se descubren errores y omisiones de las etapas previas. Esta retroalimentación permite mejorar los modelos de diseño previos. La figura es un modelo de este proceso que muestra las descripciones de diseño que pueden producirse en varias etapas del diseño. Este diagrama sugiere que las etapas son secuenciales. En realidad, las actividades del proceso de diseño se entrelazan. La retroalimentación entre etapas y la consecuente repetición del trabajo es inevitable en todos los procesos de diseño.

Una especificación para la siguiente etapa es la salida de cada actividad de diseño. Esta especificación puede ser abstracta y formal, realizada para clarificar los requerimientos, o puede ser una especificación para determinar qué parte del sistema se va a construir. Durante todo el proceso de diseño se detalla cada vez más esta especificación. El resultado final del proceso son especificaciones precisas de los algoritmos y estructuras de datos a implementarse.

Las actividades específicas del proceso de diseño son:

1. **Diseño arquitectónico**
   Aquí se identifica la estructura global del sistema, los principales componentes (subsistemas o módulos) sus relaciones y cómo se distribuyen.

2. **Especificación abstracta**
   Para cada subsistema se produce una especificación abstracta de sus servicios y las restricciones bajo las cuales debe funcionar.

3. **Diseño de interfaz**
   Para cada subsistema se diseña y documenta su interfaz con otros

subsistemas. Esta especificación de la interfaz debe ser inequívoca ya que permite que el subsistema se utilice sin conocimiento de su funcionamiento.

4. **Diseño de componentes**
   Se toma cada componente del sistema y se diseña cómo funcionará. Esto puede ser un simple dato de la funcionalidad que se espera implementar, y al programador se le deja el diseño específico.

5. **Diseño de la estructura de datos**
   Se diseña en detalle y especifica la estructura de datos utilizada en la implementación del sistema.

6. **Diseño de algoritmos**
   Se diseñan en detalle y especifican los algoritmos utilizados para proporcionar los servicios.

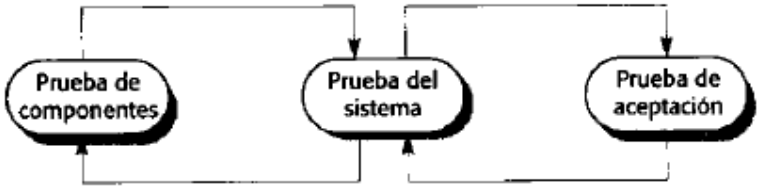
Las salidas conducen a un conjunto de especificaciones de diseño y el detalle y la representación de las mismas varía entre distintos tipos de sistemas. En algunos casos hacen falta documentos detallados que establezcan descripciones exactas del sistema. Estas especificaciones pueden representarse en diagramas o modelos de los que puede generarse de manera automática cierta codificación.

Con esta información puede realizarse la programación. Ésta es una actividad personal y no hay proceso que se siga de manera general. Algunos programadores comienzan con componentes que entiende, los desarrollan y luego, cambian hacia componentes que entienden menos. Otros dejan hasta lo último los componentes familiares. A algunos les agrada definir con anticipación datos en el proceso, que luego usan para probar en el desarrollo del programa. Por lo general lo programadores realizan algunas pruebas de código. Esta actividad es la depuración (*debugging*). La prueba de defectos y la depuración son procesos diferentes. La primera establece la existencia de defectos mientras que la segunda se dedica a localizar y corregirlos. Cuando se depura, se debe elaborar una hipótesis sobre el comportamiento observable del programa y luego, poner a prueba dicha hipótesis con la esperanza de encontrar la falla que causó la salida anómala. Poner a prueba la hipótesis quizá requiera rastrear manualmente el código del programa, o bien, tal vez se necesiten nuevos casos de prueba para localizar el problema. Con la finalidad de apoyar el proceso de depuración, se deben utilizar herramientas interactivas que muestren valores intermedios de las variables del programa así como el rastro de las instrucciones ejecutadas.

3.2.3 Validación del software

La verificación y validación del software se crea para mostrar que un sistema cumple tanto con sus especificaciones como las expectativas del cliente. Las pruebas de programa donde el sistema se ejecuta a través de datos de prueba simulados, son la principal técnica de validación. Esta última también puede incluir procesos de comprobación como inspecciones y revisiones en cada etapa del proceso de software, desde la definición de requerimientos del usuario hasta el desarrollo del programa.

Con excepción de los programas pequeños, los sistemas no deben ponerse a prueba como una unidad monolítica. La siguiente figura muestra un proceso de prueba de tres etapas donde los componentes del sistema se ponen a prueba; luego se hace lo mismo con el sistema integrado y finalmente el sistema se pone a prueba con los datos del cliente.



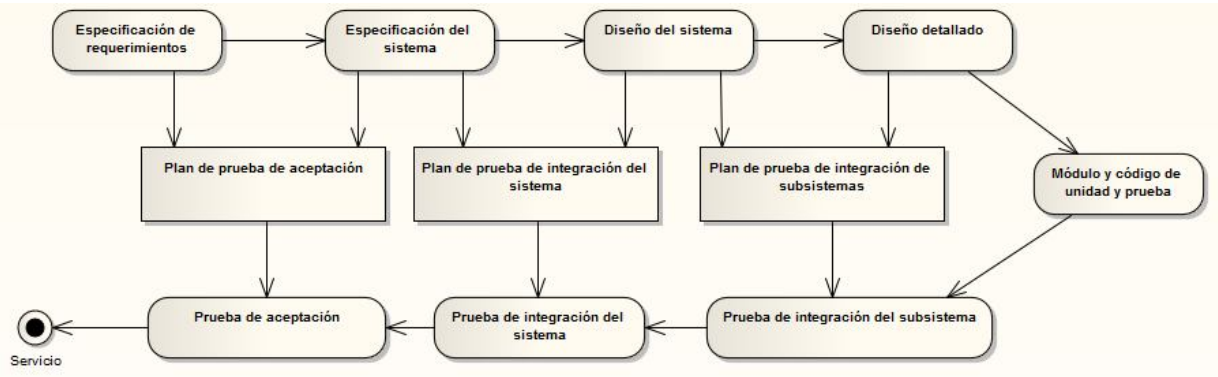
De manera ideal, los defectos de los componentes se detectan oportunamente en el proceso, en tanto que los problemas de interfaz se localizan cuando el sistema se integra. Sin embargo, conforme se descubran los defectos, el programa deberá depurarse y esto quizá requiera la repetición de otras etapas en el proceso de pruebas. Los errores en los componentes del programa pueden salir a la luz durante las pruebas del sistema. En consecuencia, el proceso es iterativo, con información retroalimentada desde etapas posteriores hasta las partes iniciales del proceso. Las etapas en el proceso de pruebas son:

1. **Prueba de desarrollo**
   Las personas que desarrollan el sistema ponen a prueba los componentes que lo constituyen. Cada componente se prueba de manera independiente. Éstos pueden ser simples entidades como funciones o clases de objeto o agrupamientos coherentes de dichas entidades. Por lo general se utilizan herramientas de automatización de pruebas como JUnit, que pueden volver a correr pruebas de componentes cuando se crean nuevas versiones.
2. **Pruebas del sistema**
   Los componentes del sistema se integran para crear un sistema completo. Este proceso tiene la finalidad de descubrir errores que resulten de interacciones no anticipadas entre componentes y problemas de interfaz entre ellos, así como de mostrar que el sistema cubre sus requerimientos y poner a prueba las propiedades emergentes del sistema. Para sistemas grandes, esto puede ser un proceso de múltiples etapas, donde los componentes se juntan para formar subsistemas que se ponen a prueba de manera individual antes de que dichos subsistemas se integren para establecer el sistema final.
3. **Pruebas de aceptación**
   Ésta es la etapa final en el proceso de pruebas, antes de que el sistema se acepte para uso operacional. El sistema se pone a prueba con datos suministrados por el cliente del sistema en vez de datos de prueba simulados. Las pruebas de aceptación revelan los errores y las omisiones en la definición de requerimientos del sistema, ya que los datos reales ejercitan el sistema de manera diferente a los datos de prueba. Asimismo, las pruebas de aceptación revelan problemas de requerimientos, donde las instalaciones del sistema en realidad no cumplan las necesidades

del usuario o cuando sea inaceptable el rendimiento.

Por lo general, los procesos de desarrollo y de pruebas de componentes están entrelazados. Los programadores construyen sus propios datos de prueba y experimentan el código de manera incremental conforme lo desarrollan.

Si se usa un enfoque incremental para el desarrollo, cada incremento debe ponerse a prueba conforme se diseña y tales pruebas se basan en los requerimientos para dicho incremento. Si se sigue un plan, las pruebas se realizan mediante un conjunto de planes de prueba. Un equipo independiente de examinadores trabaja con base en dichos planes preformulados que se desarrollaron a partir de la especificación y el diseño del sistema. En la siguiente figura se muestra cómo se vinculan los planes de prueba entre las actividades de prueba y desarrollo.



A esto se lo conoce como modelo de desarrollo **V.**

En ocasiones a las pruebas de aceptación se les identifica como ***pruebas alfa***. Los sistemas a medida se desarrollan solamente para un cliente; el proceso de prueba alfa continúa hasta que el desarrollador del sistema y el cliente estén de acuerdo en que el sistema entregado es una implementación aceptable de los requerimientos.

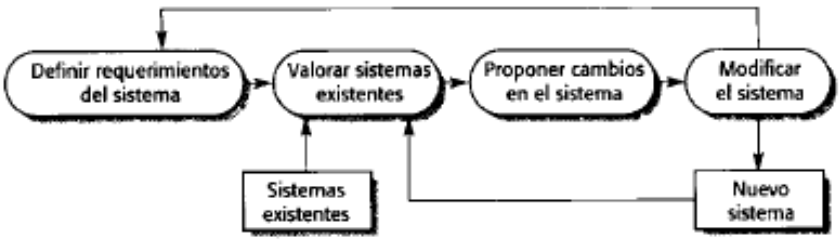
Cuando un sistema se marca como producto de software, se utiliza con frecuencia un proceso de prueba llamado ***prueba beta***. Ésta incluye entregar un sistema a algunos clientes potenciales que están de acuerdo con usar el sistema. Ellos reportan los problemas a los desarrolladores. Dicho informe expone el producto a uso real y detecta errores que no fueron anticipados. Después de esta retroalimentación, el sistema se modifica y libera ya sea para más pruebas beta o para su venta general.

3.2.4 Evolución del software

La flexibilidad de los sistemas de software es una de las razones principales por las que cada vez más software se incorpora en los sistemas grandes y complejos. Una vez tomada la decisión de fabricar hardware, resulta muy costoso hacer cambios a su diseño. Sin embargo, en cualquier momento durante o después del desarrollo del sistema, pueden hacerse cambios en el software. Incluso los cambios mayores son todavía más baratos que los correspondientes cambios al hardware del sistema.

En la historia siempre ha habido división entre el proceso de desarrollo del software y su proceso de evolución (mantenimiento). Las personas consideran el desarrollo de software como una actividad creativa en la cual se diseña un sistema desde un concepto inicial y a través de un método de trabajo. No obstante, consideran en ocasiones el mantenimiento de software como insulso y poco interesante. Aunque en la mayoría de los casos los costos de mantenimiento son varias veces los costos iniciales del desarrollo, los procesos de mantenimiento se consideran en ocasiones como menos desafiantes que el desarrollo de software original.

Esta distinción entre desarrollo y mantenimiento es cada vez más irrelevante. Es muy difícil que cualquier sistema de software sea un sistema completamente nuevo y tiene mucho más sentido ver el desarrollo y el mantenimiento como un continuo. En lugar de dos procesos separados, es más realista pensar en la ingeniería de software como un procesos evolutivo donde el software cambia continuamente a lo largo de su vida en función de los requerimientos y las necesidades cambiantes del cliente y el medioambiente.



Universidad Nacional del Litoral

FACULTAD DE INGENIERÍA
Y CIENCIAS HÍDRICAS

Ingeniería en Informática

Ingeniería de Software I

TEMA IV – El modelo de procesos estructurado

Introducción

Por muchas razones la captura de los requerimientos es la parte más dificultosa del desarrollo de un sistema de software. No se trata simplemente de la dificultad técnica del trabajo, si bien muchos proyectos exigen que el ingeniero tenga conocimientos profundos de la tecnología actualizada de procesamiento de datos. No se trata simplemente de las dificultades políticas que se presentan, especialmente en grandes proyectos, donde el nuevo sistema deberá servir a varios grupos de interés, posiblemente en conflicto. Tampoco se trata solamente de los problemas de comunicación que surgen en cualquier situación en la cual personas de diferentes antecedentes, con distintos enfoques del contexto y diferentes vocabularios deban trabajar juntas. Es la resultante de estas dificultades lo que hace que la definición de los requerimientos sea tan dura y exigente: el hecho es que el ingeniero debe hacer de intermediario entre la comunidad de los usuarios (aquellos que tienen la sensibilidad de sus problemas, pero encuentran dificultoso explicarlo) y el equipo técnico de desarrollo. El ingeniero debe hacer un balance entre aquello que es actualmente posible tecnológicamente y aquello que vale la pena hacer en la organización tal como está dirigida por sus administradores.

Si se hace un balance que resulte aceptable por todas las partes y que pueda resistir la prueba del tiempo, se ha logrado la parte más ardua del esfuerzo; si ha sido bien hecho, cualesquiera sean las dificultades de las distintas actividades de desarrollo, el sistema que se construya servirá a las necesidades de la organización. Si ha sido hecho pobremente, cualquiera sea la excelencia del software desarrollado, el sistema no satisfará las necesidades de la organización y fracasará en su propósito.

4.1 Los problemas iniciales del análisis

- El ingeniero suele encontrar muy difícil aprender lo suficiente de la organización para poder ver los requerimientos del sistema a través de los ojos del usuario. El temor es en el sentido de que al final puede haberse hecho un sistema técnicamente excelente pero no era lo que el usuario deseaba o necesitaba.

- Los usuarios suelen no saber lo que quieren o bien si lo saben con claridad, no lo transmiten de manera adecuada. Consecuentemente, el ingeniero debe de alguna manera interactuar con ellos y reflejarle de alguna forma lo que fue entendiendo en el avance del estudio. Suele entonces ser difícil mostrarle al usuario "la idea" de lo que hace actualmente el sistema, y de lo que hará. Las herramientas que se verán a continuación permitirán producir un modelo gráfico de un sistema que jugará el rol de esa "idea" constituyendo algo así como ***los planos del sistema***.

- El ingeniero puede rápidamente verse abrumado por los detalles, tanto de la organización como por los detalles técnicos del nuevo sistema. La mayor parte del tiempo de la fase de captura de requerimientos, se emplea en obtener información detallada de la situación actual, los procedimientos administrativos, los documentos de entrada, los informes producidos y requeridos, las políticas en uso y los miles de hechos que surgen de una cosa tan compleja como una organización real. A menos que exista algún esquema o estructura para organizar estos detalles, el ingeniero se verá sobrecargado con hechos y papeles. Los detalles son necesarios y deberán estar disponibles cuando se los requiera por lo

que se deberá contar con herramientas para controlar los detalles.

- Los documentos donde se ubican los detalles del nuevo sistema (llamado de distintos nombres tales como: especificación de sistema, diseño general, especificación funcional, especificación de requerimientos, etc.) no son fácilmente comprensibles por los usuarios debido a su volumen y a eventualmente a algunos conceptos técnicos que contiene. En consecuencia, suelen decir que están de acuerdo con lo descrito. A la hora de entregar el sistema terminado (sobre todo si se utiliza un ciclo de vida en cascada), tendrán algo que sí pueden entender y recién entonces podrán reaccionar. Demasiado tarde por supuesto.

- Si el documento de especificaciones pudiera escribirse de manera que tuviera sentido para los usuarios, quizá no sería de mucha utilidad para los diseñadores físicos y programadores que deben construir el sistema. Suele entonces comenzarse a especificar el diseño físico antes que el modelo lógico del sistema este terminado, dando como resultado un sistema de baja calidad.

Aun contando con las mejores herramientas analíticas posibles, aparecerán algunos de los problemas enunciados. No existen herramientas analíticas que permitan al ingeniero saber lo que tiene en su mente el usuario si éste no se lo transmite. No obstante mediante la aplicación de ciertas herramientas lógicas, los problemas de análisis se verán facilitados en gran medida.

Puede ser difícil entender (y demostrarle al usuario que se entendió) por completo un proceso de la organización a través de una descripción verbal solamente; éste es el motivo fundamental por el que se recurre a una forma de representación gráfica que los ingenieros y los usuarios puedan comprender y leer fácilmente y que además ayudan a ilustrar los componentes esenciales de un proceso y la forma en la que interactúan.

Concretamente, se tratará una forma de mostrar a los usuarios un modelo tangible vívido del sistema, de forma tal que este grupo de usuarios se imagine lo que podrá hacer cuando el sistema esté en operación. En este tema se desarrollará una metodología que permita realizar diagramas para el modelado de las funciones o procesos que se dan dentro del sistema. La nomenclatura que se desarrolla es la clásica de los autores *Chris Gane & Trish Sarson*.

4.2 Modelo de análisis de procesos (*Process Analyst Models*)

La característica fundamental de los PAM es que no muestran los procesos en serie, vale decir que un PAM no comienza ni termina jamás. Existen tareas en una organización, que pueden ser efectuadas en paralelo y no una después de la otra presentando una secuencia. Los PAM, permiten reflejar perfectamente tales situaciones. La terminología modelo de análisis de procesos es utilizada en función del nombre que adoptan estos modelos en las más conocidas herramientas CASE. No obstante, el nombre con el que se ha difundido este tipo de gráficos es el de DFD (diagrama de flujo de datos).

Como su nombre lo sugiere, los diagramas de flujo de datos se concentran en los datos que se mueven a través del sistema y no en los dispositivos o equipos. Por medio del proceso del entendimiento del área de aplicación, el ingeniero identifica y describe los

datos que se mueven a través del sistema, por qué se introducen o se retiran y qué proceso se está llevando a cabo con ellos. Es muy importante determinar cuando entran o se retiran los datos del área de aplicación. Algunas veces los datos se almacenan para su uso posterior o su consulta en almacenamientos previos.

4.2.1 Ventajas del método

Estas notaciones sencillas, son fácilmente entendidas por los usuarios y personas de la organización que son parte del proceso que se estudia; por lo tanto, el ingeniero puede trabajar con los usuarios y realmente hacer que participen en el estudio del flujo de datos. Los usuarios hacen sugerencias sobre modificaciones de los diagramas para describir en forma más exacta la actividad de la organización; también pueden examinar los diagramas y señalar los problemas rápidamente, de manera que se puedan corregir antes de que se siga avanzando en el estudio. Si los problemas no se localizan en forma anticipada en el proceso de desarrollo, será muy difícil corregirlos posteriormente; incluso, evitarlos a tiempo, puede prevenir una falla del sistema.

El análisis del flujo de datos, permite aislar las áreas de interés de la organización, estudiarlas y examinar los datos que entran a los procesos y ver cómo se modifican cuando salen del mismo. Conforme se recopilan hechos y detalles, su creciente conocimiento de los procesos lleva al ingeniero a realizar preguntas sobre partes más específicas de ellos, que a su vez los conduce a una investigación adicional.

Una investigación de sistemas elaborada en forma amplia, produce conjuntos de muchos DFDs, algunos de los cuales proporcionan una visión general de los procesos principales, y otros dan un mayor detalle para señalar elementos de datos, almacenamiento de éstos y etapas del proceso para componentes específicos de un sistema mayor. Si el ingeniero desea revisar posteriormente el sistema en su totalidad, utiliza los diagramas generales y si está interesado en estudiar un proceso en particular, utilizará los diagramas de los procesos de menor nivel.

Los niveles de los diagramas pueden ser comparados con los mapas de carreteras y calles que se utilizan cuando se viaja en un área que no es conocida. En un viaje largo, primero se usa un mapa tal vez de nivel nacional, que indica las principales carreteras y ciudades. Conforme se acerca al lugar que quiere visitar, se necesitará seguramente un mapa más detallado que indique las rutas provinciales y caminos secundarios que permitan recorrer circuitos turísticos. Una vez que han arribado a la ciudad en la que se detendrán, un mapa de nivel nacional no les servirá para ubicarse dentro de esa localidad. Tampoco lo hará el mapa más específico con circuitos turísticos. Será necesario un plano detallado de la ciudad con el nombres de las calles y en el que se destaquen los lugares interesantes de conocer como sitios históricos, museos, centros comerciales y que contengan además, las indicaciones de cómo llegar a ellos. Esta información es esencial para encontrar las direcciones correctas de las calles; no obstante, no es útil cuando se inicia el viaje y se empieza a tener problemas para obtener una orientación. Los DFDs se utilizan de la misma forma. Se desarrollan y usan de manera progresiva, que va de lo general a lo particular para el sistema de interés.

Los DFDs muestran las características lógicas de las aplicaciones, es decir, señalan **qué** ocurre sin especificar el cómo. En otras palabras, los detalles físicos como

métodos de almacenamiento, dispositivos de entrada o componentes informáticos, no son aspectos principales para el ingeniero durante la parte inicial del desarrollo; sólo después de entender los componentes lógicos, deben estudiarse los componentes físicos.

4.2.2 Desventajas del método

El mayor inconveniente presentado, es que poco se indica acerca de los datos que participan. No dicen nada acerca del tipo que son, qué asociaciones lógicas se establecen entre ellos, cómo se vinculan, cómo se afectan, etc. En estos gráficos, lo fundamental es la representación de los procesos que acontecen dentro del sistema sin importar la naturaleza de los datos participantes. Para ello son utilizadas otras herramientas de modelado de datos que se representarán con otros tipos de modelos.

4.3 Objetos del modelo

4.3.1 Las entidades externas

Las entidades externas son clases lógicas de cosas o de personas, las cuales representan una fuente o destino de datos. Por ejemplo: clientes, empleados, proveedores, contribuyentes, etc. También pueden ser una fuente o destino específico, por ejemplo, departamento de Contabilidad, DGI, depósito, etc. De acuerdo a lo descrito en los sistemas automatizados de información integrados, es totalmente válido que la entrada de flujo de información al sistema que se está considerando, provenga desde la salida de otro sistema. En ese caso, ese otro sistema emisor de información es una entidad externa.

Una entidad externa puede simbolizarse con un cuadrado o rectángulo de líneas llenas (suelen utilizarse también rectángulos con el lado inferior y el derecho de doble ancho o sombreado). La entidad puede identificarse con una letra minúscula en al ángulo superior izquierdo como referencia. En el interior, debe aparecer el nombre de la entidad. Por ejemplo la entidad externa CLIENTE, quedaría representada por:

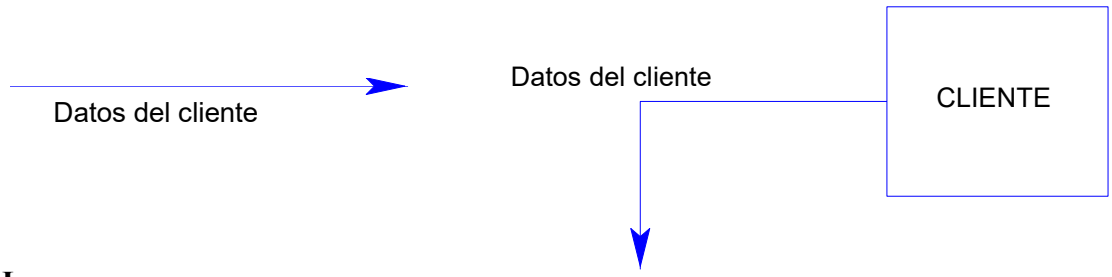
Suele ocurrir que por la topología presentada en el diagrama, se requiera repetir un símbolo representando a una entidad externa que ya ha sido definida; para ello se crea una entidad externa sinónimo cuya única diferencia consiste en una línea inclinada en el ángulo inferior derecho del rectángulo. Para el caso de repetir la entidad externa CLIENTE, el símbolo correspondiente será:

Mediante la designación del objeto entidad externa, se está estableciendo implícitamente que tal elemento se encuentra fuera de los límites del sistema considerado. A medida que se avance en el análisis y se aprenda más sobre los objetivos del usuario, se tomarán algunas entidades externas y se introducirán en el diagrama de flujo de datos del sistema, o por el contrario, se dejará de considerar alguna parte de las funciones del sistema designándola como una entidad externa con datos que fluyan desde y hacia ella.

4.3.2 Los flujos de datos

El flujo de datos se simboliza mediante una flecha, preferentemente horizontal y/o vertical, con la punta indicando la dirección del flujo. Para mayor claridad, y especialmente en los primeros borradores del diagrama, se utilizará una flecha con doble punta en lugar de dos flechas, cuando los flujos de datos estén apareados.

Los flujos de datos, se asemejan a una tubería por donde se envían paquetes de datos. Se puede hacer referencia a la línea del flujo de datos indicando los procesos, entidades o almacenes origen o destino, pero cada flujo de datos deberá tener indicada una descripción escrita de su contenido. El formato entonces de una línea de flujo de datos será:



4.3.3 Los procesos

Los objetos mencionados hasta el momento han sido las entidades externas que pueden ser fuente o destino de datos los que necesariamente entrarán o saldrán del sistema a través de una tubería de flujo de datos. Ahora bien, para la existencia del flujo de datos –datos en movimiento– será necesaria la presencia de algún agente que impulse o aspire por alguno de los extremos de la tubería y de esa manera se produzca el movimiento. Los agentes encargados de establecer el flujo de datos (en cualquier sentido), son los denominados procesos.

Resulta necesario describir la función de cada proceso y para tener una fácil

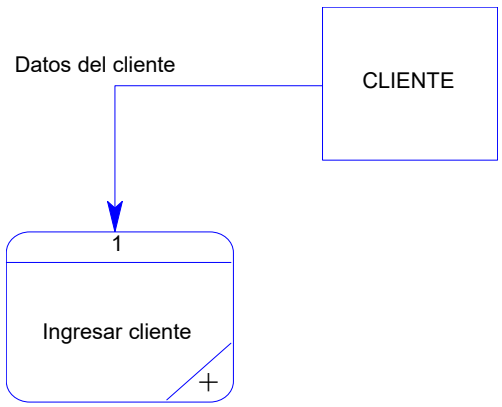
referencia de qué acción realiza, se le asigna una única identificación (vinculada en lo posible, a un sistema físico o a un proceso del sistema operativo). Los procesos pueden simbolizarse con un rectángulo vertical, con los ángulos redondeados, dividido opcionalmente en tres áreas:

- La primera de ellas consiste en un número de identificación atravesado de izquierda a derecha en la sección superior del símbolo. No existe dificultad alguna en asignar numeraciones a los procesos si se tiene en cuenta que algunos serán explotados en otros procesos más especializados, lo que implicará la asignación de nuevos números. Otro tipo de ocurrencia puede ser que durante el trabajo se detecten procesos similares que seguramente se unificarán con la consecuente desaparición de números en la secuencia que inicialmente se planteó. Una vez asignada la identificación del proceso, ésta no deberá ser cambiada excepto –como se indicara- para aperturas o unificaciones, dado que se utiliza como referencia para los flujos de datos y para la descomposición del proceso en niveles inferiores. En caso de trabajar con alguna herramienta CASE, la asignación de los números de procesos será de forma automática y algunos, suelen tener utilidades que permiten la remuneración total controlando la consistencia del modelo completo.

- La segunda, corresponde a la descripción o etiqueta que permite a través de una frase breve, dar una idea de la funcionalidad del mismo. Esta descripción de la función deberá hacerse con una frase imperativa, que consistirá idealmente en un verbo en infinitivo seguido de una cláusula objeto (cuanto más simple mejor).

- La tercera es opcional. Algunas herramientas CASE no disponen de ella o bien la utilizan con otra finalidad. Según la teoría de los autores, esta área es utilizada para indicar el lugar físico en el que tal tarea o proceso es llevado a cabo (si se tratase de una oficina o repartición), o bien el nombre o identificación del programa que actualmente posee esa funcionalidad en caso que el sistema esté informatizado. Para la metodología de trabajo propuesta, la utilidad de esta zona se resume solamente a indicar si ese proceso posee o no niveles de descomposición representados con un signo de suma (+) en el extremo inferior derecho.

Para representar un proceso, se ha tomado como ejemplo la entidad externa CLIENTE, el flujo de datos Datos del cliente, y el proceso Ingresar cliente:

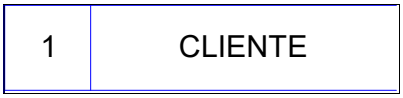


Nótese en el ejemplo, que la entidad externa CLIENTE pasa sus datos a través del conducto de flujo de datos Datos del cliente que es puesto en movimiento por el proceso Ingresar cliente. Es de destacar que el proceso mencionado posee descomposición, denotándose tal situación por la presencia del símbolo de suma (+) en el extremo inferior derecho del símbolo.

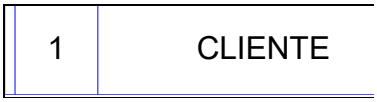
El ejemplo en cuestión es un absurdo, ya que un proceso responde a la definición de sistema, razón por la cual tendrá una o más entradas y necesariamente una o más salidas hacia otro destino. En la figura, el proceso Ingresar cliente no posee ninguna salida.

4.3.4 Los almacenes de datos

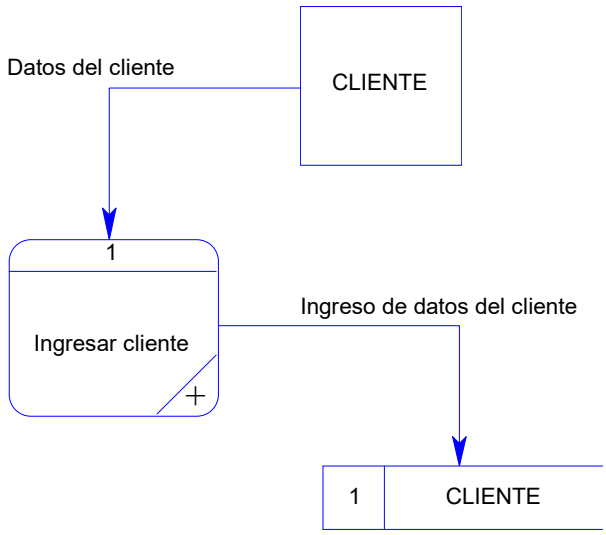
Sin tomar un compromiso físico durante el análisis, se encuentra que existen lugares donde resulta necesario definir datos para que puedan ser almacenados entre procesos. Estos lugares son los almacenes de datos. El símbolo que se adoptará para representar este objeto, consiste en dos líneas horizontales paralelas cerradas en el extremo izquierdo, preferentemente del ancho necesario para colocar el nombre descriptivo. Cada almacenamiento se indentificará con un número en un recuadro ubicado en el extremo izquierdo del gráfico para su fácil referencia. El nombre deberá ser lo suficientemente descriptivo para que el usuario note inmediatamente de qué se trata e interprete el contenido que posee.



Para evitar complicaciones con el cruce de líneas en un diagrama de flujo de datos, puede dibujarse el mismo almacenamiento de datos más de una vez en el diagrama, identificando los almacenes duplicados mediante líneas verticales adicionales colocadas a la izquierda como lo muestra la siguiente figura.



Cuando un proceso almacena datos, la flecha del flujo de datos se indica en la dirección al almacenamiento de datos. Cuando un almacenamiento de datos es accedido para ser leído únicamente, es suficiente con mostrar el grupo de elementos de datos recuperados como flujo de datos salientes. En definitiva, la posición de la punta de flecha de la línea de flujo, muestra el sentido del movimiento de los datos. A continuación se muestra cómo quedaría completo el proceso Ingresar cliente con ingreso de datos a partir de la entidad externa CLIENTE, el flujo de datos Datos del cliente e Ingreso de datos del cliente y el almacén CLIENTE.

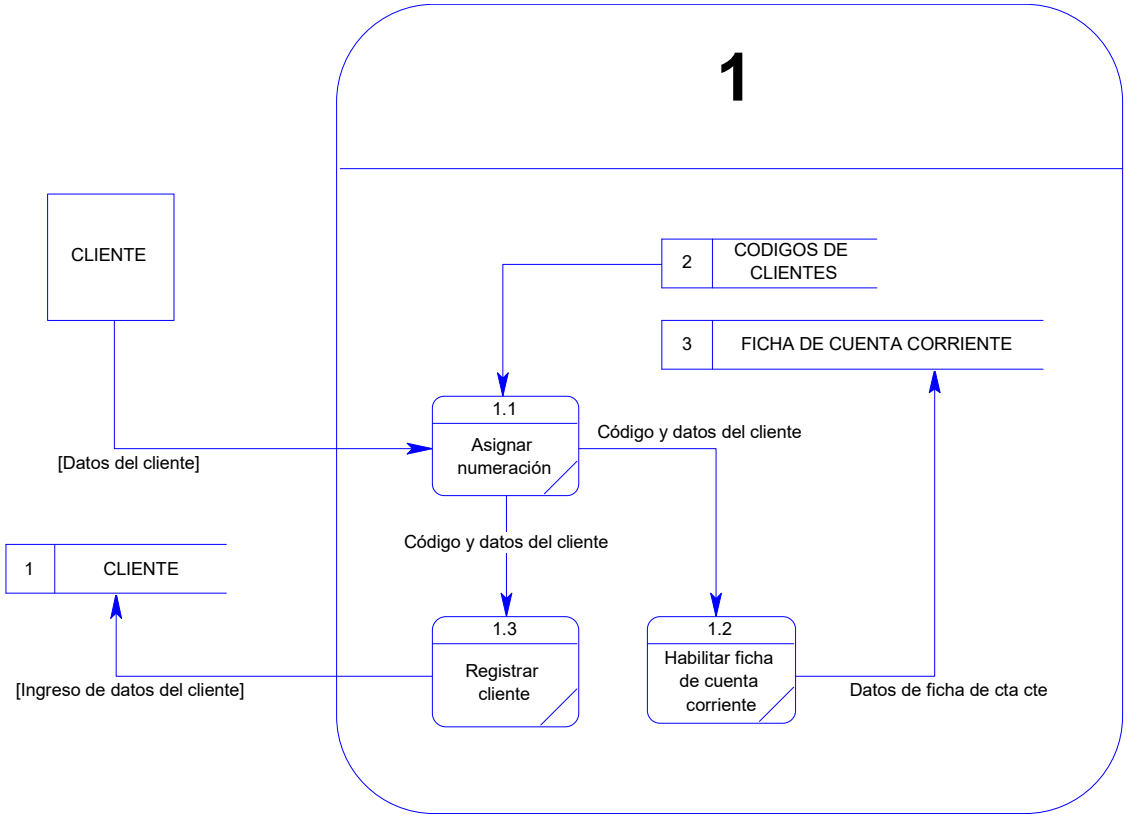


Nótese que el proceso toma los datos de la entidad externa y luego los ingresa en el almacén dispuesto para tal fin. Adviértase además que el proceso posee descomposición en procesos más sencillos o de menor nivel (signo +). Obsérvese además, que el diagrama en ningún lugar muestra cuáles ni de qué tipo son los datos del CLIENTE que se almacenan o ingresan. Esta situación es manejada por otro modelo distinto, no obstante es posible a través del uso de las herramientas de ingeniería de software asistido, la utilización de diccionarios de datos centralizados que pueden ser definidos en esta etapa.

4.4 La explosión de procesos

Un proceso de datos complejo, puede ser desagregado en otros procesos de menor jerarquía, pero que permitan reflejar con mayor claridad las tareas que se desarrollan. Tal procedimiento es denominado explosión de procesos, y su resultado es otro diagrama de flujo de datos. Cada proceso en el nivel inferior deberá estar relacionado, inversamente, con el proceso del nivel superior. Esto puede hacerse dándole a la casilla de proceso de nivel inferior un número de identificación que sea una fracción decimal de la casilla de proceso del nivel superior, por ejemplo, el 29 puede descomponerse en 29.1, 29.2, etc. y si es necesario llegar a un tercer nivel, 29.1.1, 29.1.2, etc.

La representación más clara de la explosión del proceso se obtiene dibujando los diagramas de flujo de datos de nivel inferior dentro de los límites que encuadran la casilla de proceso de nivel superior. Todos los flujos de datos entrantes y salientes de la casilla de proceso de nivel superior, deben entrar y salir a través del límite. Los flujos de datos que se muestran por primera vez en el nivel inferior, también pueden salir fuera de los límites. Por una cuestión organizativa y de nomenclatura, los almacenes de datos sólo se muestran dentro de los límites si son creados y accedidos por este proceso y no por otro.



Del ejemplo anterior, si se considera que el proceso Ingresar cliente puede ser descompuesto en:

1 Ingresar cliente:

1.1 Asignar numeración
1.2 Habilitar ficha de cuenta corriente
1.3 Registrar cliente

Cada uno de estos procesos, tienen su grupo de entradas y salidas todas interiores al proceso **1**. Las entradas y salidas al y desde el proceso a explotar, son:

[Datos del cliente] entrada desde la *entidad externa* CLIENTE
[Ingreso de datos del cliente] salida hacia el *almacén* CLIENTE

Nótese que han sido agregados los almacenes:

2 CODIGOS DE CLIENTES
3 FICHA DE CUENTA CORRIENTE

ya que solamente son accedidos dentro del diagrama de menor jerarquía. En el gráfico se representan las fronteras del proceso explotado y es de notar que la cantidad de entradas ([Datos del cliente]) y de salidas ([Ingreso de datos del cliente]) es la misma. Puede ocurrir que aparezcan más flechas de salida en el diagrama de menor nivel, pero que en el

nivel superior se corresponden con un agrupamiento lógico. Una situación similar puede presentarse con las entradas.

4.5 Desarrollo de los DFDs

Los primeros pasos en la determinación de requerimientos, como se ha indicado, se llevan a cabo para conocer las características generales de los procesos de la organización objeto del estudio. La parte de más alto nivel de los detalles, también se estudia. Conforme se comprenden mejor esos detalles, se profundiza y recopila más información específica y detallada. Se realizan preguntas específicas a un nivel de profundidad cada vez mayor: Este proceso se conoce como **análisis descendente** (*top-down*).

El paso de descripción de lo general a lo particular, se repite muchas veces en una investigación de sistemas: el entendimiento de un nivel de detalle, es extendido a un mayor detalle del siguiente nivel. En los grandes sistemas, un solo proceso, se puede extender muchas veces hasta que se describe una cantidad suficiente de precisiones para que de esta manera, pueda entenderse el proceso en su conjunto.

Los diagramas o modelos de análisis de procesos, no serán útiles si no se dibujan de forma apropiada o son difíciles de manejar. Cada nivel inferior se define utilizando de **tres** a **siete** procesos para explotar un proceso de mayor nivel. Utilizar más de siete, provoca que el diagrama sea difícil de manejar. Los diagramas son más fáciles de leer si la descripción ocupa una sola hoja de papel tipo A3. Si se necesitan más datos, el siguiente nivel puede extenderse.

4.5.1 Pautas para dibujar los diagramas de flujo de datos

1. Identificar las entidades externas involucradas. Ello implica decidir sobre un límite preliminar o fronteras del sistema. En caso de dudas, incluir dentro de los límites del sistema la primera "capa exterior" de los sistemas manuales o automatizados que tenga como interfaces.
2. Identificar las entradas y salidas programadas que se espera puedan producirse en el curso normal de la organización. Si la lista crece demasiado, seguramente existen agrupamientos lógicos de entradas y salidas.
3. Identificar consultas y pedidos de información que podrían surgir. Especificar un flujo de datos que defina la información "dada" al sistema y un segundo flujo de datos que indique que se "requiere" del sistema.
4. Jamás incluir decisiones en un DFD. De todas maneras, aunque se quiera, no se dispone de ningún mecanismo ni simbología para referenciarlas.
5. Duplicar entidades externas y almacenes de datos en caso de que se enmarañe demasiado el dibujo. Los diagramas deben resultar claros y deben evitarse los cruces de líneas de flujo de datos.
6. Producir una explosión de nivel inferior de cada proceso definido en otros diagramas, utilizando las convenciones especificadas anteriormente. Si es necesario, realizar modificaciones al diagrama de nivel superior.

4.6 Detalles de los componentes funcionales

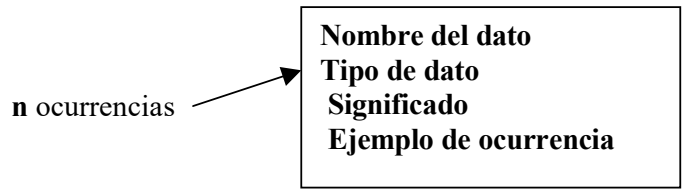
Aunque el DFD, proporciona una visión global bastante conveniente de los componentes funcionales del sistema, no da detalles de éstos. Para mostrar detalles acerca de qué información se transforma y de cómo lo hace, se ocupan dos herramientas textuales de modelado adicionales: el diccionario de datos y la especificación de procesos.

4.6.1 El diccionario de datos

Cualquiera de los objetos que son creados en la construcción del modelo de análisis de procesos, sean estos *entidades externas, flujos de datos, almacenes o procesos*, deberá estar registrados en algún lugar. Para el ejemplo seguido, se debe conocer qué significa la *entidad* externa CLIENTE, qué significa el flujo de datos Datos del cliente y además, será necesario conocer cuáles son esos datos del cliente a que se hace referencia. Esta información deberá estar organizada en un documento al que se lo denomina Diccionario de Datos, los que hoy en día y mediante el uso de herramientas CASE, su mantenimiento se hace de manera automática. En general un *diccionario de datos* deberá incluir:

- **Entidades Externas**
  Identificativo de la entidad externa
  Nombre de la entidad externa
  Significado
  Ejemplo de ocurrencia

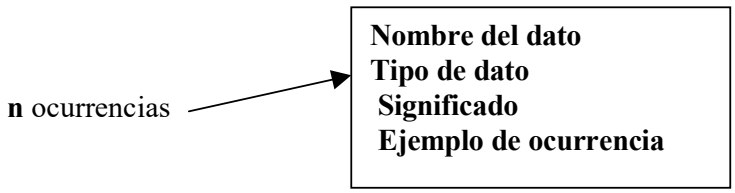
- **Flujos de datos**
  Identificativo del flujo de datos
  Nombre del flujo de datos
  Detalle de datos que transporta:



  Origen (*entidad externa, proceso, almacén***)**
  Destino (*entidad externa, proceso, almacén***)**

- **Procesos**
  Identificativo de proceso
  Nombre del proceso
  Descripción

- **Almacén**
  Identificativo del almacén
  Nombre del almacén
  Descripción
  Detalle de datos que aloja:



Es de notar que tanto las líneas de *flujo* y los *almacenes*, manejan la misma información: **datos**, radicando su diferencia que en el primer caso están en movimiento y en el segundo, son estáticos. Hasta el momento no se han tratado los ítems de datos. De acuerdo a lo especificado, los atributos de los ítems de dato son:

- **Ítems de dato:**
  Nombre del dato
  Tipo de dato
  Significado
  Ejemplo de ocurrencia

Al manejar los datos como otro elemento más que es componente de cualquiera de los objetos que lo pueden contener o mover, no se hace necesario especificarlos junto con los *flujos* y con los *almacenes*, sino que se agrupan separadamente. Esto además permite identificar en principio todos aquellos datos que están repetidos pero que significan distintas cosas o bien aquellos que tienen distinto nombre pero significan lo mismo (caso más difícil de controlar).

En definitiva, un diccionario de datos, es una lista de todos los elementos incluidos en el conjunto de los modelos de análisis de procesos que describen un sistema. El diccionario de datos, almacena detalles y descripciones de estos elementos. Si se desea conocer cuantos caracteres hay en un dato, con qué otros nombre se le conoce en el sistema, o en dónde se utilizan dentro del sistema, el diccionario de datos- si está desarrollado apropiadamente-, debe proporcionar esa información. El diccionario de datos, se desarrolla durante el análisis de los procesos y el análisis conceptual complementándose entre ambos.

4.6.2 La especificación de procesos

Un modelo de análisis de procesos o diagrama de flujo de datos, no muestra ningún tipo de decisión y además, no muestra el detalle del proceso de último nivel. Para modelar las decisiones y describir acabadamente el funcionamiento de un proceso, se recurre a la Especificación de Proceso. Este tipo de herramienta, es la que le proporciona mayor claridad al proceso que deba automatizarse mediante un programa de computadora.

La especificación de procesos, tiene como propósito definir lo que debe hacerse para transformar entradas en salidas. Es una descripción detallada de la política de negocios del usuario que cada proceso lleva a cabo. Existen varias metodologías que consideran la especificación de procesos, pero, todas deben satisfacer dos requerimientos fundamentales:

- La especificación del proceso debe expresarse de una manera que puedan verificar tanto el usuario como el analista. Precisamente por esta razón se evita el lenguaje narrativo como herramienta de especificación: es notoriamente ambiguo, sobre todo si describe acciones alternativas (decisiones) y acciones repetitivas (ciclos). Por naturaleza, tiende a causar gran confusión cuando expresa condiciones booleanas compuestas (combinaciones de operadores *AND*, *NOT* y *OR*).

- El proceso debe especificarse en una forma que pueda ser comunicada efectivamente al público amplio que esté involucrado. A pesar de que el analista es típicamente quien escribe la especificación del proceso, habitualmente será un público bastante diverso de usuarios, administradores, auditores, personal de control de calidad y otros, el que leerá la especificación de proceso.

Existe una gran interacción y superposición entre la especificación de procesos y la explosión de procesos. Las especificaciones de proceso sólo se desarrollan para los procesos de más bajo nivel en un conjunto de diagramas por niveles en un modelo de análisis de procesos. Además, la especificación de proceso para un proceso de nivel superior, es el diagrama de análisis de procesos de nivel inferior. Escribir una especificación de proceso adicional, sería superfluo y redundante; esto es, crearía una especificación más difícil de actualizar.

La principal herramienta para la especificación de procesos - no implicando que sea la única- es la descripción mediante un lenguaje estructurado (pseudo código). Su propósito, es hacer un balance razonable entre la precisión del lenguaje formal de programación y la informalidad y legibilidad del lenguaje cotidiano. Una frase en lenguaje estructurado puede consistir en una ecuación algebraica, o en una sencilla frase imperativa que consista en verbo y un objeto. Los verbos deben escogerse de entre un pequeño grupo de verbos orientados a la acción tales como: conseguir (o aceptar o leer), poner (o mostrar o escribir), encontrar (o buscar o localizar), sumar, restar, multiplicar, dividir, calcular, borrar, validar, mover, reemplazar, fijar, ordenar.

En muchas organizaciones, se llega a la conclusión de que entre 40 y 50 verbos son suficientes para describir cualquier política dentro de una especificación de proceso. Los objetos (el tema de las frases imperativas sencillas) deben consistir sólo en datos que se han definido en el diccionario o ser términos locales. Los términos locales, son aquellos que se definen explícitamente en una especificación de proceso individual; sólo son conocidos, relevantes y con significado dentro de dicha especificación de proceso (por ejemplo un cálculo intermedio).

En el siguiente ejemplo, se muestra una sencilla aplicación de especificación de proceso, que no considera datos definidos en el diccionario, simplemente quiere darse una aproximación a la manera de emplear el método:

1 - SI el monto de una factura multiplicado por el número de semanas de retraso en el pago supera los 10.000 dólares **ENTONCES:**

a. Proporcionar una fotocopia de la factura al encargado de ventas que llamará al cliente.
b. Anotar en el reverso de la factura que se le dio una copia al vendedor, junto con la fecha en

la que se hizo esto.
c. Volver a archivar la factura para estudiarla de nuevo dentro de dos semanas.

2 - EN CASO CONTRARIO, SI se han enviado más de cuatro recordatorios, **ENTONCES:**

a. Dar una copia de la factura al vendedor apropiado para que llame al cliente.
b. Registrar en el reverso de la factura que una copia ha sido enviada al vendedor, y la fecha en la que se hizo esto.
c. Volver a archivar la factura para reexaminarla dentro de una semana.

3 - EN CASO CONTRARIO (la situación aún no ha alcanzado proporciones serias):

a. Añadir 1 al contador de avisos de mora registrado en el inverso de la factura (si no se ha registrado tal contador, escribir: "*cuenta vencida de avisos de mora = 1"*).
b. **SI** la factura archivada es ilegible, **ENTONCES** hacer una nueva.
c. Enviar una copia de la factura al cliente, con el sello: *"n-ésimo aviso: pago de factura vencido, Favor de remitir inmediatamente*", donde n es el valor de avisos de mora.
d. Registrar en el reverso de la factura la fecha en la que se envió el n-ésimo aviso de mora.
e. Volver a archivar la factura para examinarla dentro de dos semanas.

4.7 Conclusión

Los sistemas siempre forman parte de sistemas mayores y siempre pueden dividirse en sistemas menores. Este punto es importante por dos razones:

- Sugiere una forma obvia de organizar un sistema computacional que se está estudiando para luego desarrollar, por el procedimiento de dividirlo en sistemas menores.

- Sugiere además que la definición del sistema que se está estudiando ha sido arbitraria. Se hubiera podido escoger un sistema ligeramente menor o mayor, dependiendo del lugar en el que se coloquen las fronteras y definan los alcances del mismo. Determinar lo que deberá abarcar un sistema y definirlo cuidadosamente para que todos los integrantes del equipo de desarrollo conozcan su contenido es una actividad crucial. Resulta una tarea muy difícil determinar drásticamente los límites. Tanto los usuarios como el equipo de trabajo, a menudo piensan que la frontera del sistema es fija e inmutable y que todo lo que se encuentre fuera, no merece la pena ser estudiado, aunque la realidad, muchas veces demuestra lo contrario. A la hora de definir ese límite o fronteras interviene siempre la naturaleza jerárquica de la complejidad.

Universidad Nacional del Litoral

FACULTAD DE INGENIERÍA
Y CIENCIAS HÍDRICAS

Ingeniería en Informática

Ingeniería de Software I

TEMA V – El modelo de conceptual

PRIMERA PARTE
EL MODELO CONCEPTUAL

Introducción

Gran parte de las labores efectuadas para un desarrollo informático, involucra el modelado del sistema que desea el usuario. En el tema anterior, se vio el modelado de las funciones y el análisis de los procesos del sistema, técnica que permite confeccionar diagramas que reflejan la esencia de cuáles son las tareas que hacen transformaciones sobre los datos en el contexto del sistema de información. En este tipo de modelo, el elemento fundamental es el constituido por los procesos, que mueven la información dentro del sistema a partir de los datos y estímulos recibidos desde fuera de las fronteras del mismo. Este tipo de metodología, permite tener una visión global bastante conveniente de los componentes funcionales del sistema, pero – como se indicara oportunamente –, sin dar detalles de ellos. Para mostrar los detalles, se recurría a dos herramientas textuales de apoyatura al modelado adicionales: el Diccionario de Datos y la Especificación de Procesos.

A pesar de que se utilicen estas herramientas de modelado adicionales y de que estos diagramas muestren la existencia de uno o más grupos de datos almacenados, deliberadamente transmiten muy poco acerca de sus detalles. Todos los sistemas almacenan y usan información acerca del ambiente en el cual interactúan; a veces, la información es mínima, pero en la mayoría de los sistemas actuales, es bastante compleja. No sólo se debe conocer en detalle qué información hay en cada almacén de datos, sino que también interesa conocer la relación que existe entre ellos. Este aspecto del sistema es tratado por el Modelo Conceptual de Datos.

5.1 Definiciones previas

5.1.1 Noción de entidad o sujeto

*Una **entidad** o sujeto es la representación de un objeto material o inmaterial con significado (cosa real o imaginaria) del universo exterior al que se le asocian atributos o propiedades que la caracterizan.*

Por ejemplo, al sujeto Cliente del universo exterior, le corresponderá en el sistema de información una entidad CLIENTE a la cual se le asociarán propiedades como Código de cliente, Apellido y Nombre, Domicilio, etc.

Gráficamente, se denotará mediante un rectángulo, indicándose en la parte superior del mismo, el nombre de la entidad y, separado por una línea, los otros objetos que la componen. Los nombres de las entidades deben ser expresados en singular.

5.1.2 Noción de relación

*Una **relación** es la consideración por el sistema de información del hecho de que existe una asociación entre objetos del universo exterior la cual tomará la forma de asociación entre las entidades correspondientes.*

Por ejemplo, entre la entidad **PEDIDO** (número, fecha), y la entidad **ARTÍCULO** (código, descripción, precio unitario), puede existir la relación ***FORMADO POR*** que expresa la asociación que existe entre el objeto **PEDIDO** y determinados objetos del tipo **ARTÍCULO**. La providencia colocada como nombre de la relación se corresponde a la construcción sintáctica que se desea lograr, y dependerá de quien es considerado como *sujeto* y quien como *objeto*. De esta manera, puede decirse que el **pedido CONTIENE artículo**, o bien **artículo *CONFORMA* pedido**. Su significación es la misma. Es común que se utilice como sujeto a la entidad de mayor jerarquía en la relación, siendo éstas normalmente las que representan hechos o eventos. En el ejemplo dado, la entidad **ARTÍCULO** no representa ningún hecho o evento, sino que es estructural. No ocurre lo mismo con la entidad **PEDIDO** que representa precisamente el hecho de haber recibido un requerimiento para adquirir o comprar artículos. De acuerdo a lo expresado, la forma más natural de armar la relación sería **PEDIDO *FORMADO POR* ARTÍCULO.**

A una relación se le pueden asignar propiedades, exactamente igual que a las entidades. La existencia de relaciones con propiedades es bastante común, sin embargo, algunos autores en sus metodologías, las consideran como entidades especiales a las que denominan **entidades débiles**. En el ejemplo considerado, la relación ***FORMADO POR*** puede ser portadora de la propiedad **cantidad** que expresa para cada **ARTÍCULO** del **PEDIDO** (contenido en el pedido), la cantidad solicitada.

La nomenclatura que inicialmente se denotará para indicar las relaciones, será una elipse con una línea horizontal en el eje mayor y se colocará en la parte superior el nombre de la relación, y en la parte inferior, los atributos o propiedades que la caracterizan.

5.1.3 Noción de propiedad

*El concepto de **propiedad** se corresponde a la noción del atributo que caracteriza a la entidad o a la relación.*

Un atributo es cualquier detalle que sirve para calificar, identificar, clasificar, cuantificar o expresar el estado de una entidad o una relación. En general, es cualquier descripción de una característica de importancia. Se corresponden con uno y solamente un tipo de dato, básicamente numérico, alfanumérico o tipo fecha.

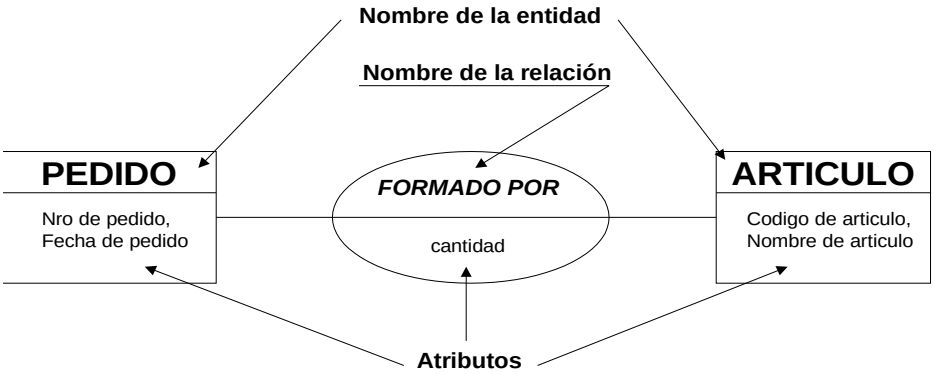
Los atributos o propiedades, pueden tener normas de validación y ciertas restricciones de formato, por ejemplo, el valor del atributo **importe de sueldo** no puede ser menor que cero, el valor del atributo **tipo de documento** puede ser 1, 2, 3, 4 ó 5 (según una norma de codificación) y no tomar ningún otro valor; el valor del atributo **fecha de entrega** de un pedido, no puede ser anterior al valor del atributo **fecha del pedido** original, etc.

5.1.4 Noción de dominio

*Un **dominio** es una abstracción que representa un conjunto de reglas de validación, restricciones en los formatos, valores posibles a tomar, tipo de dato y otras características propias de los atributos presentes en el modelo.*

Con la definición aportada, ahora un atributo, no necesariamente estará vinculado a un tipo de dato y además expresar sus limitaciones para solamente esa propiedad, sino que puede estar asociado a un **dominio**, y es éste el que tiene todas las características que deberá heredar el atributo. Por ejemplo si de define el dominio **Domicilio** y se define que será una cadena de caracteres no nula de ochenta símbolos de longitud, entonces por ejemplo **dirección postal de cliente**, **domicilio legal de cliente**, **domicilio comercial de cliente**, **domicilio de la empresa** etc. será conveniente y más sencillos que estén asociados al dominio y no tener que definir todas las limitantes para cada uno de los atributos. Además permite el mantenimiento de un formato estándar para todos los datos de características semejantes.

A manera de ejemplo, se graficará la entidad **PEDIDO** y la entidad **ARTÍCULO** vinculadas por la relación ***FORMADO POR***.



5.2 Clasificación de las entidades

5.2.1 Entidades permanentes

Las entidades permanentes son aquellas que se conservan constantemente en la base de información, pero que se pueden actualizar en cualquier momento. Corresponden a la estructura, **y** no representan hechos. Por ejemplo: **CLIENTE**.

Las propiedades de una entidad permanente pueden cambiar, pero la entidad en sí es estable. Las propiedades de una entidad permanente pueden ser de dos tipos:

- Propiedades de **filiación**:
  Son las que corresponden a su descripción. Por ejemplo: **Nombre del cliente** de una entidad **CLIENTE**.

- Propiedades de **situación:**

  Son las que expresan en que estado se encuentra la entidad permanente en un instante dado, situación que representa en general una acumulación o una posición. Por ejemplo, **stock disponible** de una entidad **ARTÍCULO**.

5.2.2 Entidades de tipo movimiento

Las entidades de tipo movimiento, son las que conservan hechos. Se trata de movimientos memorizados. Estas entidades son las imágenes de los eventos que portan estos movimientos. Están ligadas a los hechos circunstanciales. Una entidad de tipo movimiento, es el registro de un evento en el sistema de información.

Por ejemplo, para el caso de la entidad **PEDIDO** - memorizado en el sistema de información -, es el resultado del evento **PEDIDO *PASADO POR UN* CLIENTE**. Una particularidad que presentan, es que existe un momento en que ya no se puede actualizar una entidad tipo movimiento (no puede modificarse un pedido una vez que ya fue pasado).

5.3 Clasificación de las relaciones

5.3.1 Relaciones permanentes

Son relaciones entre entidades permanentes que se conservan constantemente y de las que se puede modificar sus propiedades en cualquier momento. Son relaciones estructurales. Por ejemplo dada la entidad **EMPLEADO** y la entidad **OFICINA**, la relación ***ASIGNADO A*** es de carácter permanente (es el vínculo lógico que deberá existir entre las oficinas y los empleados).

5.3.2 Relaciones de tipo movimiento

Son relaciones entre entidades permanentes o de tipo movimiento que representan el registro de un evento (relaciones coyunturales). Por ejemplo dada la entidad **PEDIDO** y la entidad **ARTÍCULO**, la relación ***FORMADO POR***, es de tipo movimiento.

5.4 Clasificación de las propiedades

La primera clasificación que establecemos es relativa a la naturaleza de la propiedad. Se dice que una propiedad es concatenada si es posible descomponerla o dividirla en otras propiedades. Una propiedad es elemental si no puede descomponerse (ha llegado al máximo grado de división y es una primitiva). Esta clasificación depende en gran medida del sistema que es objeto del estudio. Las propiedades que para uno son elementales, tal vez para otro no lo sean y admitan subdivisiones. El criterio para definir si la propiedad es elemental o no, debe ser definido por quien hace el estudio del sistema que establecerá si vale o no la pena separar o unificar conceptos.

Por ejemplo en el caso de indicar un domicilio, tal vez se coloque como que:

Dirección = Calle + Número + Piso + Departamento (caso de concatenación)

La propiedad **piso** o cualquiera de los otros términos involucrados, constituye una propiedad **elemental** (de acuerdo al criterio de quien ha indicado esto). Tal vez esto sea de utilidad pues permitirá realizar por ejemplo, una consulta de manera más sencilla que devuelva todos los datos de los clientes que viven en una calle determinada pero en un quinto piso. En este caso, la propiedad **dirección** es **concatenada**. Puede ocurrir también que el domicilio de un cliente simplemente sea considerado como un elemento más en cuyo caso no requeriría ser subdividido, razón por la cual domicilio ya es una propiedad elemental.

Otra forma de clasificar una propiedad, es por su grado de persistencia. Una propiedad cuyo valor no puede obtenerse a partir de ninguna deducción (dato primitivo), necesariamente debe ser **memorizada** en la base de información del sistema. Otras no hace falta que lo sean pues pueden obtenerse de manera indirecta y en estos casos se dice que son **deducibles** o **calculadas**.

La última forma de clasificar las propiedades, es relativa a su tipo. Se distinguen tres clases de propiedades:

- **Códigos:** Son informaciones sintéticas representativas de objetos materiales o inmateriales del universo exterior, según una ley de correspondencia rigurosa (sistema de codificación) por la que a todo objeto existente se le asocia un valor y sólo un valor del código de forma tal que a dos objetos diferentes les corresponden dos valores diferentes de código. Un código puede servir para identificar una entidad, es decir, de propiedad característica de esa entidad (clave de identificación).

- **Etiquetas:** Son datos alfanuméricos, cualitativos, simples cadenas de caracteres, que pueden ser suministrados por el sistema o sobre los que se pueden hacer clasificaciones o comparaciones, pero que no pueden participar en cálculos.

- **Cantidades:** Son datos numéricos, cuantitativos, que pueden participar en cálculos.

5.5 Tipos y ocurrencias

Un tipo, es un conjunto de elementos que tienen las mismas características (clase). Una ocurrencia de un tipo, es un elemento en particular perteneciente a dicho conjunto (objeto).

Entidad - Tipo:

Un tipo de entidad o entidad - tipo, es una clase de entidades particulares que tienen propiedades análogas.

Ocurrencia de Entidad - Tipo:

Una ocurrencia de entidad - tipo, es una entidad concreta perteneciente a este tipo.

Por ejemplo:

CLIENTE es una **entidad – tipo**.
El cliente **Juan Pérez** es una ocurrencia de esta entidad - tipo.

Un concepto especial de las entidades es el **identificativo o clave de identificación**, y lo que permite es distinguir una ocurrencia en particular, de cualquier otra perteneciente a la misma entidad – tipo. El identificativo es una propiedad o concatenación de propiedades que caracteriza cada ocurrencia de la entidad - tipo. Una entidad – tipo puede tener más de un medio alternativo de identificación única. En los gráficos, este concepto aparecerá subrayado o en negrita para diferenciarlo de los otros. De acuerdo al ejemplo anterior, se tiene que:

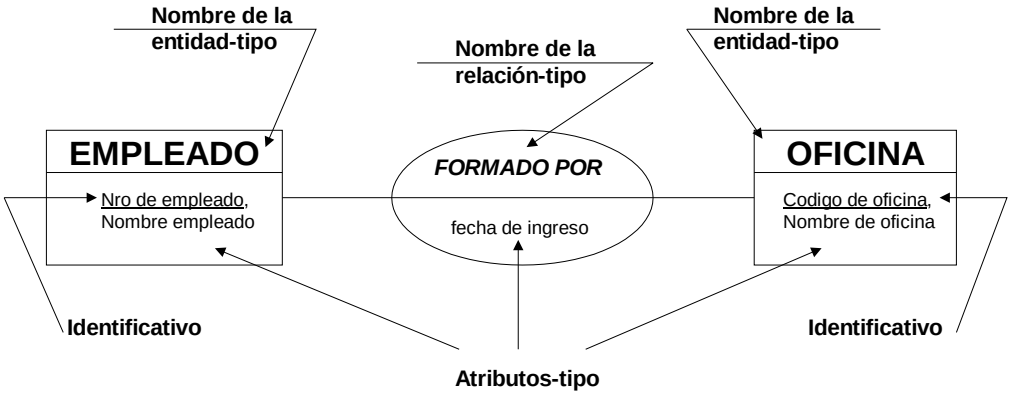
Relación-tipo:

Una relación-tipo, es una relación definida entre diversas entidades-tipo. Cada conjunto de ocurrencias de entidades que componen la relación-tipo, constituye una ocurrencia de la relación-tipo.

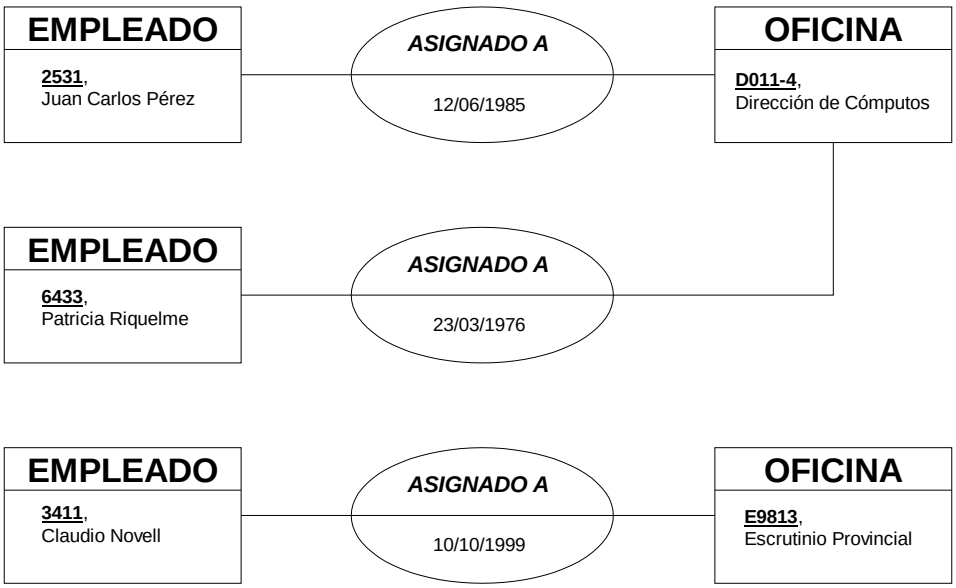
Propiedad-tipo:

Una propiedad-tipo, es una clase de propiedades semejantes. Una ocurrencia de una propiedad-tipo, es un valor tomado por esta propiedad.

Entidad - tipo, Relación - tipo y Atributos - tipo



Ocurrencias de Entidad - tipo, Relación - tipo y Atributos - tipo



En el ejemplo, se ve que los empleados **Pérez** y **Riquelme**, están asignados a la misma oficina (Dirección de Cómputos).

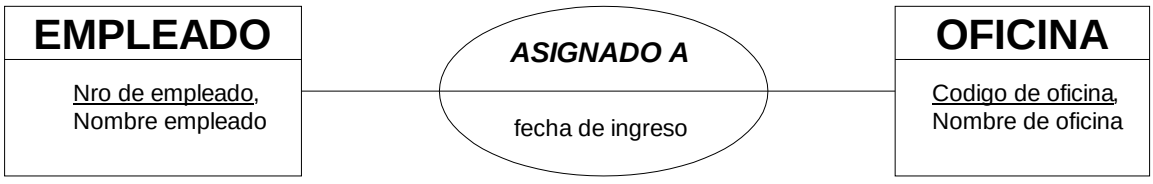
5.6 Características de una relación

5.6.1 Colección

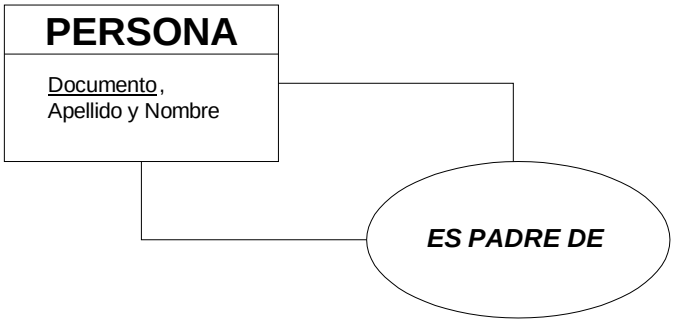
La colección de una relación-tipo, es la lista de entidades-tipo sobre las que se define la relación. En el ejemplo anterior, la colección de la relación ***ASIGNADO A*** se define sobre la colección (**EMPLEADO, OFICINA**).

5.6.2 Dimensión

La dimensión de una relación-tipo, es el número de ocurrencias de entidades alcanzadas por una única ocurrencia de la relación-tipo. Es superior o igual al número de entidades de la colección. Por ejemplo:



Colección: **EMPLEADO, OFICINA**
Dimensión: **2**



Colección: PERSONA
Dimensión: 2

Son necesarias dos ocurrencias de la relación **PERSONA** para una ocurrencia de la relación ***ES PADRE DE*** (se necesitan dos personas, una como padre y otra como hijo).

Una relación de dimensión 2, se la denomina relación **binaria.** Se hace esta acotación, ya que la mayoría de herramientas de diseño asistido CASE, trabajan con relaciones binarias. Otra característica que poseen, es que tales relaciones no pueden contener atributos, razón por la cual se deberán transformar en entidades débiles.

5.6.3 Funcionalidad

Se define la funcionalidad de una relación-tipo, como la correspondencia existente entre dos entidades-tipo **X e Y**, distinguiéndose las siguientes:

Uno a uno (1-1)

A cualquier ocurrencia de **X**, sólo le corresponde una ocurrencia de **Y** y recíprocamente.

Uno a muchos (1-n):

A toda ocurrencia de **X** corresponde 1 o varias ocurrencias de **Y**, y a toda ocurrencia de **Y**, corresponde una sola de **X**.

Muchos a muchos (m-n):

A toda ocurrencia de **X** corresponde una o varias ocurrencias de **Y** y recíprocamente.

5.6.4 Cardinalidad

La noción de cardinalidad permite expresar la totalidad o parcialidad de la participación de las entidades en la relación y su funcionalidad.

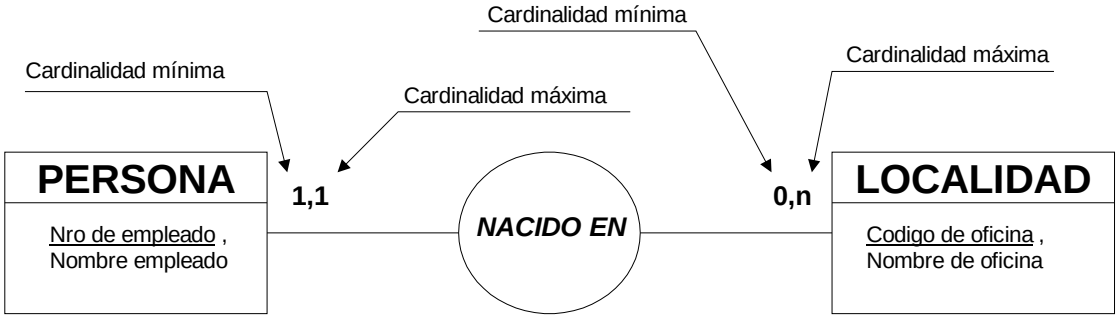
Cardinalidad Mínima:

La cardinalidad mínima de una relación, es el mínimo número de veces en la que cada ocurrencia de una entidad-tipo, participa en una relación-tipo. La cardinalidad mínima **0**, corresponde a una relación parcial. La cardinalidad mínima **1**, significa que no puede existir una ocurrencia de la entidad-tipo sin participar en una ocurrencia de la relación, vale decir que corresponde a una relación total.

Cardinalidad Máxima:

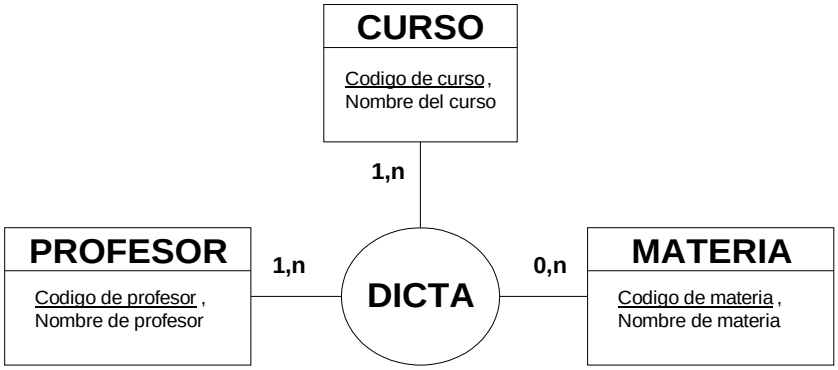
La cardinalidad máxima de una relación, es el máximo número de veces en la que cada ocurrencia de una entidad-tipo, participa en una ocurrencia de la relación-tipo. La cardinalidad máxima **1**, significa que cualquier ocurrencia de la entidad-tipo, no puede participar más que en una ocurrencia de la relación-tipo. La cardinalidad máxima **n**, significa que una ocurrencia de la entidad-tipo, puede estar implicada en un máximo de n ocurrencias de la relación.

Representación Gráfica



Representa que una **PERSONA**, ha nacido al menos en **1 LOCALIDAD** (cardinalidad mínima **1**) y a lo sumo en **1 LOCALIDAD** (cardinalidad máxima **1**), lo que es equivalente a decir que ha nacido en un único lugar. En una **LOCALIDAD** al menos pueden haber nacido **0 PERSONAS** (cardinalidad mínima **0** del subconjunto de personas considerado) y a lo sumo pueden haber nacido **n PERSONAS** (cardinalidad máxima **n**).

Ejemplo:



Un **PROFESOR** ***DICTA*** al menos una **MATERIA** y puede ***DICTAr*** varias. Una **MATERIA**, puede que no sea ***DICTAda*** y si se ***DICTA***, puede serlo muchas veces. En un **CURSO** se ***DICTA*** al menos una **MATERIA** con un **PROFESOR** pero pueden ***DICTA***rse varias **MATERIA**s siempre con un **PROFESOR**.

5.7 Reglas de gestión o reglas de negocio

Las reglas de gestión del modelo conceptual de datos, definen los condicionamientos que deben respetarse por el modelo. Por Ejemplo:

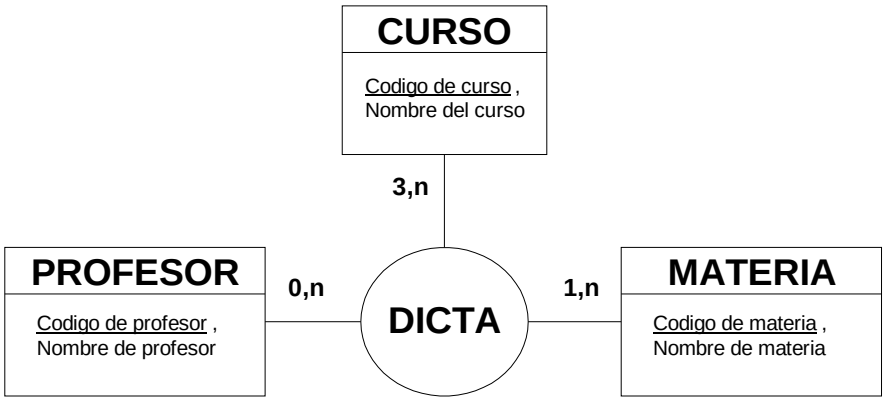
En el **MCD** de una escuela, las reglas de gestión pueden ser las siguientes:

Regla 1:
Todo **PROFESOR** ***DICTA*** en principio, al menos una **MATERIA**, pero algunos de ellos pueden tener licencia especial en razón de sus trabajos de investigación, vale decir que no ***DICTAN*** ninguna

Regla 2:
Toda **MATERIA**, se ***DICTA*** en al menos, un **CURSO**.

Regla 3:
En todo **CURSO**, se ***DICTAN*** al menos, tres **MATERIAS**.

El **MCD** sería entonces:



Es de destacar que la cardinalidad mínima **3** en curso, en realidad y desde el punto de vista de la funcionalidad de las relaciones, no tiene importancia. Simplemente es un detalle que podrá tener alguna regla de negocios posterior. En el modelo conceptual de datos, bastaría con indicar **1**, ya que precisamente lo que trata de recalcarse precisamente es si la participación de la entidad en la relación es parcial o

total (**0** ó **1**).

Las reglas de gestión, expresan los CONDICIONAMIENTOS DE INTEGRIDAD del modelo. Estos condicionamientos de integridad, representan las leyes del universo real que se modela del sistema de información. Se distinguen dos tipos de condicionamientos:

a. Condicionamientos estáticos

Se pueden referir a:

- **Una propiedad:** su formato, lista de posibles valores, entorno o rango de valores admisibles, etc.
- **Diversas propiedades de una misma relación o entidad**. Por ejemplo que la *fecha de pedido* sea menor o igual que la *fecha de entrega* y siendo éstas propiedades de la entidad **PEDIDO**.
- **Propiedades de ocurrencias distintas de una relación o entidad**. Por ejemplo que, en un histórico de precios de artículos, la *fechaInicio* del precio actual, debe ser mayor que la *fechaInicio* del último precio anterior.
- **Propiedades de entidades/relaciones diferentes.** Por ejemplo que la suma de la cantidad de productos vendidos y entregados a los clientes, debe ser igual a la suma de los productos que faltan en el depósito.
- **Cardinalidades**
- **Dependencias funcionales**

b. Condicionamientos dinámicos:

Los condicionamientos dinámicos, expresan las reglas de evolución y afectan directamente el paso del sistema de información de un estado a otro. Por ejemplo que se registre que el sueldo de un empleado en un período no puede ser inferior al sueldo del período del ejercicio anterior.

5.8 Dependencias funcionales

5.8.1 Dependencias funcionales entre propiedades

5.8.1.1 Dependencia funcional

Se dice que dos propiedades **a** y **b**, están ligadas por una dependencia funcional y se denota:

$$\mathbf{a} \longrightarrow \mathbf{df} \longrightarrow \mathbf{b}$$

si el conocimiento del valor de **a**, determina uno y sólo un valor de **b**.

p.e.:

$$\textit{Codigo\_cliente} \longrightarrow \mathbf{df} \longrightarrow \textit{Nombre\_cliente}$$

El conocimiento de *Codigo_cliente*, determina uno y sólo un *nombre_cliente.* En otras palabras, si se conoce el código del cliente, se debe poder conocer su nombre que será único.

Lo recíproco es falso. El nombre del cliente no permite conocer su código, pues varios clientes pueden tener el mismo nombre.

No necesariamente una dependencia funcional se referirá a un atributo elemental. La dependencia funcional, puede afectar a la concatenación de varias propiedades.

p.e.:

$$\textit{Nro\_pedido} + \textit{Codigo\_de\_articulo} \longrightarrow \mathbf{df} \longrightarrow \textit{Cantidad}$$

La sola referencia del *Codigo_de_articulo*, determinará solamente de qué artículo se trata pero no basta para determinar la cantidad pedida, ya que un mismo producto puede presentarse en varias boletas de pedido distintas. En consecuencia – si se admite que un artículo puede figurar a lo sumo una sola vez en una boleta de pedido – para poder determinar unívocamente el valor de una *Cantidad*, se deberá conocer el ***Nro_pedido + Codigo_de_articulo***.

5.8.1.2 Dependencia funcional elemental

Se dice que existe una ***dependencia funcional elemental*** entre **a** y **b**, y se indica como:

$$\mathbf{a} \longrightarrow \mathbf{b}$$

SI

$$\mathbf{a} \longrightarrow \mathbf{df} \longrightarrow \mathbf{b}$$

donde $\mathbf{a} = \mathbf{a}_1 + \mathbf{a}_2 + \ldots\ldots + \mathbf{a}_n$
tal que $\mathbf{a}_i$ es una propiedad elemental

y no existe ningún $\mathbf{a}_i$ que por sí solo, pueda determinar el valor de **b**.

p.e.: sea

Codigo_cliente + *Nombre_cliente* $\longrightarrow$ **df** $\longrightarrow$ *Direccion_cliente* **no es elemental**,

Donde **a** = *Codigo_cliente* + *Nombre_cliente*
entonces $\mathbf{a}_1$ = *Codigo_cliente*

$$\mathbf{a}_2 = Nombre\_cliente$$

pero para determinar la *Direccion_cliente*, basta solamente con *Codigo_cliente* ($\mathbf{a}_1$) (es más, ocurre que *Nombre_cliente* es también determinado por *Codigo_cliente*). Entonces se deduce que la expresión **1**, no constituye una *dependencia funcional elemental*, simplemente es una *dependencia funcional*.

En cambio
Codigo_cliente ⟶ **df** ⟶ *Direccion_cliente*

es una *dependencia funcional elemental* y se puede escribir:

Codigo_cliente ⟶ *Direccion_cliente*

Es de notar que una dependencia funcional elemental, únicamente es cuestionable si el valor de **a** es una concatenación de atributos elementales. En caso contrario, si ya es una dependencia funcional, ésta será elemental.

Para el ejemplo expuesto anteriormente:

Nro_pedido + *Codigo_de_articulo* ⟶ **df** ⟶ *Cantidad*

puede verse que **a** es una concatenación de atributos elementales (puede que sea o no una dependencia funcional elemental), y que el *Nro_pedido* **no** puede determinar el valor de *Cantidad* por sí solo, y el valor de *Codigo_de_articulo* tampoco puede determinar el valor de *Cantidad* por sí solo; pero la concatenación de ambos sí lo determina unívocamente, por lo tanto constituye una ***dependencia funcional elemental***.

Nro_pedido+*Codigo_de_articulo* ⟶ *Cantidad*
(dependencia funcional elemental)

5.8.1.3 Dependencia funcional elemental directa

Se dice que la propiedad **b** depende funcionalmente de **a** mediante una ***dependencia funcional elemental directa***, si:

1. Esta dependencia es elemental:
   a ⟶ **b**
2. ***No existe*** una propiedad **c**, tal que:
   a ⟶ **df** ⟶ **c** y **c** ⟶ **df** ⟶ **b**

Dicho de otra forma, se elimina toda transitividad.

p.e.:
un **PROFESOR** (*Codigo_profesor*, *Nombre_profesor*), dicta una y solamente una
MATERIA (*Codigo_materia*, *Nombre_materia*). La siguiente proposición:

Codigo_profesor ⟶ *Nombre_materia* (**1**)

Es una *dependencia funcional elemental* debido a que un profesor dicta una y solamente una materia, entonces el conocimiento del identificativo de profesor (*Codigo_profesor*), determinará el nombre de la materia (*Nombre_materia*) que dicta. También son válidas las siguientes proposiciones:

Codigo_profesor ⟶ *Codigo_materia* (**2**)

Codigo_materia ⟶ *Nombre_materia* (**3**)

Nótese que (**1**) **NO** es una ***dependencia funcional elemental directa***, ya que las dependencias funcionales (**2**) y (**3**) determinan la existencia de una dependencia transitiva. (**2**) y (**3**) son ***dependencias funcionales elementales directas***.

5.8.1.4 Clave de identificación de una entidad

Se conoce como clave de negocios o identificativo de una entidad, a una propiedad (o concatenación de propiedades) que pertenecen a esa entidad, tal que todas las demás propiedades, dependen de ella funcionalmente y de forma tal que no sea verdadera para ninguna de sus partes (dependencia funcional elemental).

Por ejemplo, sea la entidad **CLIENTE** con sus atributos (a, b, …., n):

CLIENTE (*Cod_cliente, Nombre_cliente, Direccion_cliente***)**

Cod_cliente + Nombre_cliente **no es una clave a pesar que**

Cod_cliente + *Nombre_cliente* ⟶ **df** ⟶ *Direccion_cliente*

ya que permanece verdadero para la parte *Cod_cliente* de la concatenación *Cod_cliente* **+** *Nombre_cliente*, ya que *Cod_cliente* determina perfectamente *Direccion_cliente*.

En cambio ***Cod_cliente*** es una clave, pues:

Cod_cliente ⟶ *Nombre_cliente*
Cod_cliente ⟶ *Direccion_cliente*

Se debe tener en cuenta que una entidad puede tener **varias claves**. Se debe prever y especificar claramente en el modelo, cuál de las claves actuará como identificativo. Así, si la entidad **EMPLEADO**, tiene las propiedades *Codigo_empleado* y *CUIL* que son clave, habrá que elegir cual de las dos se desempeñará como identificativo.

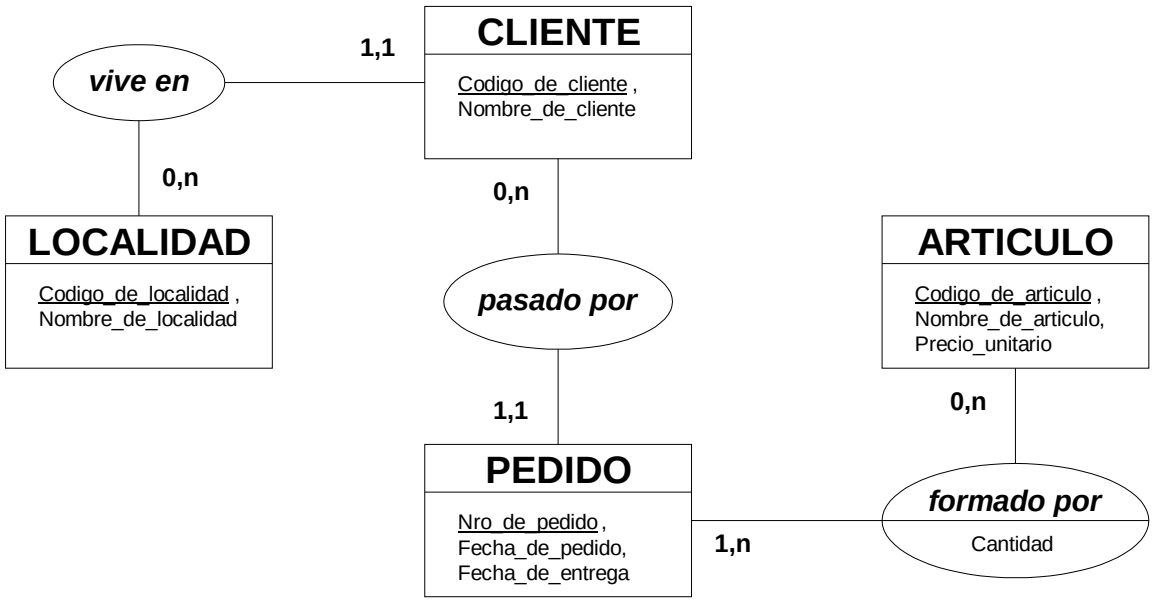
Obsérvese además que la dependencia funcional por la cual una propiedad depende de la clave, no es necesariamente elemental. Todo lo mencionado respecto a dependencias funcionales y clave de identificación de una entidad puede verse reflejado

en el siguiente ejemplo de aplicación.

Sean las siguientes reglas de gestión:

1. Un **CLIENTE** reside en una única **LOCALIDAD**.
2. Una boleta de **PEDIDO** corresponde a uno y solamente un **CLIENTE**.
3. Una boleta de **PEDIDO** estará conformada por varios **ARTÍCULOS** conociéndose la *Cantidad* de cada uno de ellos.
4. Un **ARTÍCULO** puede estar presente en varios **PEDIDOS**, pero no podrá estar repetido en el mismo.

El correspondiente modelo será de la forma:



En la entidad **ARTÍCULO**, se tiene la dependencia funcional:

$$Codigo\_de\_articulo \longrightarrow Nombre\_de\_Articulo$$

Por la relación ***formado por***, se tiene la dependencia:

$$Nro\_de\_pedido + Codigo\_de\_Articulo \longrightarrow Cantidad$$

Lo que significa que el conocimiento del *Nro_de_pedido* y del *Codigo_de_Articulo* determina la cantidad pedida.

Generalidades sobre los identificativos o clave de negocios

- Todas las ocurrencias de una entidad - tipo, deben ser distintas.

- Toda entidad - tipo, debe contar al menos, con una clave de negocios candidata.
- Una de las claves de negocio candidatas deberá ser elegida como identificativo principal de la entidad – tipo.
- El identificativo principal debe ser la mínima de las claves de negocio candidatas. Ninguna de sus partes puede a su vez, ser una clave candidata.
- Una clave de negocio candidata que no es el identificativo es una clave de negocios alternativa.

5.8.2 Dependencias funcionales entre entidades

Las dependencias funcionales entre propiedades, hay que considerarlas respecto a las entidades y relaciones.

Se dice que existe una dependencia funcional entre dos entidades **A** y **B** y se indica como:

A ⟶ **B**

si toda ocurrencia de **A**, determina una y sólo una ocurrencia de **B**.

En el ejemplo anterior, se tiene (a través de la relación ***pasado por***).

PEDIDO ⟶ **CLIENTE**

ya que el solo conocimiento del **PEDIDO** permite determinar unívocamente a qué **CLIENTE** pertenece.

Las cardinalidades **1,1** de **PEDIDO** en esta relación, expresan que todo cualquier **PEDIDO**, determina uno y sólo un cliente. Se concluye entonces que

LAS CARDINALIDADES 1,1 CORRESPONDEN SIEMPRE A UNA DEPENDENCIA FUNCIONAL

Hay que considerar las dependencias funcionales entre entidades a través de las relaciones entre esas entidades. Es posible asimilar las dependencias funcionales entre entidades a las dependencias funcionales entre los identificativos de estas entidades. Entonces con respecto a lo declarado anteriormente

PEDIDO ⟶ **CLIENTE**

puede asimilarse a:

Nro_de_pedido ⟶ *Codigo_de_cliente*

Otra ocurrencia referida al ejemplo puede ser:

CLIENTE ⟶ **LOCALIDAD**
Codigo_de_cliente ⟶ *Codigo_de_localidad*

5.8.3 Propiedades de las dependencias funcionales

Reflexividad:

a ⟶ *df* ⟶ ***a***

Proyección:

si ***a*** ⟶ *df* ⟶ ***b + c*** entonces
a ⟶ *df* ⟶ ***b*** y ***a*** ⟶ *df* ⟶ ***c***

Ejemplo: *Codigo_cliente* ⟶ *Nombre_cliente+Domicilio*
entonces
Codigo_cliente ⟶ *Nombre_cliente* y
Codigo_cliente ⟶ *Domicilio*

Ampliación:

si ***a*** ⟶ *df* ⟶ ***b*** y
a ⟶ *df* ⟶ ***c*** entonces
a + b ⟶ *df* ⟶ ***c***

Ejemplo: *Codigo_cliente* ⟶ *Nombre_cliente* y
Codigo_cliente ⟶ *Domicilio*
entonces
Codigo_cliente + Nombre_cliente ⟶ *Domicilio*

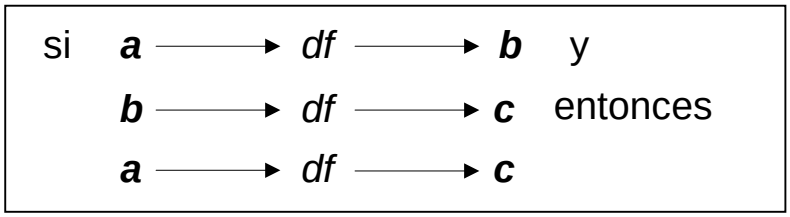
Aditividad:

si ***a*** ⟶ *df* ⟶ ***b*** y
a ⟶ *df* ⟶ ***c*** entonces
a ⟶ *df* ⟶ ***b + c***

Ejemplo: *Codigo_cliente* ⟶ *Nombre_cliente* y
Codigo_cliente ⟶ *Domicilio*
entonces

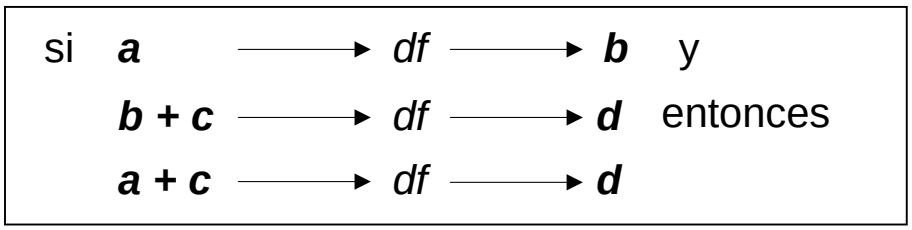
Codigo_cliente ⟶ *Nombre_cliente+Domicilio*

Transitividad:



Ejemplo: *Codigo_cliente* ⟶ *Codigo_postal* y
Codigo_postal ⟶ *Nombre_localidad*
entonces
Codigo_cliente ⟶ *Nombre_localidad*

Pseudo-Transitividad:



Ejemplo: *Codigo_cliente* ⟶ *Codigo_postal* y
Codigo_postal + Nombre_cliente ⟶ *Nombre_localidad*
entonces
Codigo_cliente + Nombre_cliente ⟶ *Nombre_localidad*

SEGUNDA PARTE
LA NORMALIZACIÓN DEL MODELO CONCEPTUAL

5.9 Introducción

En la primera parte del tema se han presentado las definiciones primitivas para la modelación conceptual de datos. En ésta, a partir de todas esas primitivas enunciadas, se desarrollarán metodologías que permitirán construir modelos conceptuales de manera consistente y con la eliminación total de redundancias. Éste último es el objetivo principal de **La Normalización**. *La normalización, es un proceso mental abstracto totalmente natural y obvio.* Consiste solamente en aplicar el sentido común y realizar las cosas de manera obvia.

5.10 La Normalización

La normalización, es una técnica, basada en reglas bien definidas que permite que todas las entidades y relaciones presentes en el sistema, tengan las redundancias eliminadas.

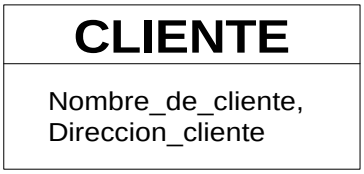
Las entidades del modelo conceptual de datos, deben verificar todas las reglas de normalización que serán desarrolladas a continuación.

La normalización, se realiza utilizando descomposición **sin pérdidas**. Ello implica que a partir de un conjunto de datos agrupados originalmente de una manera determinada, se realice la descomposición en varios subconjuntos sin perder el sentido que presentaba la información original. Los procesos que llevan a realizar la descomposición, se organizan por niveles de complejidad, yendo de lo más sencillo y general a lo de mayor detalle y particular. Tales niveles son los que conforman las denominadas Formas Normales.

5.10.1 Primera forma normal (1FN)

En una entidad, todas las propiedades son elementales, y existe al menos una clave de negocios o identificativo definido. Cada atributo deberá tener un único valor para cualquier ocurrencia de la entidad en un momento cualquiera.

Por ejemplo, sea la entidad **CLIENTE**:

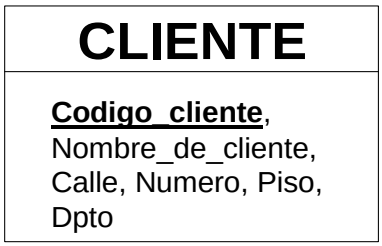


Esta entidad, no está en **1FN**, pues, no existe clave (varios clientes pueden tener el mismo nombre). Por otra parte, si *Dirección_cliente* es:

Direccion_cliente = Calle + Numero + Piso + Departamento

no constituye una propiedad elemental, entonces por más que la entidad tenga una clave de negocios, no estaría en **1FN** por la propiedad concatenada utilizada.

La clave, si es única, se tomará como identificativo. Si existen varias claves, se elegirá una de ellas. Toda entidad debe tener un identificativo. Para que esté en 1FN, debería ser:

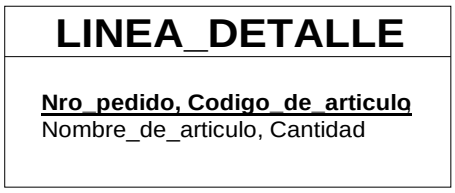


Nótese que la clave o identificativo sugerido, se destaca marcándose en letra negrita o subrayado (en el ejemplo, de ambas formas).

5.10.2 Segunda forma normal (2FN)

*Para que una entidad esté en segunda forma normal, deberá estar en **1FN** y además, toda propiedad de la entidad deberá depender de la clave mediante una dependencia funcional elemental. Dicho de otra manera, toda propiedad de la entidad deberá depender del identificativo completo (caso de tratarse de una concatenación de atributos) y no solamente de una parte de él.*

Obsérvese que, de acuerdo a la definición, el caso del ejemplo anterior de la entidad **CLIENTE**, que se encuentra en **1FN**, además está en **2FN**, ya que su clave de identificación no está conformada por una concatenación de atributos, en consecuencia todos los atributos dependen de la clave a través de una *dependencia funcional elemental*. Como regla práctica, se deberá poner en tela de juicio todas aquellas entidades en las que la clave esté conformada por dos o más atributos. Por ejemplo, sea la entidad **LINEA_DETALLE** de un **PEDIDO**:



El ejemplo presentado está en **1FN** ya que posee clave de identificación y además todos los atributos son elementales. Como la clave está conformada por la concatenación de dos atributos, cabe cuestionarse si está o no en **2FN**. De acuerdo al gráfico, tal clave de identificación está dada por *Nro_pedido + Codigo_de_Articulo*, pero la dependencia funcional:

Nro_pedido + Codigo_de_Articulo ⟶ **df** ⟶ *Nombre_de_articulo*

no es elemental, puesto que:

Codigo_de_Articulo ⟶ **df** ⟶ *Nombre_de_articulo*

En consecuencia, esta entidad no está en **2FN**. Para llevarla a tal estado, el MCD debería ser:

5.10.3 Tercera forma normal (3FN)

Para que una entidad esté en tercera forma normal, deberá estar en 2FN y además, todas las propiedades deberán depender de la clave mediante una dependencia funcional elemental directa.

Por ejemplo, sea la entidad **CLIENTE**:

Esta entidad está en **2FN** ya que la clave no es una concatenación de atributos y todas las propiedades dependen funcionalmente de esa clave (consecuentemente estará en **1FN**), pero no está en **3FN** ya que la dependencia funcional:

Codigo_cliente ⟶ **df** ⟶ *Nombre_de_localidad*

no es directa a causa de la transitividad

Codigo_cliente ⟶ **df** ⟶ *Codigo_de_localidad* ⟶ **df** ⟶ *Nombre_de_localidad*

El MCD debería ser:

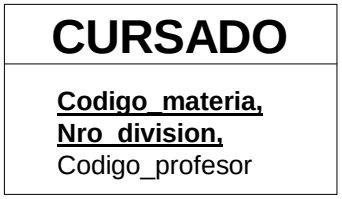
5.10.4 Forma normal de BOYCE-CODD (BCFN)

Si una entidad tiene un identificativo concatenado, ninguno de los elementos que componen este identificativo debe depender de alguna propiedad.

La **1FN, 2FN y 3FN**, se ocupan de restricciones que afectan sólo a las propiedades **no claves**. No obstante, es común referirse a la **BCFN**, como **3FN**.

Si se admiten las reglas de gestión:

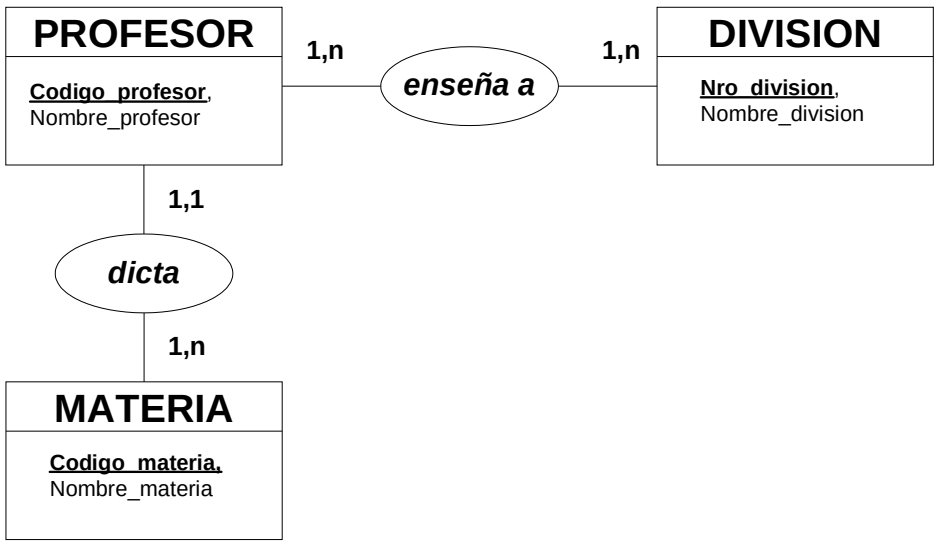
- Una **MATERIA** puede ser impartida por varios **PROFESOR**es pero en una **DIVISIÓN**, es dictada por un único profesor.
- Existen varias divisiones identificadas por número de división.



Para el caso planteado, la entidad del gráfico está normalizada ya que el profesor depende de la materia y del curso en donde la dictará. No puede determinarse de ninguna otra manera. Además ninguna de las partes de la clave determina a ningún otro atributo. Si se agrega la siguiente restricción:

- Cada **MATERIA** es impartida por un único **PROFESOR**.

no está en **BCFN**, pues, la clave de la entidad, dada por *Codigo_materia + Nro_division*, depende de la propiedad *Codigo_profesor*. Concretamente se establece la dependencia *Codigo_profesor* → *Codigo_materia*. El MCD debería ser entonces:



Las normalizaciones anteriores, tienen por objeto eliminar las redundancias (no es necesario repetir la descripción de un producto pedido, cada vez que se solicite dicho producto) y eliminar las anomalías de actualización (si se elimina un empleado, se deseará sin duda conservar la oficina a la que estaba asignado).

5.11 Cumplimiento de las condiciones de integridad – Las Relaciones

Para el desarrollo de todas las condiciones de integridad, se tomará en consideración un ejemplo de aplicación que tendrá las siguientes características:

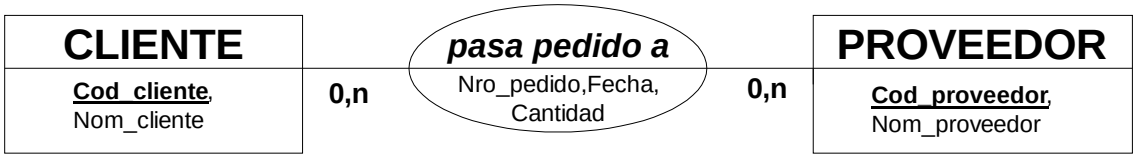
1. Un **CLIENTE** puede realizar sus **PEDIDOs** de **ARTÍCULOs** a distintos **PROVEEDORes** cada uno en una *fecha* determinada y con un *número de pedido*.

2. El **CLIENTE** al pasar el pedido al **PROVEEDOR** especifica a través del **PEDIDO** la *cantidad* de **ARTÍCULOs** que solicita.

5.11.1 Verificación

*En toda ocurrencia de entidad-tipo o de relación-tipo, no debe existir más que un único valor de cada propiedad (no repetitiva). Para las entidades esta regla procede de la **1FN**. Tal regla, deberá permanecer verdadera para las relaciones.*

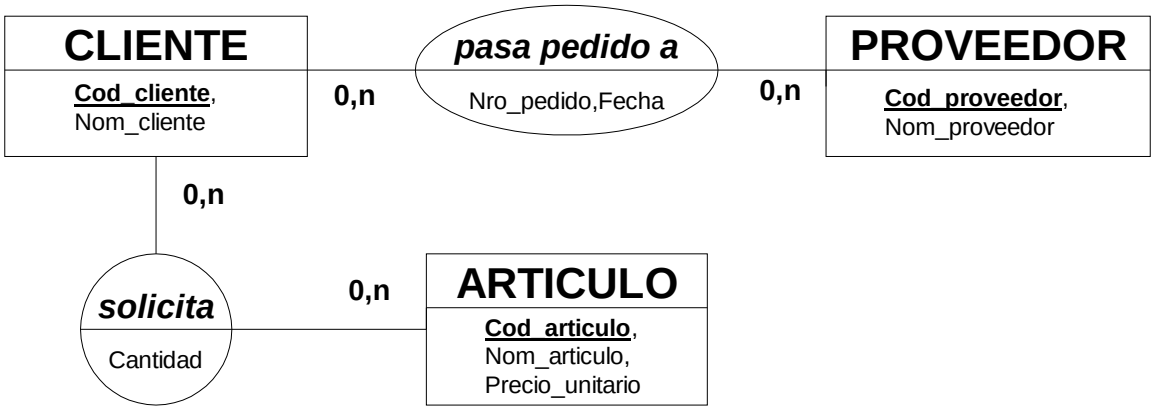
De acuerdo a la descripción anterior, considérese el siguiente MCD:



Nótese que la relación ***pasa pedido a***, no está verificada, ya que puede haber varios valores de *Cantidad* en un pedido realizado por un cliente a un proveedor (una por cada artículo solicitado). Entonces, el valor de *Cantidad*, no depende sólo del cliente y del proveedor, sino también del artículo solicitado. En otras palabras:

En una relación, las propiedades deben depender funcionalmente de las entidades que son colección de la relación, o lo que es lo mismo, de la concatenación de sus identificativos. En consecuencia, la clave de una relación es la concatenación de las claves de las entidades que son colección de ella.

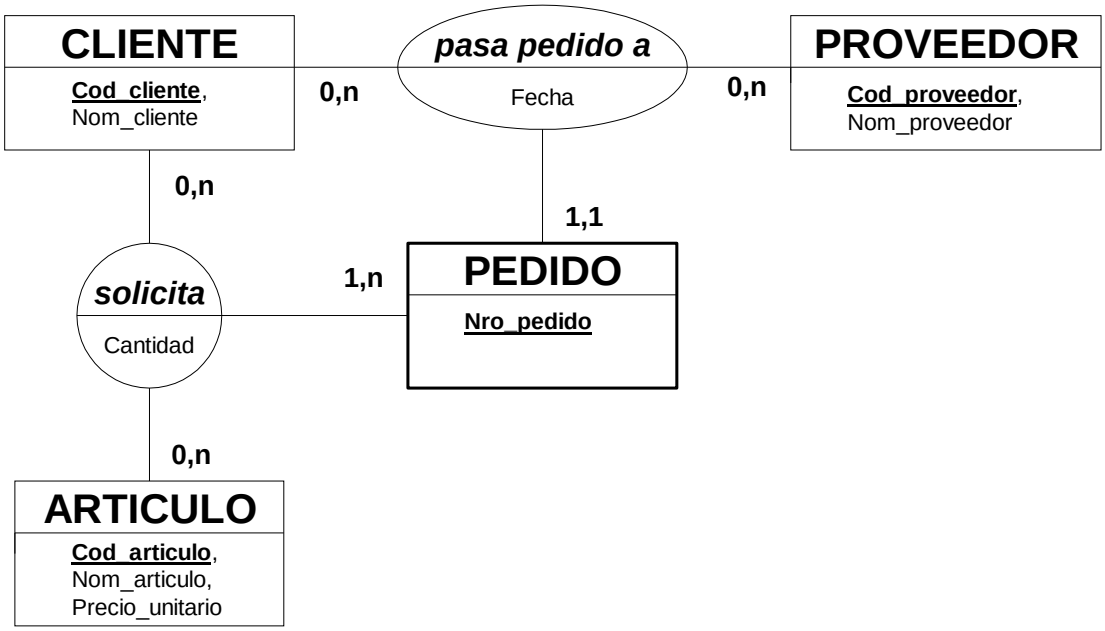
El siguiente MCD, mejora al anteriormente expuesto:



Nótese que ahora que la propiedad *Cantidad*, que figuraba en la relación ***pasa pedido a***, desaparece es ésta pues es inherente a los artículos pedidos (relación ***solicita***).

No obstante, en la relación ***solicita***, la *Cantidad*, no depende solamente del cliente y del artículo, sino también del Número de pedido, ya que un cliente puede realizar varios pedidos del mismo artículo. El Número de pedido (*Nro_pedido*), no puede deducirse aunque se conozca el CLIENTE, el ARTÍCULO solicitado y el PROVEEDOR al que fue pasado tal pedido, ya que puede haber varias boletas de pedido para un cliente dado, un representante dado y un artículo dado. En consecuencia, no se cumple la regla de verificación.

En este caso, se hace necesario crear la entidad-tipo **PEDIDO**. El MCD final quedará entonces de la manera que se representa seguidamente.



En el modelo, nótese ahora que una única ocurrencia de la propiedad *Cantidad*, depende del *Nro_pedido*, del *Cod_Articulo* y del *Cod_cliente*. Esta concatenación la determina perfectamente concluyendo que el modelo está VERIFICADO.

5.11.2 Normalización de las relaciones

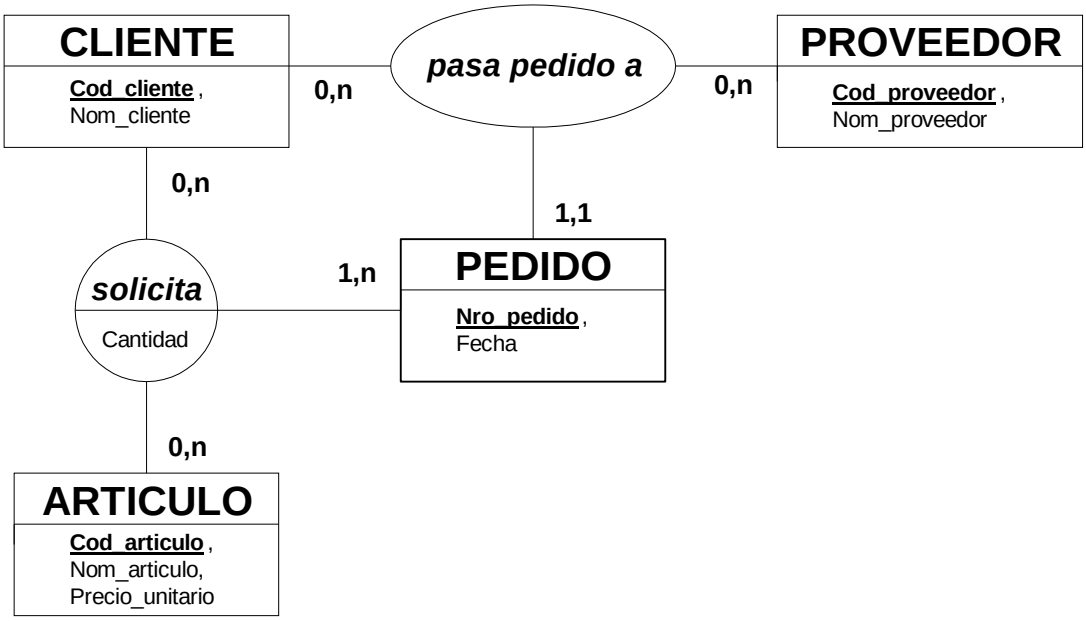
Cada propiedad de la relación, debe depender funcionalmente de la concatenación del conjunto de identificativos de las entidades colección de la relación, y no de ningún subconjunto de ellos. Esto implica que estos atributos deben tener una dependencia elemental respecto del conjunto.

Entonces, deberá haber una dependencia total de las propiedades de la relación con respecto a las entidades.

Considerando el gráfico anterior, se tiene que la propiedad *fecha* (en la relación ***pasa pedido a***), depende de ***Nro_pedido*** + ***Cod_cliente*** + ***Cod_proveedor***; y que la propiedad *Cantidad* (en la relación ***solicita***) depende de ***Cod_cliente + Nro_pedido + Cod_Articulo***.

Es de notar que la fecha en que se realiza un pedido, solamente depende del pedido, sin importar a que cliente corresponde o a qué proveedor es pasado, en consecuencia, tal dependencia es solamente de un subconjunto de la concatenación de los identificativos mencionados, no existiendo dependencia plena de las entidades (o de sus identificativos) **CLIENTE** o **PROVEEDOR** que participan en la relación ***pasa pedido a*** por lo que la relación no está normalizada. Para normalizarla, la propiedad *Fecha,* debe desaparecer de la relación ***pasa pedido a*** e incluirse en la entidad **PEDIDO**.

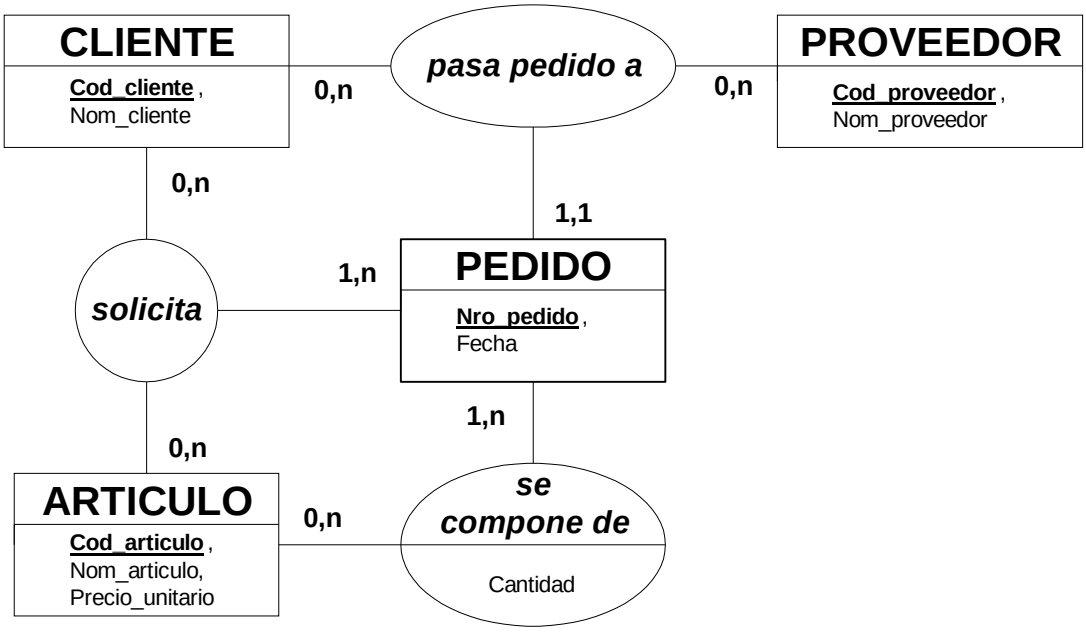
El MCD quedará:



Pero, la *Cantidad* pedida puede conocerse si se sabe al ***Nro_pedido*** y el ***Cod_Articulo*** de donde se concluye que la relación ***solicita*** no está normalizada, ya que involucra además a la entidad **CLIENTE** (***Cod_cliente***) sin depender de ésta. Se tiene entonces que:

Nro_pedido* + *Cod_Articulo ⟶ *Cantidad*

En este caso, la propiedad *Cantidad*, no puede migrarse a una entidad como se hizo con el atributo *Fecha*, entonces *Cantidad*, debe estar afectada a una relación distinta, no existente hasta el momento. Se hace necesario crear la relación-tipo ***se compone de*** en la que sólo intervendrán las entidades **PEDIDO** y **ARTÍCULO**. El MCD propuesto queda entonces:



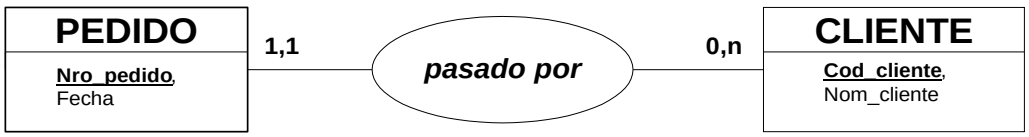
5.11.3 Descomposición de relaciones

*Consiste en reemplazar una relación de dimensión **n**, en varias relaciones de dimensiones más pequeñas, utilizando las dependencias funcionales que se pueden detectar en la relación.*

Del ejemplo, una relación pasible a ser descompuesta es ***solicita***. En la relación ***pasa pedido a*** del MCD anterior, existe la dependencia funcional:

PEDIDO ⟶ **CLIENTE**

Se puede, por consiguiente, descomponer esta relación en dos:



Y en

De lo hecho, la relación ***pasado por***, ya existía en la relación ***pasa pedido a*** (la dependencia **PEDIDO** ⟶ **CLIENTE** procede de la relación ***pasa pedido a***). La descomposición no es posible más que con dos condiciones**:**

1. *La cardinalidad mínima de las entidades de la izquierda en la dependencia funcional, debe ser 1 en la relación a descomponer (relación total para estas entidades).*

2. *Si la dependencia funcional procede de otra relación diferente a la que se desea descomponer, es necesario que se refiera a las mismas ocurrencias de entidades que la relación a descomponer.*

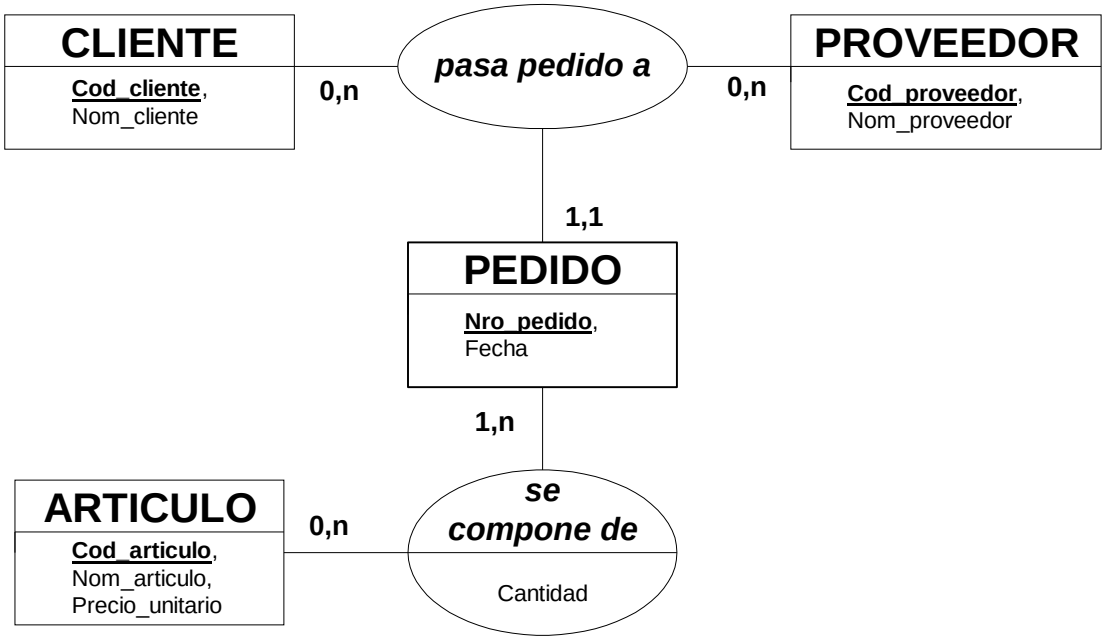
En el ejemplo, la dependencia funcional que ha permitido la descomposición es:

PEDIDO ⟶ **CLIENTE**

1. A través de esta dependencia, aplicada a la relación ***pasado por***, determina una cardinalidad mínima de **PEDIDO** es **1** (y máxima de **1** por ser una dependencia funcional). Aplicando la misma dependencia sobre la relación ***solicita***, la cardinalidad mínima de **PEDIDO** en esta relación es **1**, y en consecuencia, se asegura el cumplimiento de la regla 1.

2. La relación ***pasa pedido a***, y la relación a descomponer ***solicita***, ponen en juego las mismas ocurrencias de **CLIENTE** y de **PEDIDO**, pues son los propios clientes los que efectúan los pedidos y los que piden los artículos, y por consiguiente los artículos pedidos por los clientes corresponden a los mismos pedidos que los pedidos efectuados por los clientes. Se cumple la regla 2.

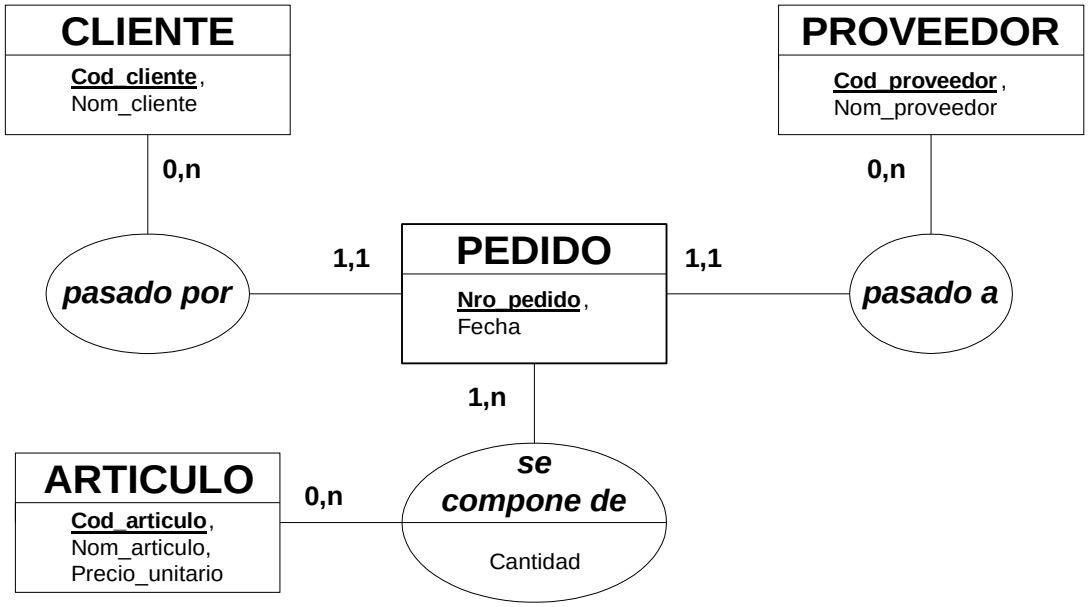
En el MCD la relación ***solicita***, se puede descomponer, por consiguiente, en ***pasa pedido a*** (que ya existía) ya que un **PEDIDO** se refiere a uno y solamente un **CLIENTE**, y en una nueva relación ***pide producto*** que vincularía al **CLIENTE** con los **ARTÍCULOS** solicitados. Como los artículos solicitados están vinculados con **PEDIDO** a través de la relación ***se compone de***, entonces puede suprimirse el doble empleo de la misma relación. Consecuentemente, la relación ***solicita*** desaparece, quedando el modelo:



De igual forma, las dependencias funcionales (debidas a las cardinalidades 1,1 de **PEDIDO** en ***pasa pedido a***):

PEDIDO ⟶ **CLIENTE** y
PEDIDO ⟶ **PROVEEDOR**

permiten descomponer ***pasa pedido a*** en dos relaciones binarias ***pasado por*** (entre **CLIENTE** y **PEDIDO**) y ***pasado a*** (entre **PROVEEDOR** y **PEDIDO**). De estas descomposiciones, se obtiene el siguiente MCD:



5.12 Aplicación de formas normales

La normalización, no es una actividad aplicable solamente a las metodologías que apuntan al diseño de bases de datos. Ésta, es necesaria siempre que se trabaje con datos (atributos) que deban ser organizados de cierta manera (entidades y relaciones) y que permitan mantener la integridad total respecto a la situación de partida. La forma en que serán posteriormente utilizadas estas agrupaciones definidas, podría ser manual, o automatizada y dentro de ésta, mediante sistemas de archivos convencionales o bien sobre bases de datos. Los ejemplos desarrollados a continuación, permiten apreciar la aplicación de las distintas instancias de evolución a través de las formas normales, logrando de esta manera reunir toda la información de partida con las redundancias eliminadas. Se tomarán para ello, atributos agrupados en forma de tablas del tipo planilla de electrónica. Demás está agregar que estos ejemplos, no apuntan a ninguna forma de utilización de datos bajo un lenguaje o base de datos específico.

Reglas de gestión:

1. La organización objeto de estudio, cuenta con varios departamentos en que está subdivida.
2. Todos los empleados de la organización, deben estar asignados a uno y sólo un departamento.
3. Cada departamento tiene un sólo jefe.
4. Existen varios proyectos de distinto tipo.
5. Cada empleado, tiene asignado uno o varios proyectos distintos, y una cantidad de horas asignadas a cada uno.

La primera tabla que se planteará, contendrá toda la información volcada. En ella puede verse la estructura (columnas que la componen) y las ocurrencias presentadas (filas).

EMPLEADO		DEPARTAMENTO				PROYECTO			
		Descripción		Jefe					
Código	Nombre	Código	Nombre	Código	Nombre	Código	Nombre	Inicio	Horas
COEMP	*NOEMP*	*CODEP*	*NODEP*	*COJEF*	*NOJEF*	*COPRO*	*NOPRO*	*FEIN*	*HS*
908	SÁNCHEZ	10	VENTAS	988	PÉREZ	10 30 40	FACTIBILIDAD DESARROLLO MERCADO	12/10/99 05/01/00 02/02/00	150 260 350
562	GÓMEZ	20	MARKETING	387	LOYARTE	20 50	ANÁLISIS SOFTWARE	20/12/99 23/11/99	400 600
988	PÉREZ	10	VENTAS	988	PEREZ	10 20 40	FACTIBILIDAD ANÁLISIS MERCADO	03/11/99 11/12/99 28/01/00	230 450 480
921	MÉNDEZ	15	INSUMOS	919	ROLDAN	20	ANÁLISIS	15/02/00	630

Una forma directa de construir la tabla, es generar el llamado registro compuesto. El registro compuesto, tiene toda la información con respecto a cada conjunto existente de formas de relación que afecten a todas las variables. La cuestión es hacer esto de modo de obtener un agrupamiento significativo. Para lograr un registro compuesto, se necesita una forma de relación compuesta como la propuesta. La tabla propuesta, tiene toda la información que se puede necesitar para muchas aplicaciones. Y éste es justamente el problema. MUCHA INFORMACIÓN ES REPETITIVA. Por ejemplo, el nombre del proyecto, está repetido varias veces; el nombre del departamento también entre otras cosas y puede llevar a la inconsistencia de la información repetida, además de ocupar espacio innecesario.

Partiendo de un registro compuesto, se puede ahorrar mucho espacio conservando los datos sobre empleados, proyectos y departamentos en conjuntos separados. Si se procede de este modo, es necesario encontrar una forma de recolectar los datos más tarde para obtener el registro compuesto o un registro menor si es todo lo que se necesita para una aplicación en particular. A esto apunta la Normalización.

1FN

Mediante la aplicación de la teoría especificada para la primera forma normal y considerando la entidad **EMPLEADO** en relación al proyecto asignado que tiene, se obtendrían las siguientes tablas:

EMPLEADO

COEMP
NOEMP
CODEP
NODEP
COJEF
NOJEF

COEMP	NOEMP	CODEP	NODEP	COJEF	NOJEF
908	SÁNCHEZ	10	VENTAS	988	PÉREZ
562	GÓMEZ	20	MARKETING	387	LOYARTE
988	PÉREZ	10	VENTAS	988	PÉREZ
921	MÉNDEZ	15	INSUMOS	919	ROLDÁN
919	ROLDÁN	15	INSUMOS	919	ROLDÁN

En donde la clave es el código de empleado **COEMP**.

ASIGNACION

COEMP COPRO
NOPRO
FEIN
HS

COEMP	COPRO	NOPRO	FEIN	HS
908	10	FACTIBILIDAD	12/10/99	150
908	30	DESARROLLO	05/01/00	260
908	40	MERCADO	02/02/00	350
562	20	ANÁLISIS	20/12/99	400
562	50	SOFTWARE	23/11/99	600
988	10	FACTIBILIDAD	03/11/99	230
988	20	ANÁLISIS	11/12/99	450
988	40	MERCADO	28/01/00	480
921	20	ANÁLISIS	15/02/00	630

En donde la clave es la concatenación de las propiedades código de empleado y código de proyecto: **COEMP + COPRO**.

2FN

De acuerdo a la segunda forma normal, en la tabla de asignaciones, el nombre del proyecto (**NOPRO**), depende solamente del código de proyecto (**COPRO**) y no de toda la clave concatenada, por lo que se para llevarla a 2FN se debe descompone:

ASIGNACION

COEMP COPRO
FEIN
HS

COEMP	COPRO	FEIN	HS
908	10	12/10/99	150
908	30	05/01/99	260
908	40	02/02/00	350
562	20	20/12/99	400
562	50	23/11/99	600
988	10	03/11/99	230
988	20	11/12/99	450
988	40	28/01/00	480
921	20	15/02/00	630

Consecuentemente, la tabla proyecto, quedará de la forma:

PROYECTO

COPRO
NOPRO

COPRO	NOPRO
10	FACTIBILIDAD
20	ANÁLISIS
30	DESARROLLO
40	MERCADO
50	SOFTWARE

3FN

Según los principios de la tercera forma normal, en la tabla **EMPLEADO**, existen dos dependencias funcionales no directas. En uno de los casos:

CODEP ⟶ **NODEP**

que se resuelve descomponiendo. En el otro:

COJEF ⟶ **NOJEF**

Puede ser absorbida en la misma tabla, pues un jefe también es un empleado. Entonces:

EMPLEADO

COEMP
NOEMP
CODEP
COJEF

COEMP	NOEMP	CODEP	COJEF
908	SÁNCHEZ	10	988
562	GÓMEZ	20	387
988	PÉREZ	10	988
921	MÉNDEZ	15	908
919	ROLDÁN	15	919

DEPARTAMENTO

CODEP
NODEP

CODEP	NODEP
10	VENTAS
15	INSUMOS
20	MARKETING

BCFN (Forma normal de Boyce Codd)

Para ejemplificar esta forma normal, se recurrirá a la siguiente tabla:

DEPARTAMENTO	PROYECTO	RESPONSABLE	HORAS-HOMBRE
10	P1	362	1.000
10	P3	486	500
15	P1	298	2.500
15	P2	320	1.700
15	P3	486	800

En la que la clave es una concatenación de las propiedades **DEPARTAMENTO** y **PROYECTO**.

Si se agregan las siguientes reglas de gestión:

1. Ningún empleado, puede ser responsable de más de un proyecto y todos los proyectos tienen al menos un responsable

2. Cada departamento no tiene más que un responsable por proyecto

La tabla, no está en **BCFN**, pues la propiedad **PROYECTO** de la clave concatenada, depende de la propiedad no clave **RESPONSABLE**. Se obtiene entonces la **BCFN** descomponiendo:

DEPARTAMENTO	RESPONSABLE	HORAS-HOMBRE
10	382	1.000
10	486	500
15	298	2.500
15	320	1.700
15	486	800

RESPONSABLE	PROYECTO
382	P1
486	P3
298	P1
320	P2

En donde la clave es **DEPARTAMENTO + RESPONSABLE**, y **RESPONSABLE** respectivamente.

5.13 Formas normales de orden superior

Ciertos autores, consideran la existencia de otras formas normales que son casos muy particulares. Si se trabaja ordenadamente, normalizando entidades (hasta 3FN), verificando, normalizando y descomponiendo relaciones, no será necesario plantearse si se cumple o no con la 4FN o 5FN. Se verá además que si bien siempre se nombran las entidades, la 4FN y la 5FN hacen referencia a la forma en que se pueden o no descomponer relaciones.

5.13.1 Cuarta forma normal (4FN)

Para estar en **4FN**, la entidad no debe contener dependencias de valores múltiples. Por ejemplo:

Sean las entidades: **EMPLEADO, LENGUAJE** y **HARDWARE**
Sean las relaciones:

EMPLEADO conoce LENGUAJE
EMPLEADO conoce HARDWARE

y considérese que un empleado, puede ser diestro en uno o más tipos de lenguajes y hardware, y estos conocimientos son independientes entre sí. Tómese como ejemplo la ocurrencia del empleado **328** que posee conocimientos de:

LENGUAJE:	1 – PASCAL	**HARDWARE**:	100 – AS400
	2 – COBOL		200 – VAX
	3 – BASIC		300 – PC

La representación de esta información podría realizarse de las siguientes formas:

DISYUNTO

Nº Empleado	Lenguaje	Hardware
328	1	--
328	2	--
328	3	--
328	--	100
328	--	200
328	--	300

PRODUCTO

Nº Empleado	Lenguaje	Hardware
328	1	100
328	1	200
328	1	300
328	2	100
328	2	200
328	2	300
328	3	100
328	3	200
328	3	300

COMPRIMIDO

Nº Empleado	Lenguaje	Hardware
328	1	100
328	2	200
328	3	300

Otra manera de resolver el problema, es a través de la creación de un atributo no existente hasta el momento que indique el tipo de conocimiento.

Nº Empleado	Tipo	Conocimiento
328	L	1
328	L	2
328	L	3
328	H	100
328	H	200
328	H	300

En los ejemplos planteados, la clave de la entidad, será la concatenación de **EMPLEADO**, **LENGUAJE** y **HARDWARE**, y para el último caso, será **EMPLEADO**, ***Tipo***, ***Conocimiento*** **(LENGUAJE ó HARDWARE)**. Nótese que para el último caso, el atributo conocimiento tiene dos significados distintos, dependiendo éste del valor que toma otro atributo (*tipo*).

Si el conocimiento que se pretende representar, no está relacionado (el del software con el del hardware), para normalizar se impone la descomposición.

Nº Empleado	Lenguaje
328	1
328	2
328	3

Nº Empleado	Hardware
328	100
328	200
328	300

En donde la clave, estará dada para cada uno por **EMPLEADO**, **LENGUAJE** y **EMPLEADO**, **HARDWARE** respectivamente.

Pero, si se desea mantener reflejado el conocimiento de un empleado sobre un determinado software que corre sobre determinado hardware, la descomposición anterior no permite reorganizar la información inicial ya que existe una dependencia de valor múltiple (no está en 4FN). Refiriéndose al ejemplo anterior, si el empleado 328 conoce el lenguaje 1 que corre sobre el hardware 200, y el lenguaje 3 que corre también sobre el lenguaje 200, resulta claro que la descomposición anterior no permite mostrar tal situación, es por ello que en estos casos, para llegar a la 4FN se hace necesario:

Nº Empleado	Lenguaje	Hardware
328	1	200
328	3	200

En donde la clave estará dada por la concatenación de **EMPLEADO, LENGUAJE** y **HARDWARE** sin poderlo descomponer, ya que está en 4FN.

5.13.2 Quinta forma normal (5FN)

Valores múltiplemente dependientes, no admiten descomposición. Por ejemplo:

- **EMPRESA *fabrica* PRODUCTO**
- **AGENTE *representa* EMPRESAS**
- **AGENTE *venden determinados productos de* EMPRESA**

Considérese el siguiente ejemplo:

Agente	Empresa	Producto
Sánchez	FORD	Auto
Sánchez	FIAT	Camión
García	FORD	Camión

Si se intenta descomponer esta tabla de relación, quedará**:**

Agente	Empresa
Sánchez	FORD
Sánchez	FIAT
García	FORD

Agente	Producto
Sánchez	Auto
Sánchez	Camión
García	Camión

Empresa	Producto
FORD	Auto
FIAT	Camión
FORD	Camión

Combinando nuevamente para obtener la información original, quedará:

Fila incorrectamente creada →

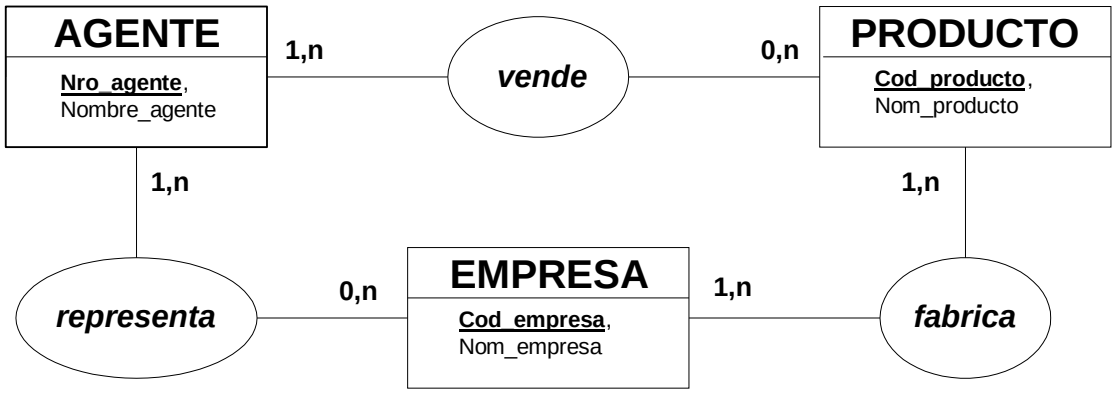
Agente	Empresa	Producto
Sánchez	FORD	Auto
Sánchez	**FORD**	**Camión**
Sánchez	FIAT	Camión
García	FORD	Camión
García	FORD	Auto

Se concluye entonces, que para descomponer, debe respetarse la restricción SIMÉTRICA:

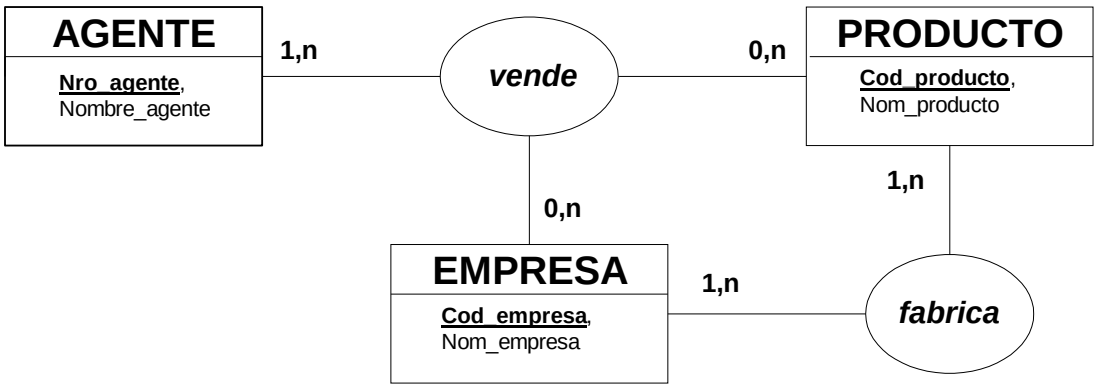
el AGENTE **A** VENDE EL PRODUCTO **P**
y el AGENTE **A** REPRESENTA A LA EMPRESA **E**
y la EMPRESA **E** PRODUCE EL PRODUCTO **P**

restricción:

el AGENTE **A** VENDE EL PRODUCTO **P** PARA LA EMPRESA **E**



Nótese que en el MCD graficado, no hay manera de poder determinar los productos de qué empresa son los que vende el agente. Precisamente el problema radica en la existencia de una dependencia de valores múltiples. Consecuentemente, éste MCD no está en 5FN. El MCD quedará de la siguiente forma:



El nudo del problema se centra en la relación ***VENDE*** que es donde la dependencia múltiple no admite descomposición (nuevamente se ve que la forma normal se refiere a una relación). Concluyendo, no es posible descomponer una relación cuya DIMENSIÓN ES MAYOR O IGUAL QUE TRES.

5.14 Construcción de modelos conceptuales de datos

Se explicará el método con la ayuda de un ejemplo. El sistema de información contiene esencialmente las propiedades que figuran en las boletas de pedido. La apariencia del formulario de pedido es del tipo:

MCD Intermediarios					
			Boleta de pedido Nº **0000-0000125683**		
			Fecha: **…. / …. / 20…**		
	Cliente:	….. - ……..……......			
F	***Domicilio:***	…… Nº …..	***Localidad:***	……. - ……	
I					
C	***Proveedor:***	….. - ……..……......			
H	***Domicilio:***	…… Nº …..	***Localidad:***	……. - ……	
unl					

Cód.	***Descripción***	***Cantidad***	***Precio Unitario***	***Importe***
			TOTAL	

Recopilación de la información

Después de la recopilación de información relativa al sistema de información existente, se reúnen todos los documentos utilizados, así como las descripciones de los diversos ficheros actualmente en uso. El siguiente paso, consistirá en la determinación y especificación clara de

las reglas de gestión:

Regla 1: Un cliente puede tener ninguna o muchas boletas de pedido.
Regla 2: Una boleta corresponde a un único cliente.
Regla 3: Un pedido tiene uno o varios artículos. Al menos tendrá uno.
Regla 4: Un artículo solamente puede estar una vez en un pedido.
Regla 5: Una boleta de pedido se pasa a solamente un proveedor (que no siempre es el mismo para un cliente dado).

En caso de tratarse de un sistema manual, no existirá todavía una gran utilización de codificaciones. Se supondrá en estos casos que existen códigos para identificar las entidades evidentes. Por ejemplo el Código de Cliente para **CLIENTE**, y el Código de Proveedor para **PROVEEDOR**, el Código de Localidad para **LOCALIDAD**.

Construcción del diccionario de datos

A partir de todos los ítemes de datos que se encuentran, se construye el diccionario de datos, en el que se deberá consignar:

- **Nombre del atributo**: es la etiqueta con la que se identificará a la propiedad representada.
- **Significado**: descripción relativa al uso del atributo.
- **Tipo**: clasificación del ítem de dato según sea numérico, carácter o fecha (en el marco de un compromiso asumido al trabajar con un lenguaje de programación que use sistemas de archivos o de una base de datos en particular, aparecerán nuevos tipos de datos). Los tipos de datos que tengan las mismas características tales como limitantes y se considere que pertenecen a una misma clase, pueden conformar tipos de datos de usuario conocidos además como dominios.
- **Longitud**: es la cantidad de caracteres o espacios que ocupa.
- **Naturaleza**: permite indicar en principio si el atributo es elemental, concatenado o calculado. En caso de ser concatenado, debe especificarse cómo está conformado, y si es calculado, especificar la forma de obtenerlo. Por otra parte, permite indicar si se trata de un atributo de tipo movimiento, filiación o situación.
- **Regla de cálculo**: permite indicar las validaciones que tiene cada atributo, valores admisibles, lista de valores posibles, condiciones de integridad, forma de calcularlo, formato, etc.

Seguidamente se presenta el diccionario de datos que surge del análisis del formulario de boleta de pedido.

Nombre	Significado	Tipo	Long	Naturaleza		Regla de Cálculo
Nro_pedido	Es la numeración que permite identificar una boleta de pedido.	N	10	E	M	Entero no nulo y mayor que cero. Formato: ##########
Fecha_pedido	Es la fecha en que se confecciona un pedido.	F	10	E	M	No nulo. Formato dd/mm/aaaa
Cod_cliente	Es la clave de identificación de un cliente.	N	5	E	FI	***A crear. Entero no nulo y mayor que cero.***
Nom_cliente	Es el nombre o denominación de un cliente.	A	30	E	FI	No nulo.
Domi_cliente	Es el domicilio de residencia de un cliente.	A	45	CO	FI	Calle_cliente + Nro_domicli
Calle_cliente	Es la calle del domicilio de residencia de un cliente.	A	35	E	FI	
Nro_domicli	Es el número del domicilio de residencia de un cliente.	A	10	E	FI	
Cod_locali_cliente	Es el código postal de la localidad de residencia de un cliente.	N	5	E	FI	Entero no nulo y mayor que cero.
Nom_locali_cli	Es el nombre de la localidad de residencia de un cliente.	A	30	E	FI	No nulo.
Cod_provee	Es el código de identificación de un proveedor.	N	5	E	FI	***A crear. Entero no nulo y mayor que cero.***
Nom_provee	Es el nombre o denominación de un proveedor.	A	30	E	FI	No nulo.
Domi_provee	Es el domicilio de residencia de un proveedor.	A	45	CO	FI	Calle_provee+ Nro_domiprovee
Calle_provee	Es la calle del domicilio de residencia de un proveedor.	A	35	E	FI	
Nro_domiprovee	Es el número del domicilio de residencia de un proveedor.	A	10	E	FI	
Cod_locali_provee	Es el código postal de la localidad de residencia de un proveedor.	N	5	E	FI	Entero no nulo y mayor que cero.
Nom_locali_provee	Es el nombre de la localidad de residencia de un proveedor.	A	30	E	FI	No nulo.
Cod_Articulo	Es el código de identificación de un artículo.	A	5	E	FI	No nulo. Formato: un caracter + 4 dígitos
Nom_Articulo	Es el nombre o denominación de un artículo.	A	30	E	FI	No nulo.
Cantidad	Es la cantidad de artículos solicitada en una línea de la boleta de pedido.	N	5	E	M	Entero no nulo y mayor que cero.
Precio_unitario	Es el precio unitario de venta de un artículo.	N	10,2	E	FI	No nulo y mayor que cero. Formato: ocho dígitos para la parte entera y dos para la decimal.
Importe_linea	Es el costo de un artículo con relación a la cantidad solicitada que figura en cada línea de detalle de la boleta de pedido.	N	10,2	CA	M	***Precio_unitario** * **Cantidad***. No nulo y mayor que cero. Formato: ocho dígitos para la parte entera y dos para la decimal.

Nombre	Significado	Tipo	Long	Naturaleza		Regla de Cálculo
Total_pedido	Es el importe total que figura en la boleta de pedido.	N	10,2	CA	M	Sumatoria de ***Importe_linea*** No nulo y mayor que cero. Formato: ocho dígitos para la parte entera y dos para la decimal.

Referencias

Tipo:
A – Alfabético
F – Fecha
N – Numérico

Naturaleza:
E – Elemental
CO – Concatenado
CA – Calculado
FI – Filiación
SI – Situación
M – Movimiento

Si una propiedad se emplea en diferentes utilizaciones, habrá que considerar en principio, que se trata de propiedades distintas. Una vez establecidas las entidades corresponderá determinar si se deben reunir en una sola o tienen significado distinto. En el diccionario, figura el código postal y en nombre de localidad en dos oportunidades, una haciendo referencia al cliente y la otra al proveedor.

Depuración del diccionario de datos

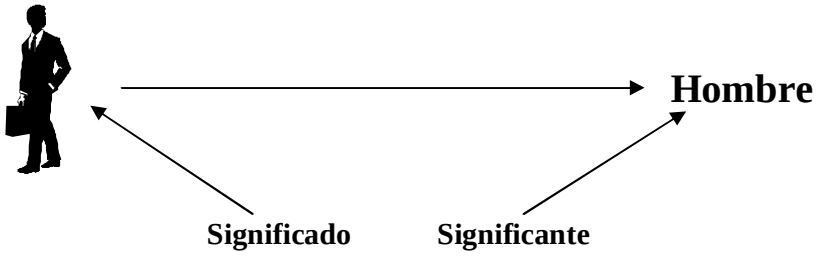
Eliminar los atributos calculados que se puede determinar a partir de otros mediante alguna regla de transformación explicitada, como ***Importe_linea*** **y** ***Total_pedido*** ya que el primero surge del producto entre la ***cantidad * precio_unitario*** y el segundo de la sumatoria de éstos últimos.

Eliminar los atributos concatenados como ***Domi_cliente*** **y** ***Domi_provee*** ya que se utilizan en su lugar los que se especificaron como elementales.

La confusión en la definición de los datos, puede surgir de la relación que se establece entre los significados y sus significantes.

El significado es la conceptualización que se tiene del objeto abstracto o concreto que se quiere representar o cualificar.

El significante es la palabra o signo empleado para representar un significado.



Los sinónimos, son dos significantes distintos para un mismo significado. Por ejemplo, las propiedades *Dirección_cliente* y *Domicilio_cliente* representan el mismo significado pero tienen dos nombres de atributos (significantes) distintos.

La polisemia, es un mismo significante para dos significados distintos (homónimos). Por ejemplo, si se tiene la propiedad *Dirección* que se refiere a la dirección del cliente dentro de la entidad cliente, a la dirección del proveedor dentro de la entidad proveedor, etc.

Para depurar la lista de datos, deben en primer lugar ser eliminados los sinónimos y las polisemias. Puede ocurrir que cuando se comienza a trabajar, no se detecten diferencias y algunos atributos estén mal definidos. Conforme se avanza en el diseño del modelo, tales diferencias se ponen de manifiesto rápidamente.

En el diccionario de datos, se dará un nombre para todos y cada uno de los tipos de datos y se eliminarán los sinónimos y polisemias. Para el caso del ejemplo, el atributo ***Cod_locali_cliente*** es sinónimo de ***Cod_locali_provee*** y consecuentemente ***Nom_locali_cli*** es sinónimo de ***Nom_locali_provee***; por hacer referencia simplemente a **LOCALIDAD**. Para obtener un nombre más acorde, estos grupos serán reemplazados como ***Cod_locali*** y ***Nom_locali*** siendo las características, las mismas a las ya definidas.

Determinación de las entidades y las relaciones

Una manera natural y semi intuitiva, basada en el conocimiento de la realidad que se modela, puede ordenarse según los siguientes pasos:

1. Se buscan las propiedades de la lista que puedan ser identificativos de las entidades. Se realizan los muestreos necesarios hasta obtener las entidades que van a configurar el MCD.

2. Se describen esas entidades, asignando las propiedades que forman el resto de la descripción de las mismas.

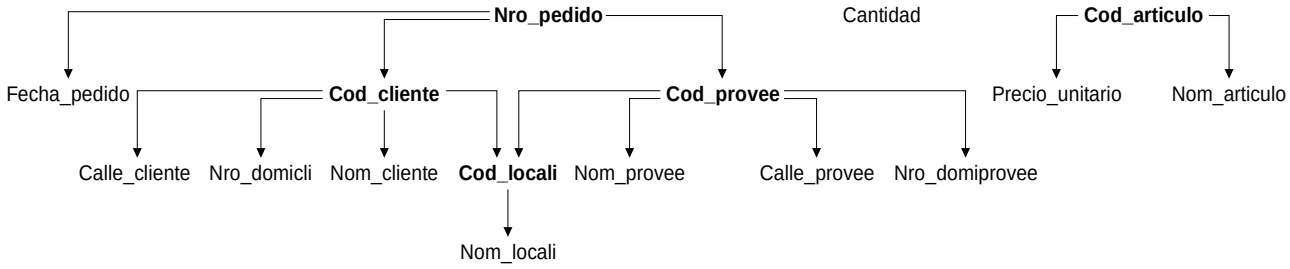
3. Las propiedades que queden en la lista, y que no se han atribuido a ninguna entidad, pertenecerán a relaciones (con su correspondiente colección).

El proceso descrito, se puede formalizar a través de lo que se conoce como grafo de dependencias funcionales, o su correspondiente matriz.

Grafo de dependencias funcionales

Se extrae del diccionario de datos la lista de las propiedades que no están ni concatenadas ni son calculadas. En el ejemplo propuesto, se rechazan las propiedades: ***Domi_cliente***, ***Domi_provee*** (*concatenadas*), ***Importe_linea*** y ***Total_pedido*** (*calculadas*).

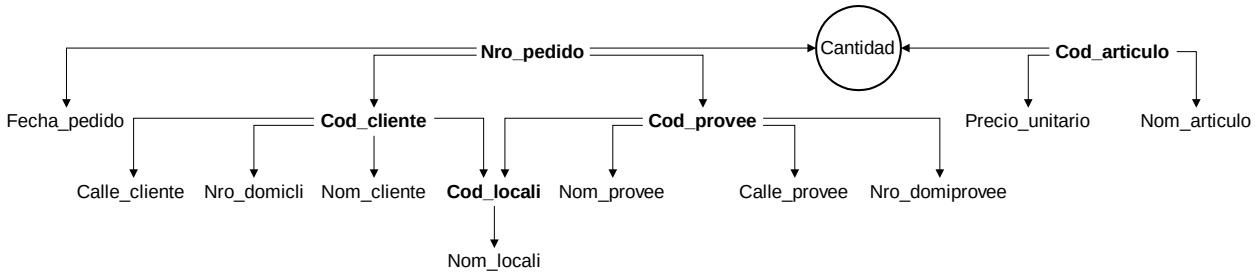
Se establece luego la Lista de Dependencias Funcionales cuyo dominio de partida no contiene más que una sola propiedad no concatenada a partir del examen de los documentos y de los identificativos propuestos. Esta lista de **df** se puede visualizar mediante un grafo como el siguiente:



Nótese que la propiedad ***Cantidad*** ha quedado aislada.

Si quedan propiedades aisladas, se buscan dependencias funcionales que conduzcan a estas propiedades a partir de la concatenación de propiedades. Si no se encuentra ninguna, tal propiedad se continúa dejando aislada. En este caso, se utiliza la dependencia:

Nro_pedido + Cod_articulo ⟶ ***Cantidad***



Se intentará asegurar, siempre, que las propiedades aisladas no se correspondan a entidades aisladas (casos de tablas propias del sistema de aplicación), para que las que habría que suponer una clave de identificación que permita añadir las dependencias que faltan. Por ejemplo, si existiese la propiedad ***Cod_articulo***, se estaría entonces en presencia de dos propiedades aisladas: ***Precio_unitario*** y ***Nom_articulo***, claramente afectadas a la entidad **ARTÍCULO**. Se establecería entonces el identificativo ***Cod_articulo*** (u otro nombre), que aportaría nuevas dependencias lo que permitiría concretamente llegar a ***Precio_unitario*** y ***Nom_articulo***. Sería preciso entonces, volver a comenzar a partir del grafo inicial.

Si el grafo obtenido conduce a ciclos, se elimina esta anomalía suprimiendo una dependencia funcional. En caso de existir transitividades, éstas deben ser eliminadas.

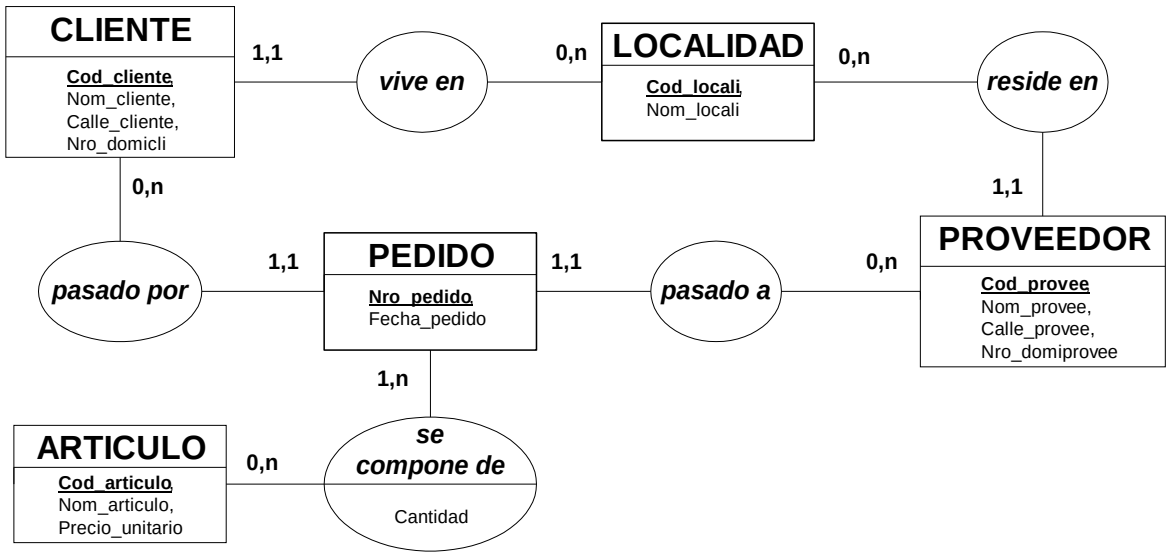
Establecimiento del MCD

Los arcos terminales obtenidos a partir de las propiedades elementales, definen las ENTIDADES. Los atributos que están en el origen de cada uno de esos arcos, serán los identificativos de tales entidades. Aplicando estas definiciones al grafo, se tiene que:

ENTIDAD	Atributo	Atributo	Atributo	Atributo
PEDIDO	***Nro_pedido***	Fecha_pedido		
CLIENTE		***Cod_cliente***	Nom_cliente	
			Calle_cliente	
			Nro_domicli	
LOCALIDAD			***Cod_locali***	Nom_locali
Relación	*Cantidad*	***Cod_provee***	Nom_provee	
			Calle_provee	
			Nro_domiprovee	
LOCALIDAD (R)			***Cod_locali* (R)**	
ARTÍCULO	***Cod_articulo***	Nom_articulo		
		Precio_unitario		

Los arcos restantes, establecen las relaciones. Las propiedades no aisladas restantes se afectan a las relaciones. Aquellas que no pueden ser asignadas a relaciones, constituirán entidades aisladas.

Las reglas de gestión, deben permitir encontrar las cardinalidades. De esta manera, el modelo quedará:



Es necesario, por último, asegurar que se cumplen las reglas de verificación, normalización y descomposición. Todo ello, ocurre en el ejemplo anterior.

Cuantificación del MCD

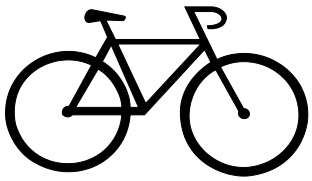
Finalmente, se completa la información del modelo, adicionando fichas con datos de filiación de las entidades incluyendo la cuantificación de los elementos recogidos en las entrevistas. Esto dará una idea del tamaño de modelo completo. Se realizan separadamente tablas de cuantificación para las entidades y para las relaciones.

5.15 MDC extendido

Hasta el momento se han trabajado con elementos conceptuales de los diagramas entidad-relación que resultan genéricos para todos los autores que se explayan sobre el tema. A continuación se agregarán nuevos conceptos primitivos que permitirán agregar expresividad a los modelos conceptuales.

Abstracciones en el diseño del modelo conceptual de datos

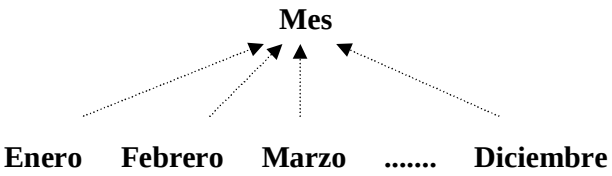
La abstracción es un proceso mental que se aplica al seleccionar algunas características y propiedades de un conjunto de objetos y excluir otras no pertinentes, vale decir que se hace una abstracción al fijar la atención en las propiedades consideradas esenciales de un conjunto de cosas y desechar sus diferencias. Si se considera por ejemplo el siguiente gráfico:



El concepto de bicicleta puede verse como resultado de un proceso de abstracción, lo que hace excluir todos los elementos o detalles de la estructura de la bicicleta (cadena, pedales, frenos, manubrio, etc.) y todas las posibles diferencias entre bicicletas. Comúnmente se asocia un nombre con cada abstracción. El dibujo es una representación de esta abstracción. Otra representación sería una descripción en castellano del mismo dibujo. En el diseño conceptual de datos, se usan tres tipos de abstracciones: clasificación, agregación y generalización.

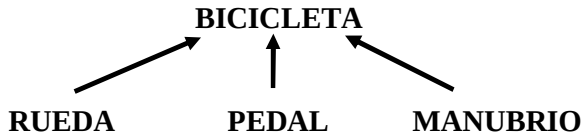
5.15.1Abstracción de clasificación

La abstracción de clasificación, se usa para definir un concepto como una *clase* de objetos de la realidad, caracterizados por propiedades comunes. Por ejemplo, se tiene que el concepto *bicicleta* es la clase cuyos miembros son todas bicicletas (la bicicleta roja, la bicicleta de Claudio, etc.). De igual manera el concepto *mes* es la clase cuyos miembros son *Enero, Febrero, ..., Diciembre* como se muestra en la figura. Se representa gráficamente la clasificación como un árbol de un nivel, que tiene como raíz la clase y como hojas los elementos de la clase. Las ramas del árbol se representan por líneas discontinuas. Cada rama del árbol indica que un nodo hoja es un miembro de la clase que representa la raíz.

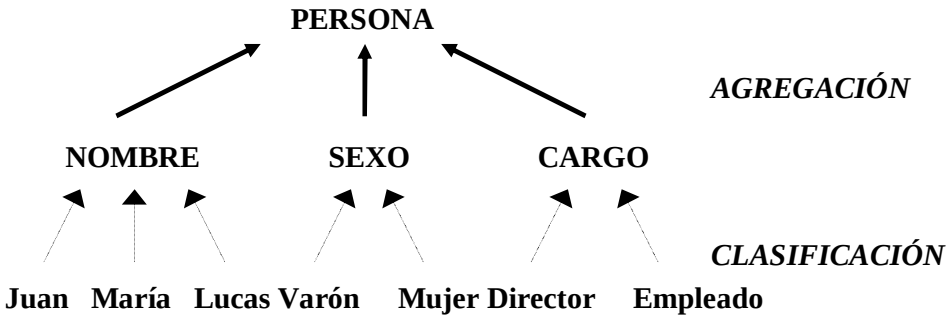


5.15.2 Abstracción de agregación

Una abstracción de agregación define una nueva clase a partir de un conjunto de (otras) clases que representan sus partes componentes. Se aplica esta abstracción cuando, partiendo de las clases rueda, pedal, manubrio, etc., se forma la clase BICICLETA. La abstracción por agregación se representa por un árbol de un nivel en el cual todos los nodos son clases; la raíz representa la clase creada por agregación de las clases representadas por las hojas. Cada rama del árbol indica que una clase hoja es una parte de la clase representada por la raíz. Para distinguir la agregación de la clasificación las ramas dirigidas están representadas por líneas gruesas que van de los componentes a los objetos agregados.

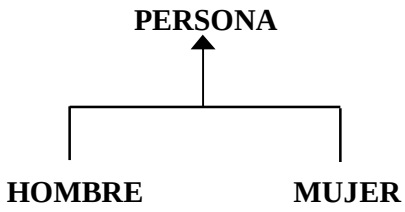


La clasificación y la agregación son las dos abstracciones básicas utilizadas para construir estructuras de datos dentro de los modelos y de muchos lenguajes convencionales de programación. La clasificación es el procedimiento utilizado cuando, partiendo de elementos individuales de información, se identifican tipos de atributos. La agregación es el procedimiento mediante el cual se reúnen tipos de campos relacionados en grupos como por ejemplo tipos de registros.



5.15.3 Abstracción de generalización

Una abstracción de generalización define una relación de subconjunto entre los elementos de dos o más clases. Por ejemplo, la clase **VEHICULO** es una generalización de la clase **BICICLETA**, ya que todas las bicicletas son vehículos. Asimismo, la clase **PERSONA** es una generalización de las clases **HOMBRE y MUJER**.



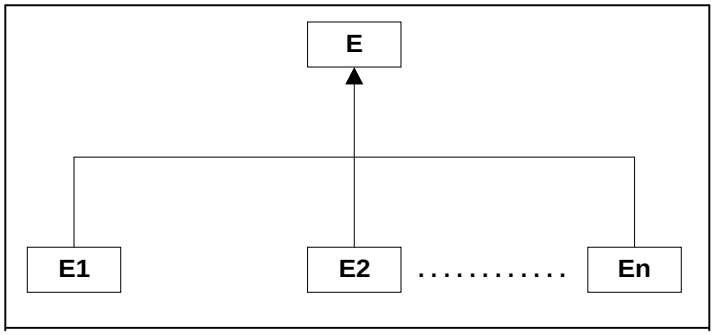
Cada generalización se representa con árbol de un nivel, en el que todos los nodos son clases, con la clase genérica como raíz y las clases subconjunto como hojas; cada rama del

árbol expresa que una clase hoja es un subconjunto de la clase raíz. Para distinguir la generalización de otras abstracciones, se usa una flecha sencilla apuntando hacia la raíz. La abstracción de generalización, a pesar de ser muy común e intuitiva, no se usa en muchos modelos de datos. Sin embargo, es muy útil por su cualidad fundamental de herencia: en una generalización, todas las abstracciones definidas para la clase genérica son heredadas por las clases subconjunto. Dentro del concepto de abstracción de generalización, se encuentra embebido el de cobertura. La cobertura se clasifica en función de las cardinalidades mínima y máxima, determinando si es total o parcial o si es exclusiva o superpuesta. Es total (t), si cada elemento de la clase genérica corresponde al menos a un elemento de las clases subconjunto; es parcial (p) si existe algún elemento de la clase genérica que no corresponde a ningún elemento de las clases subconjunto. Es exclusiva (e) si cada elemento de la clase genérica corresponde, a lo sumo, a un elemento de las clases subconjunto; es superpuesta (s) si, al contrario, existe algún elemento de la clase genérica que corresponde a elementos de dos o más clases subconjunto diferentes.

Las tres abstracciones son independientes: ninguna de ellas puede describirse en función de las otras, y cada una de ellas proporciona un mecanismo diferenciado en el proceso de estructuración de la información.

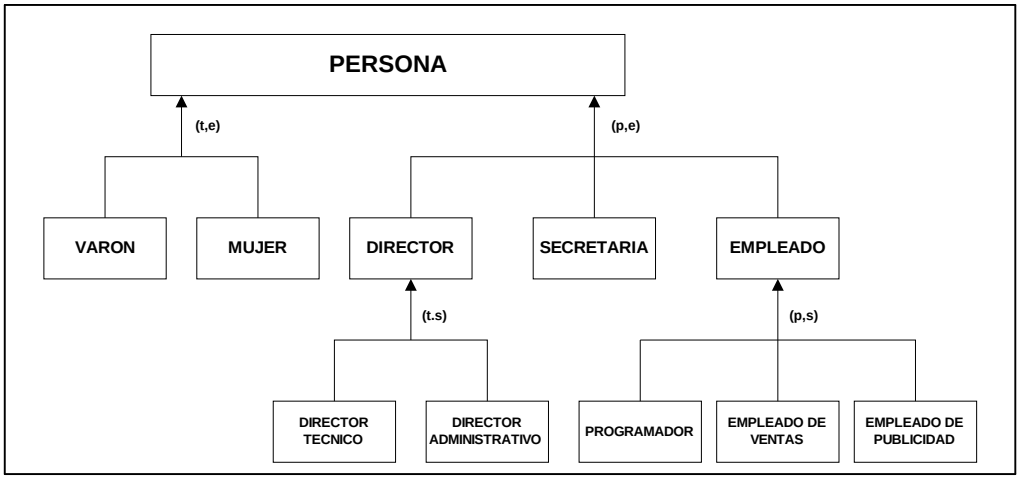
5.15.4 Jerarquía de generalización

En el modelo entidad–relación es posible establecer jerarquías de *generalización* entre las entidades. Una entidad **E**, es una generalización de un grupo de entidades $\mathbf{E_1, E_2, ..., E_n}$, si cada objeto de las clases $\mathbf{E_1, E_2, ..., E_n}$, es también un objeto de la clase **E**. Una generalización en el modelo entidad – relación, expresa la abstracción de generalización expuesta anteriormente. Nótese que solamente se mostrará en la representación gráfica, el bloque de entidad y el nombre de ella. La representación esquemática será la siguiente:

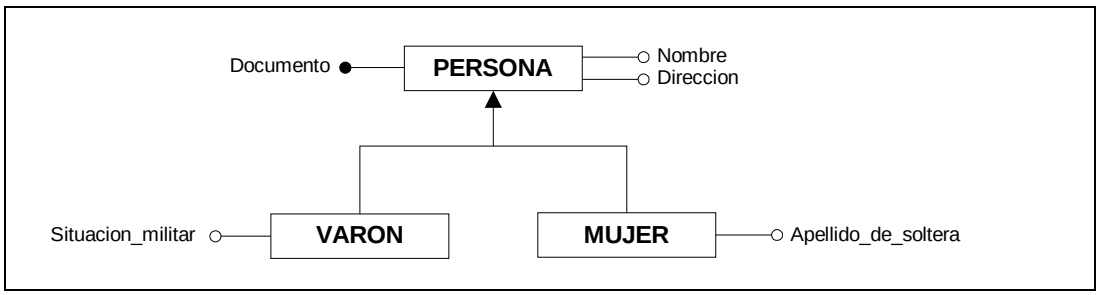


En la figura, la flecha indica la entidad generalizada.

Cada entidad puede participar en múltiples generalizaciones, posiblemente en el papel de entidad genérica con respecto a una generalización y en el papel de entidad subconjunto con respecto a otra generalización. La siguiente figura, presenta una jerarquía de generalización compleja para la entidad **PERSONA**. Lo opuesto a la generalización se denomina especialización.



La propiedad fundamental de la abstracción de generalización es que todas las propiedades de la entidad genérica son heredadas por las entidades subconjunto. En términos del modelo entidad - relación, esto significa que cada atributo, relación o generalización definido para la entidad genérica, será heredado automáticamente por todas las entidades subconjunto de la generalización. Esta propiedad es importante, porque permite construir jerarquías de generalización estructuradas. Considerando el siguiente gráfico, la propiedad de herencia establece que los atributos *nombre* y *dirección* de **PERSONA** son también atributos de **VARON** y **MUJER**; luego, pueden ser eliminados de las entidades subconjunto, simplificando el esquema.

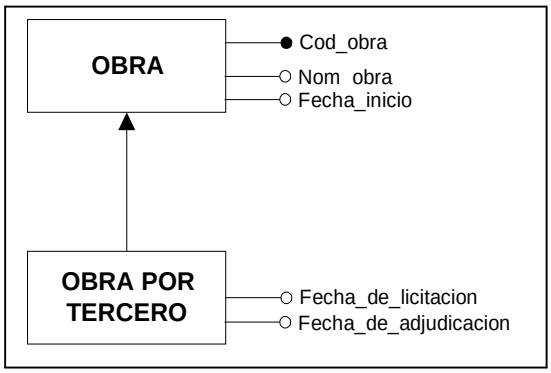


Si bien la simbología utilizada difiere de la propuesta anteriormente, su significado no cambia en absoluto.

5.15.5 Subconjuntos

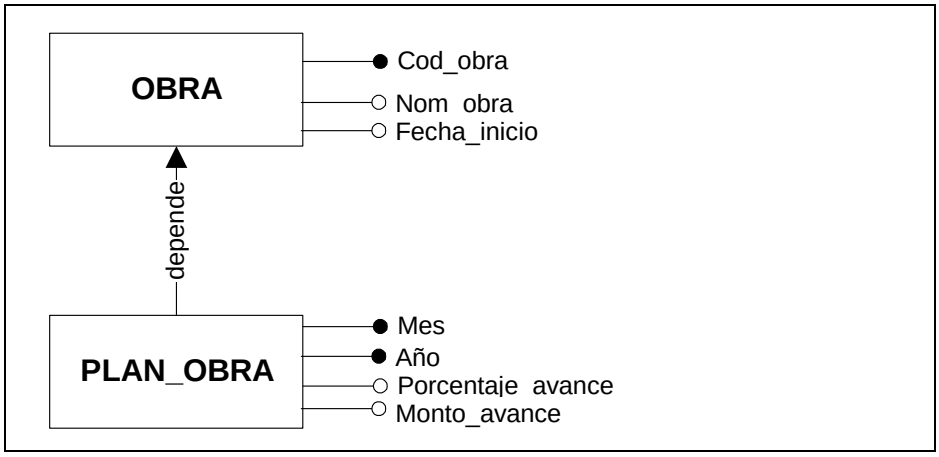
Un subconjunto es un caso particular de jerarquía de generalización, con una sola entidad subconjunto. Se tratarán por separado los subconjuntos porque la cobertura de un subconjunto, es claramente parcial y exclusiva y no necesita definirse. Se representan los subconjuntos con una flecha que une la entidad genérica a la entidad subconjunto y apunta hacia la entidad genérica, como lo indica la figura. Suponga por ejemplo la existencia de la entidad **OBRA** que tendrá como atributos (*Cod_obra, Nom_obra, Fecha_inicio*), pero hay un grupo de obras que son realizadas por empresas privadas y adjudicadas a través de un proceso de licitación que necesariamente requieren registrar además de los atributos comunes, la *Fecha_de_licitación* y la *Fecha_de_adjudicación*. Surge entonces la necesidad de crear una especialización (**OBRA POR TERCERO**) de la entidad **OBRA** que contenga tales

propiedades.



5.15.6 Entidades dependientes y entidades débiles

Existen casos de características similares a las de subconjuntos, con la diferencia que las entidades no son parte de otra de jerarquía superior y de la que heredan sus atributos, sino que, poseen atributos propios pero no identidad propia única (identificativo único) y para ello, requieren heredarlo de la entidad de nivel superior de la que son totalmente dependientes. Por ejemplo, la planificación de ejecución (*Porcentaje_avance, Monto_avance*) de una obra, es totalmente dependiente de la obra en cuestión, siendo el identificativo para cada ocurrencia, el *Cod_obra*, *Mes* y *Año*. De acuerdo a la teoría desarrollada durante el curso, se ha establecido que puede haber atributos asociados a entidades y a relaciones. Existe gran cantidad de software de diseño asistido en los que no resulta posible asociar atributos a una relación, y además, se limitan a trabajar pura y exclusivamente con relaciones binarias. Para salvar esta situación, se suelen crear entidades totalmente dependientes también denominadas entidades débiles que no poseen un atributo identificativo, sino que el identificativo estará dado por la concatenación de las claves de las entidades que se vinculan con él (normalización de relaciones). En estos casos, esta entidad será totalmente dependiente de al menos dos entidades "fuertes".



5.16 Estrategias de diseño para los modelos conceptuales de datos

La creación de un modelo conceptual de datos, es un proceso incremental: la percepción de la realidad se refina y enriquece de forma progresiva, y el esquema conceptual se desarrolla gradualmente. Las distintas formas de encarar el estudio y comenzar el diseño, son las que dan el nombre a las estrategias que se mencionan a continuación:

5.16.1 Estrategia descendente

Se parte de un esquema que contiene abstracciones de alto nivel y luego se aplican refinaciones descendentes sucesivas. Se puede comenzar por especificar unos cuantos tipos de entidades de alto nivel; luego al especificar sus atributos, se dividen en tipos de entidades y de relaciones de menor nivel. El proceso de especialización para refinar un tipo de entidades convirtiéndolo en subclases, es otro ejemplo de estrategia de diseño descendente.

5.16.2 Estrategia ascendente

Se parte de un esquema que contiene abstracciones básicas, y luego se combinan o se les agregan otras abstracciones. Por ejemplo se puede comenzar con los atributos y agruparlos en tipos de entidades y relaciones. Conforme avanza el diseño se podrían agregar nuevas relaciones entre tipos de entidades. El proceso de generalizar subclases para obtener clases generalizadas de más alto nivel es otro ejemplo de estrategia de diseño ascendente.

5.16.3 Estrategia centrífuga

Este es un caso especial de estrategia ascendente, en la que la atención se concentra en un conjunto principal o núcleo de conceptos que son los más evidentes. A continuación el modelado se extiende hacia afuera al considerar conceptos nuevos en las cercanías de los ya existentes. Por ejemplo se podría especificar en el esquema unos cuantos tipos de entidades obvias y continuar agregando otros tipos de entidades y de relaciones vinculadas con ellos.

5.16.4 Estrategia mixta

En vez de seguir una estrategia específica durante todo el diseño, los requerimientos se dividirán según una estrategia descendente, y se diseñará una parte del modelo para cada partición de acuerdo con una estrategia ascendente. Por último, se combinarán las diferentes partes del modelo.

5.17 Conclusión

Los modelos de datos son elementos que permiten describir la realidad. El bloque de construcción elemental común a todos los modelos de datos, es una pequeña colección de mecanismos de abstracción primitivos. Las siguientes consideraciones justifican acabadamente la importancia del enfoque conceptual:

- El diseño conceptual no se ayuda mucho de herramientas automáticas.

- El diseñador asume total responsabilidad sobre el proceso de entender y transformar los requerimientos en esquemas conceptuales.

- A partir de la primera conceptualización, muchos sistemas de bases de datos ofrecen herramientas para la creación rápida de prototipos, usando lenguajes de cuarta generación de aplicaciones, formatos de pantallas e informes.

- Estas herramientas pueden ser usadas directamente por no profesionales para desarrollar bases de datos simples y facilitan el trabajo a los creadores profesionales de bases de datos.

- El diseño conceptual, es la fase más crucial del diseño de bases de datos, y el desarrollo posterior de la tecnología de bases de datos, no cambiará esta situación.

- Aun si se supone que el diseño conceptual está dirigido por un profesional, se alcanzan resultados satisfactorios sólo mediante la cooperación con los usuarios de las bases de datos, quienes tienen que describir las necesidades y de explicar el significado de los datos.

- Las características básicas del diseño conceptual y de los modelos conceptuales de datos, son relativamente simples y su entendimiento no requiere mayor conocimiento técnico previo sobre sistemas de bases de datos. De este modo, los usuarios pueden aprender fácilmente lo suficiente sobre diseño conceptual para orientar a los diseñadores en sus decisiones e incluso para diseñar bases de datos simples.

- Una influencia fuerte del usuario final sobre las decisiones de diseño tiene muchas consecuencias positivas:

    - mejora la calidad del esquema conceptual

    - eleva la probabilidad de que el proyecto converja hacia el resultado esperado

    - reduce los costos de desarrollo

- Un gran argumento de fuerza, es su independencia de un fabricante de base de datos en particular. Esta característica genera las siguientes ventajas:

    - La elección del software de base de datos se puede posponer, y el esquema conceptual puede sobrevivir a una decisión tardía de cambiar el software.

    - Si el software de base de datos o los requerimientos de la aplicación cambian, el esquema conceptual puede aún usarse como punto de partida de la nueva actividad de diseño.

- Las diferentes bases de datos, descritas mediante su esquema conceptual, se pueden comparar en un marco homogéneo de trabajo. Esta característica facilita la construcción de sistemas consolidados a partir de varias bases de datos ya existentes y la creación de un diccionario de datos integrado.

Universidad Nacional del Litoral

FACULTAD DE INGENIERÍA
Y CIENCIAS HÍDRICAS

Ingeniería en Informática

Ingeniería de Software I

TEMA VI – Orientación a objetos - UML

Introducción

El lenguaje unificado de modelado (**UML**) es una de las herramientas de mayor uso en el mundo actual para el desarrollo de sistemas. Esto se debe a que permite a los creadores de sistemas poder generar diseños que capturen sus ideas en una forma convencional y uniforme y fácil de comprender para comunicarlas a otras personas.

La comunicación de las ideas es de suma importancia. Antes del advenimiento del UML, los analistas al evaluar los requerimientos de los usuarios, generaban un análisis de requerimientos apoyados en algún tipo de notación que ellos mismos comprendieran (aunque el cliente no lo hiciera) y daban tal análisis a los programadores y esperaban que el producto final cumpliese con lo que el cliente deseaba. Dado que el desarrollo de sistemas es una actividad humana, hay muchas posibilidades de cometer errores en cualquier etapa del proceso.

Conforme aumenta la complejidad del mundo, los sistemas informáticos también lo hacen. Para manejar la complejidad es necesario organizar el proceso de diseño de tal forma que los analistas, desarrolladores, clientes y otras personas vinculadas lo comprendan. El UML proporciona esa organización. Un constructor no podría crear una compleja estructura de edificios sin crear primero un proyecto detallado. Esos planos tienen notaciones estándares y cualquier profesional de esa especialidad lo podría comprender perfectamente. En sistemas de software ese estándar de facto que se ha adoptado es precisamente el UML.

El UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos. La finalidad de los diagramas es presentar diversas perspectivas de un sistema a las cuales se les conoce como ***modelo***. El modelo UML describe lo que supuestamente hará un sistema (el **qué**), pero no dice **cómo** implementarlo.

El UML es una creación de *Grady Booch, James Rumbaugh* e *Ivar Jacobson*. Cada uno de ellos durante los años ochenta y noventa diseñó su propia metodología para el análisis y diseño orientado a objetos. Éstas predominaron sobre sus competidores y a mediados de los noventa empezaron a intercambiar ideas entre sí y consecuentemente desarrollaron un trabajo en conjunto. Como se indicara, las distintas ideas eran relativas al análisis y diseño orientado a objetos, razón por la cual para comprender todo lo relacionado con el UML se requiere tener claro todo lo relacionado a la orientación a objetos.

6.1 ORIENTACIÓN A OBJETOS

Antes de comenzar con los fundamentos de la orientación a objetos, conviene mencionar a su oponente (el paradigma anterior a la OO y que se encuentra en proceso de cambio): El **análisis y diseño estructurado** (o funcional). En éste se construyen modelos basados en el procesamiento de datos. Quien utilice este enfoque intentará dividir el problema bajo estudio identificando una serie de procesos, manipulaciones o tratamientos de datos (llamados funciones o procedimientos en las fases de diseño e implementación) que, organizados de modo que puedan llamarse unos a otros, proporcionen la solución. Siempre existe una separación entre datos y procesos: los procesos manipulan y usan datos, pero no se integran con ellos.

El **paradigma estructurado** (análisis estructurado + diseño estructurado + programación estructurada) en la ingeniería de software se basa en la **abstracción por descomposición funcional o por procedimientos**: el problema estudiado se descompone en una serie de capas sucesivas de procesos, hasta que finalmente se descompone en procesos relativamente fáciles de implementar y codificar (desarrollo *top-down*). Un programa estructurado se divide en unidades lógicas (módulos) mediante el uso de funciones y procedimientos; los detalles más internos del programa residen en los módulos de más bajo nivel y los módulos de más alto nivel se encargan del control lógico del programa.

El paradigma estructurado mantendrá su vigencia, posiblemente, por cierto tiempo en la ingeniería del software pues muchas aplicaciones informáticas de gestión utilizan mayoritariamente pseudoobjetos u objetos que no pueden considerarse objetos de pleno derecho (por ejemplo, objetos sin atributos u objetos sin métodos), generalmente asociados a las estructuras tradicionales de datos. Otro de los defectos más comúnmente señalados por los críticos del paradigma estructurado es la separación clara e insalvable entre el análisis y el diseño, es decir, entre lo que se quiere que haga el sistema y cómo lo hace. En el paradigma OO la frontera entre el análisis y el diseño es difusa, y los objetos se introducen desde el principio. Es más natural pasar del análisis a la implementación en el paradigma OO que en el estructurado: el salto conceptual es menor en aquél.

La conversión de los términos del análisis OO hacia construcciones en lenguajes de programación OO es bastante directa, lo que constituye una importante ventaja sobre el paradigma estructurado.

Algunos autores consideran que el enfoque OO ha evolucionado a partir del enfoque estructurado y otros que es un salto revolucionario, cualitativo más que cuantitativo. Desde luego, los partidarios del salto revolucionario suelen ser firmes partidarios de la OO, cuando no creadores de metodologías OO. Así, en la obra *Object- Oriented Modeling and Design* **[Rumbaugh et al, 1991]** se considera que el diseño orientado a objetos es una nueva forma de pensar en los problemas usando modelos sobre conceptos del mundo real y, en *Ingeniería del Software - Un Enfoque práctico* **[Pressman, 1992]**, Pressman refiere que "*a diferencia de otros métodos de diseño, el diseño orientado a objetos da como resultado un diseño que interconecta los objetos de datos y las operaciones, de forma que modulariza la información y el procesamiento en vez de sólo la información. La naturaleza del diseño orientado a objetos está ligada a tres conceptos básicos: abstracción, modularidad y ocultación de la información.*"

Concluyendo, dado que no hay un acuerdo unánime, es que realmente el enfoque OO utiliza abstracciones no presentes en el enfoque estructurado y que no admiten equivalencia.

El paradigma de objetos fomenta una metodología basada en componentes de software de manera que primero se genera un sistema mediante un conjunto de objetos, luego podrá ampliarse agregándole funcionalidad a los componentes ya generados o agregándole nuevos componentes y finalmente, se podrán volver a utilizar los objetos generados para un nuevo sistema.

Para la introducción a la orientación a objetos se tratarán los siguientes conceptos fundamentales (algunos de ellos ya tratados en el tema anterior pero desde una perspectiva acotada a los datos):

- **Abstracción**
- **Encapsulamiento**
- **Herencia**
- **Polimorfismo**
- **Envío de mensajes**
- **Relaciones entre clases**
- **Agregación y composición**
- **Clasificación**

Un objeto es una instancia de una clase (o categoría). Cuenta con una ***estructura*** (es decir atributos o propiedades) y ***acciones***. Éstas son todas las actividades que el objeto es capaz de realizar. Por ejemplo, dada la clase **Persona** ésta puede tener como atributos *altura, peso, fecha de nacimiento* etc. También puede realizar tareas como *comer, dormir, leer, escribir, hablar,* etc. Si se tuviera que crear un sistema que manejara información acerca de las personas, sería muy probable que se incorporen algunos atributos y acciones en el software. Toda clase puede verse desde tres perspectivas distintas pero complementarias:

1. **Como conjunto o colección de objetos.**

   Esta perspectiva apunta a la columna vertebral del concepto de clase: una clase implica clasificación y abstracción. En realidad, una clase no deja de ser la formalización y verbalización de unos procesos que los seres humanos realizan continuamente, a menudo de forma inconsciente y preprogramada. El ser humano piensa con ideas, con abstractos; y, a medio camino entre la percepción y la cognición, se halla la función de clasificar, es decir, la función de poder afirmar que aquello percibido pertenece a un grupo de cosas (clase) o a otro. Pensar consiste en buscar semejanzas y olvidar diferencias; consiste en abstraer, en generalizar.

   Por la propia naturaleza de la percepción humana, resulta difícil razonar con conceptos que rompen clasificaciones establecidas. Por ejemplo, seguramente muchos biólogos no hayan podido entender qué pasaba cuando se descubrió el ornitorrinco. El ornitorrinco rompía una clasificación fundamental de los zoólogos: la de mamífero y no mamífero. Este curioso animal que pone huevos, rompía la tradicional clasificación según la cual los mamíferos tienen pelo o algo similar en alguna parte del cuerpo, son lactantes siendo crías, son de sangre caliente y **no ponen huevos**.

   Todos los seres humanos sabemos que un pájaro volando en vertical no deja de ser un pájaro. En contraste, las máquinas y los sistemas de software (incluso las inteligencias artificiales) carecen de esa capacidad de abstracción y de esa extraña capacidad informe llamada "sentido común". En el campo del reconocimiento digital de imágenes, por ejemplo, resulta difícil que los sistemas de reconocimiento comprendan que un ave – o un avión, o cualquier forma geométrica – volando en vertical, en horizontal o en cualquier otra orientación sigue siendo la misma ave o el mismo objeto que en reposo. Es decir, resulta difícil que comprendan que se encuentran ante distintos estados de un mismo objeto. Pese a los importantes avances logrados en el campo de las inteligencias artificiales, poco se ha avanzado en el desarrollo de sistemas de IA generales. En consecuencia, el análisis OO, basado en la

abstracción y que incluye la identificación de las clases del dominio del problema, continúa siendo una actividad humana.

2. **Como plantilla para la creación de objetos.**
   En esta perspectiva se destaca que la relación entre una clase y los objetos derivados de ella puede considerarse como la existente entre una fábrica y las cosas producidas por ésta. Un ejemplo: una fábrica de automóviles produce automóviles del mismo modo que una clase Automóvil crea objetos automóviles. Una clase Automóvil solamente producirá objetos automóviles, del mismo modo que una fabrica real de automóviles sólo produce coches, no televisores o aviones.

3. **Como definición de la estructura y comportamiento de una clase.**
   En esta se hace hincapié en que la definición de una clase permite definir una sola vez la estructura común, así como usarla cuantas veces sea necesario.

Se llama atributo a la abstracción de una característica común para todas las instancias de una clase, o a una propiedad o característica de un objeto. Las operaciones de una clase son servicios ofrecidos por ésta, que llevan (o pueden llevar) a cambios en el estado de un objeto de dicha clase. Conceptualmente, una operación puede considerarse como una petición a un objeto para que haga algo. Cuando las clases se implementan en lenguajes OO, se habla de variables de instancia (implementaciones software de atributos) y de métodos (implementaciones software de operaciones). Cada instancia de una clase (u objeto) contiene su propio conjunto de variables de instancia, cuyos valores pueden variar con el tiempo. El conjunto de los valores tomados por las variables de instancia de un objeto, en un instante dado, representa el estado del objeto. Dicho en sentido inverso, el estado de un objeto viene representado por los datos almacenados en sus variables de instancia.

Cuando se consideran sistemas de software, enseguida se percibe la tendencia evolutiva de los lenguajes de programación hacia la representación y reproducción de la manera como nosotros contemplamos el mundo que nos rodea. Muy pocas personas comprenden a primera vista la secuencia de código ensamblador que permite sumar dos números enteros; menos aún pueden pensar directamente en binario; pero casi todos entendemos que una clase Entero representa el conjunto de los números enteros, y que éstos tienen su propio comportamiento, en el cual se incluye la operación suma.

Con la evolución y el desarrollo de los lenguajes de programación, se reproduce el modo de pensar al cual nos ha conducido la evolución, o sea que estos lenguajes tienden a ver el mundo tal como nosotros lo vemos de manera totalmente natural.

6.1.1 La abstracción

Es la vía fundamental por la que los humanos combaten la complejidad. Surge de un reconocimiento de las similitudes entre ciertos objetos, situaciones o procesos del mundo real, y la decisión de concentrarse en esas similitudes e ignorar por el momento las diferencias. Es una descripción simplificada que enfatiza ciertos detalles significativos y suprime otros por irrelevantes. Denota las características esenciales de un objeto que lo

distinguen de todos los demás tipos de objeto y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador. En definitiva, se refiere a quitar propiedades y acciones de un objeto para dejar solamente aquellas necesarias. Diferentes problemas requieren distintas cantidades de información, aun si estos problemas pertenecen a un área en común.

6.1.2 El encapsulamiento

La esencia del encapsulamiento es que cuando un objeto trae consigo su funcionalidad, ésta última se oculta. Por lo general, la mayoría de las personas que ven televisión no sabe ni se preocupa de la complejidad electrónica que hay detrás de la pantalla ni de todas las operaciones que tienen que ocurrir para mostrar una imagen. El televisor hace lo tiene que hacer sin mostrar el proceso necesario para ello (como la mayoría de los electrodomésticos). En un sistema de consta de objetos de software, éstos depende unos de otros de alguna manera. Si uno de ellos falla y los especialistas de software tienen que modificarlo de alguna forma, el ocultar sus operaciones de otros objetos significará que tal vez no será necesario modificar los demás objetos. De esta manera, el monitor de la computadora, en cierto sentido, oculta sus operaciones de la CPU, o sea que si algo falla en el monitor, se reparará o reemplazará sin tener que reparar o reemplazar la CPU.

El encapsulamiento entonces es el ocultamiento de la información. Se esconden todos los detalles internos del sistema que se estudia como una caja negra y resulta más importante comprender **qué** hace que el **cómo** lo hace. La abstracción y el encapsulamiento son conceptos complementarios: la abstracción se centra en el comportamiento observable de un objeto, mientras que el encapsulamiento se centra en la implementación que da lugar a este comportamiento. El encapsulamiento se consigue mediante la ocultación de la información, que es el proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales; típicamente, la estructura de un objeto está oculta, así como la implementación de sus métodos.

Entonces, cada clase debe tener dos partes: *una interfaz* y *una implementación*. La *interfaz* de una clase, captura solamente su vista externa que alcanza a la abstracción que se ha hecho del comportamiento común de todas las instancias de la clase y ahí se listan o declaran las acciones que pueden tener lugar. La *implementación* de una clase comprende la representación de la abstracción así como los mecanismos que consiguen el comportamiento deseado. El encapsulamiento es la característica fundamental que debe poseer la implementación ya que se deben esconder todos los detalles de cómo se hacen las cosas listadas en la interfaz. Para el caso planteado del televisor, la interfaz del mismo viene dada a través de las perillas y botones o el control remoto.

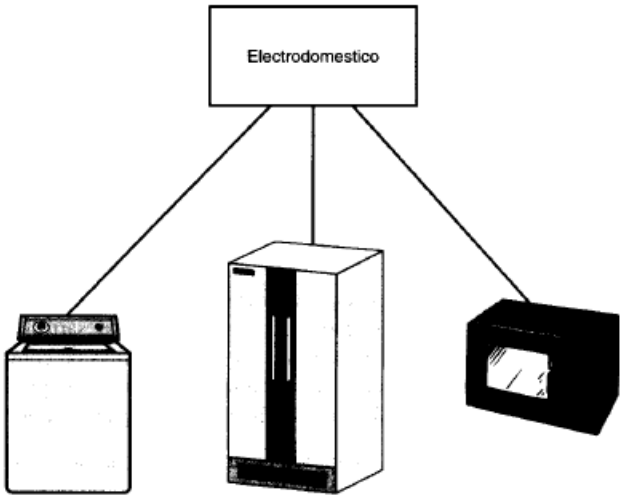
6.1.3 La herencia

La herencia es la jerarquía de clases más importante y es un elemento esencial de los sistemas orientados a objetos. Define una relación entre clases en la que una clase comparte la estructura de comportamiento definida en una o más clases (herencia simple o herencia múltiple respectivamente). La herencia denota una relación ***es un***. A medida que se desarrolla la jerarquía de herencias, la estructura y comportamiento comunes a diferentes clases tenderá a migrar hacia superclases comunes. Por esta razón se dice que la herencia es

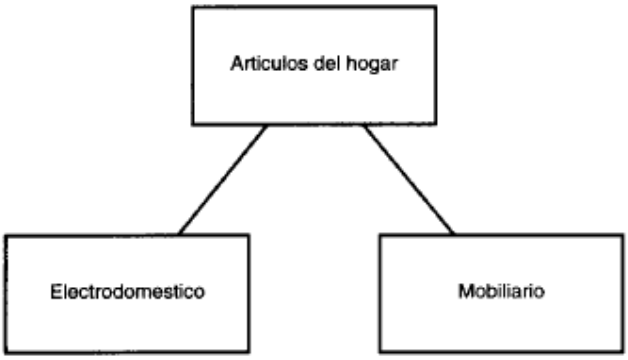
una jerarquía de *generalización/especialización*: las superclases representan abstracciones generalizadas y las subclases representan especializaciones en las que los atributos y métodos de la superclase sufren añadidos, modificaciones o incluso ocultaciones. Si no existiese la herencia, cada clase sería una unidad independiente desarrollada a partir de cero. El hecho que una subclase herede de otra clase implica que se viola el encapsulamiento de la superclase. Los distintos lenguajes de programación, hacen concesiones al respecto, definiendo por ejemplo partes *privadas, protegidas y públicas* las que restringen el acceso de los clientes.

La herencia múltiple es conceptualmente correcta, pero en la práctica introducen ciertas complejidades en los lenguajes de programación. Estos problemas son por ejemplo, colisiones entre nombres de superclases diferentes y la herencia repetida.

Por ejemplo, para el caso del televisor o la lavadora o heladera, microondas, lavaplatos, etc, si bien son clases, forman parte de una clase más genérica: **Electrodomestico**. Un electrodoméstico cuenta con los atributos de interruptor y cable eléctrico y las operaciones de encendido y apagado. Cada una de las clases que dependen de **Electrodomestico** heredarán los mismos atributos y operaciones. Se dice además que **Electrodomestico** es la superclase y que las otras son subclases.



La herencia no tiene porqué terminar ahí, pueden ocurrir que **Electrodomestico** sea a su vez subclase de **Articulos del hogar** como se muestra en la figura:



6.1.4 El polimorfismo

Según Rumbaugh: "*Polimorfismo: Toma de varias formas; propiedad que permite a una operación tener distintos comportamientos en diferentes clases*". En ocasiones puede ocurrir que una misma operación tenga el mismo nombre en diferentes clases, como ser ***abrir()*** se puede abrir una puerta, una persiana, un periódico, una cuenta bancaria, un regalo, etc. y en caso se hará un acción diferente. En la orientación a objetos, cada clase ***sabe*** como realizar tal operación. Esto es el polimorfismo.

En primera instancia, parecería que este concepto es más importante para los desarrolladores que para los modeladores, ya que los primeros tienen que crear el software que implemente tales métodos en los programas y deben estar conscientes de diferencias importantes entre las operaciones que pudieran tener el mismo nombre. No obstante también es importante para los modeladores ya que permite hablar con el cliente en sus propias palabras y terminología.

6.1.5 Envío de mensajes

Se mencionó que en un sistema los objetos trabajan en conjunto. Esto se logra mediante el envío de mensajes entre ellos. Un objeto envía a otro un mensaje para realizar una operación y el objeto receptor la ejecutará.

Un televisor y su control remoto son un ejemplo muy intuitivo. Cuando se presiona el botón de encendido el control remoto le envía un mensaje al televisor para que se encienda. El televisor recibe el mensaje, lo identifica como una petición para encender y ejecuta ese método (encender). Cuando se desea ver otro canal, se presiona el botón correspondiente y nuevamente se envía otro mensaje que el televisor recibe y procesa. Muchas de las cosas que se hace mediante el control remoto también se podrían hacer sobre el televisor presionando los botones correspondientes. La interfaz que el televisor presenta no es la misma que le muestra el control remoto sin embargo el efecto es el mismo.

6.1.6 Relaciones entre clases

Considérense las analogías y diferencias entre las siguientes clases de objetos: *flores, margaritas, rosas rojas, rosas amarillas, pétalos y mariquitas*. Supónganse las siguientes observaciones:

- Una margarita es un tipo de flor,
- Una rosa es un tipo (distinto) de flor,
- Las rosas rojas y las amarillas son tipos de rosas.
- Un pétalo es una parte de ambos tipos de flores.
- Las mariquitas se comen ciertas plagas como los pulgones que pueden infectar ciertos tipos de flores.

Se establecen relaciones entre dos clases por las siguientes razones:

- Una relación entre clases podría indicar un tipo de **compartición**. Por ejemplo, las margaritas y las rosas son tipos de flores, lo que indica que ambas tienen pétalos con colores llamativos, fragancia, etc.
- Una relación entre clases podría indicar algún tipo de **conexión semántica**, así se dice que las rosas rojas y las amarillas se parecen más entre ellas que las margaritas y las rosas, y las margaritas y las rosas se relacionan entre ellas más estrechamente que los pétalos y las flores. Análogamente existe una conexión simbiótica entre las mariquitas y las flores: las mariquitas protegen a las flores de ciertas plagas que a su vez sirven de fuente de alimento a la mariquita.

  En total existen tres tipos básicos de relaciones entre clases:

  - La generalización / especialización que denota **es un**. Por ejemplo, la rosa **es un** tipo de flor lo que quiere decir que una rosa es una subclase especializada de una clase más general (la de las flores). **Herencia**
  - La relación todo/parte que denota una relación **parte de**. Por ejemplo, un pétalo no es un tipo de flor, sino que es una **parte de** la flor. **Agregación**
  - La asociación que denota alguna dependencia semántica entre clases de otro modo independientes como entre las mariquitas y las flores. **Asociación**.

Las asociaciones son el tipo más general pero también el de mayor debilidad semántica. La identificación de asociaciones entre clases, es frecuentemente una actividad de análisis y diseño inicial, momento en el cual se comienza a descubrir las dependencias generales entre las abstracciones. A medida que se continúa el diseño y la implantación, se refinarán a menudo estas asociaciones débiles orientándolas hacia una de las otras relaciones de clase más concretas. Aparecen entonces las agregaciones y la herencia. De todas maneras se necesitan las relaciones de uso que establecen los enlaces entre las clases.

1. **Relación de herencia**

   La herencia permite que unos objetos puedan basarse en otros ya existentes. En términos de clases, la herencia es el mecanismo por el cual una clase **X** puede heredar propiedades de una clase **Y** de modo que los objetos de la clase **X** tengan acceso a los atributos y operaciones de la clase **Y** sin necesidad de redefinirlos. Suelen usarse los términos *hijo/padre* o *subclase/superclase* para designar el par. A veces se dice que la subclase es una especialización de la superclase y que la superclase es una generalización de las subclases.

   La herencia presenta dos cualidades contradictorias entre sí. A saber: una clase hija extiende o amplia el comportamiento de la clase padre, pero también restringe o limita a la clase padre (una subclase se encuentra más especializada que su clase padre). Existe cierta tirantez esencial, constitutiva, entre los dos conceptos (herencia como extensión y herencia como especialización); a veces se olvida esta tensión, pero siempre sigue ahí.

   Suele identificarse la herencia mediante la regla “es un” o “es un tipo de”. Por ejemplo, toda ***Motocicleta*** es un ***Vehículo***, luego la clase ***Motocicleta*** es una subclase de la clase ***Vehículo***. En general, no resulta recomendable emplear la herencia cuando no funcione la regla “X es un Y” o, más precisamente, cuando no pueda justificarse que todo instancia de la clase X es tam-

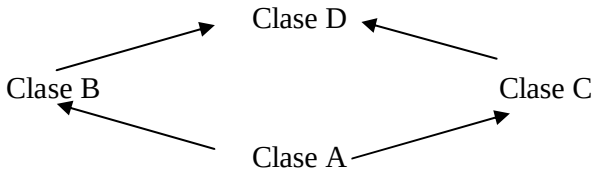
bién una instancia de Y. Por ejemplo: una clase ***ConductorMotocicleta*** que heredase de dos superclases ***Persona*** y ***Motocicleta*** constituiría un mal uso de la herencia ya que no toda instancia de ***ConductorMotocicleta*** es una instancia de ***Motocicleta***. Del mismo modo, no conviene establecer una relación de herencia entre una clase ***Motor*** y una clase ***Coche***, pues un coche no es un género de motores. El uso de la herencia para reutilizar código entre clases que incumplen la regla "es un" suele considerarse incorrecto, aunque a veces se permite y se denomina herencia de implementación o de funcionalidad.

La relación "es un" ilustra una característica crucial de la herencia: un objeto de una subclase puede usarse en cualquier lugar donde se admita un objeto de la superclase. Lo contrario no es cierto (esto también ocurre en el mundo real: los padres, por ejemplo, no heredan los rasgos de los hijos ni pueden intercambiarse por ellos).

La noción de que un objeto de una clase hija puede sustituirse por un objeto de la clase padre conduce inexorablemente a que se incorporen, cuando hablemos de los objetos pertenecientes a una clase, los objetos pertenecientes a todas las subclases. Por claridad, se usa "la clase" de un objeto para referirse a la clase más especializada de la cual es instancia el objeto. Lo dicho en el párrafo inmediatamente superior puede escribirse un poco más precisamente: ***un objeto de una clase especializada puede substituirse por un objeto de una clase más general en cualquier situación donde se espere un miembro de la clase general, pero no al revés***.

La herencia se clasifica asimismo en herencia simple y herencia múltiple. En la herencia simple, una clase sólo hereda (es subclase) de una superclase. En la herencia múltiple, una subclase admite más de una superclase. Entendamos que heredar de más de una superclase no quiere decir que la herencia múltiple consista en que una subclase pueda heredar de una clase que sea, a su vez, subclase de otra clase. Una subclase que herede, mediante herencia múltiple, de dos o más superclases puede mezclar las propiedades de las superclases, en ocasiones de forma ambigua o poco recomendable. Dos problemas fundamentales se presentan con la herencia múltiple:

- Herencia repetida: La clase A hereda de B y C, que a su vez derivan de D. Consecuencia: la clase A hereda dos veces de D.

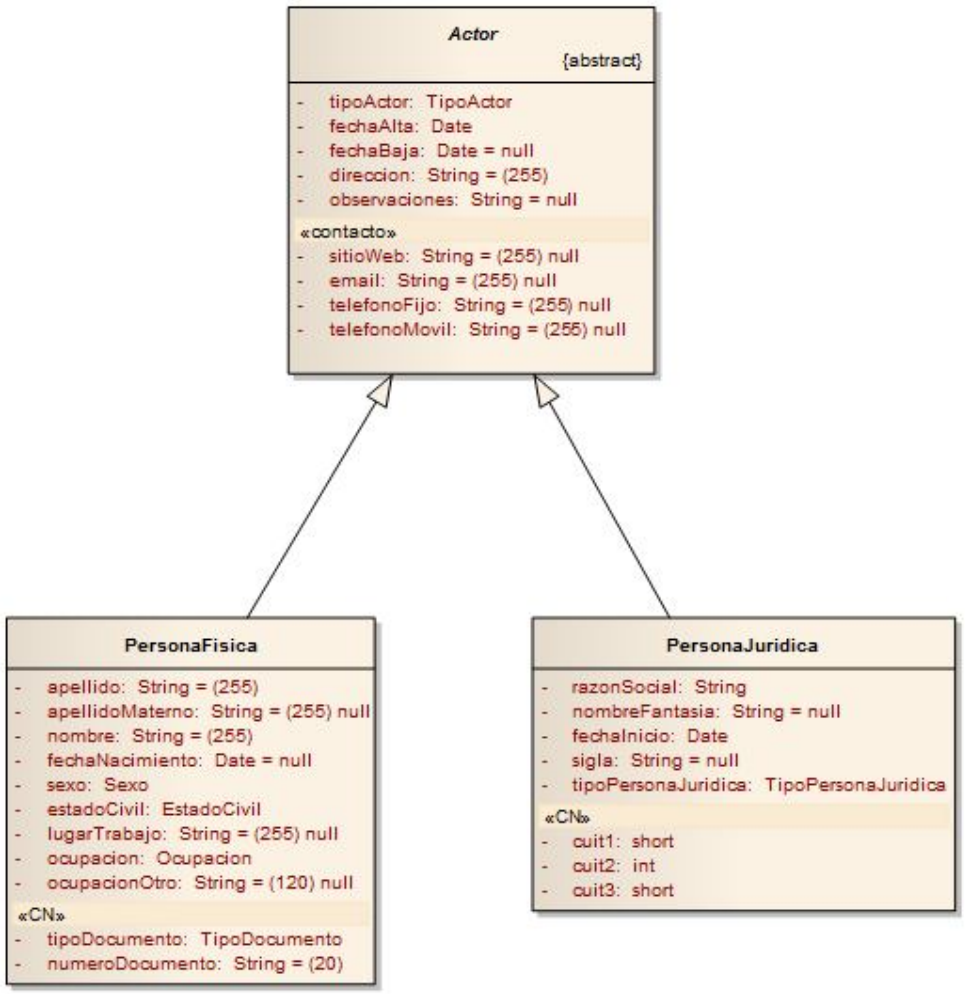


- Conflictos de nombres. Si una clase A hereda simultáneamente de dos superclases B y C, aparecerá un conflicto de nombres si usan el mismo nombre para algún atributo o método. ¿Qué definición usará A del atributo o del método con el mismo nombre? ¿La de la superclase B o la de C? ¿O ninguna de ellas? Generalmente este problema se solu-

ciona redefiniendo en la subclase la propiedad o método con el mismo nombre en las superclases.

Con la herencia simple, cada subclase tiene exactamente una superclase. Sin embargo, fuerza frecuentemente al programador a derivar de una sola entre dos clases factibles. Esto limita la aplicabilidad de las clases predefinidas haciendo muchas veces necesario el duplicar código. Por ejemplo, no existe forma de derivar un gráfico que es a la vez un círculo y una imagen; hay que derivar de uno o del otro y ***reimplantar la funcionalidad de la clase que se excluyó***.

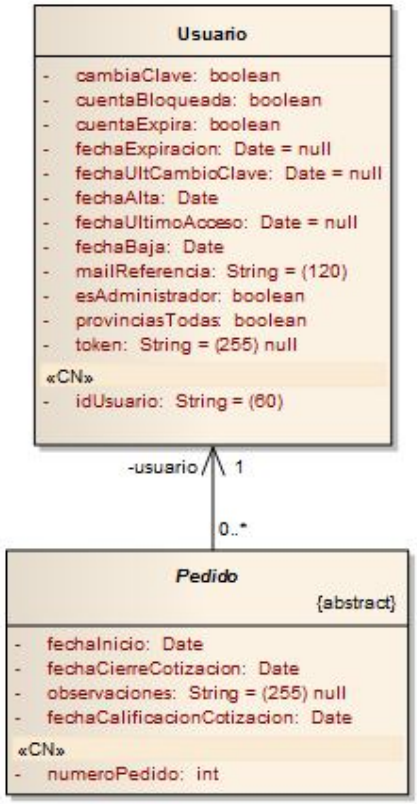
Ejemplo de herencia:



2. Relación de asociación.

Suponga el caso típico de factura y artículos de las factura. Esto se puede representar mediante una asociación simple entre las dos clases. Esta asociación sugiere una relación bidireccional ya que dada una instancia de artículo se debería poder encontrar la factura que denota su venta y dada una instancia de factura se deberían poder localizar todos los artículos que la componen. Esta es una asociación de uno a muchos en la que cada instancia

de artículo puede tener un puntero a su última factura y cada instancia de venta puede tener una colección de punteros que denota los artículos vendidos. La asociación sólo denota una dependencia semántica y no establece la dirección de esa dependencia (a menos que se indique expresamente, una asociación implica relación bidireccional) ni establece la forma exacta en que una clase se relaciona con otra (solo puede denotarse esta semántica nombrando el rol que desempeña cada clase en relación con la otra). Sin embargo, esta semántica es suficiente durante el análisis de un problema ya que es necesario identificar estas dependencias determinando los participantes, rol y su cardinalidad. La cardinalidad indica la multiplicidad de la asociación. Es el mismo concepto de funcionalidad visto anteriormente (uno a uno, uno a muchos y muchos a muchos). En el ejemplo se muestra un Pedido que es realizado por uno solamente un usuario. La visibilidad es desde el pedido hacia el usuario y se indica con una punta de flecha.

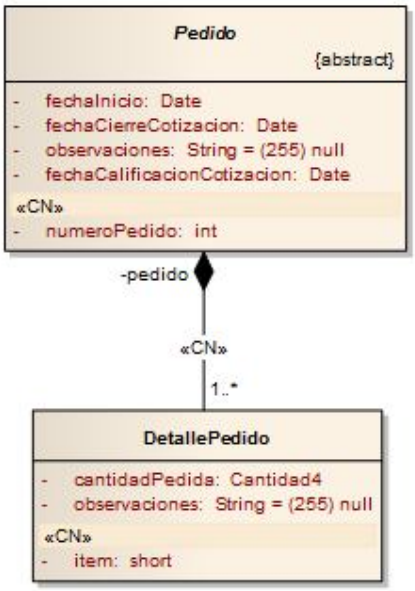


3. Relación de agregación / composición.

Las **agregaciones** son una especialización de las asociaciones, vinculada a una relación de la forma todo-parte. Los objetos parte forman parte del objeto todo, pero la vinculación entre las partes y el todo no es absoluta: se pueden crear y destruir de modo independiente instancias de cada clase involucrada en la relación. La relación todo-parte suele reconocerse porque se manifiesta en la forma de "X está compuesto por Y". Por ejemplo: "los dibujos están formados por elementos gráficos", "la casa está formada por muros", etcétera. Un ejemplo de agregación nos lo da la relación entre un mouse y una computadora. Un mouse puede quitarse de una computadora y colocar-

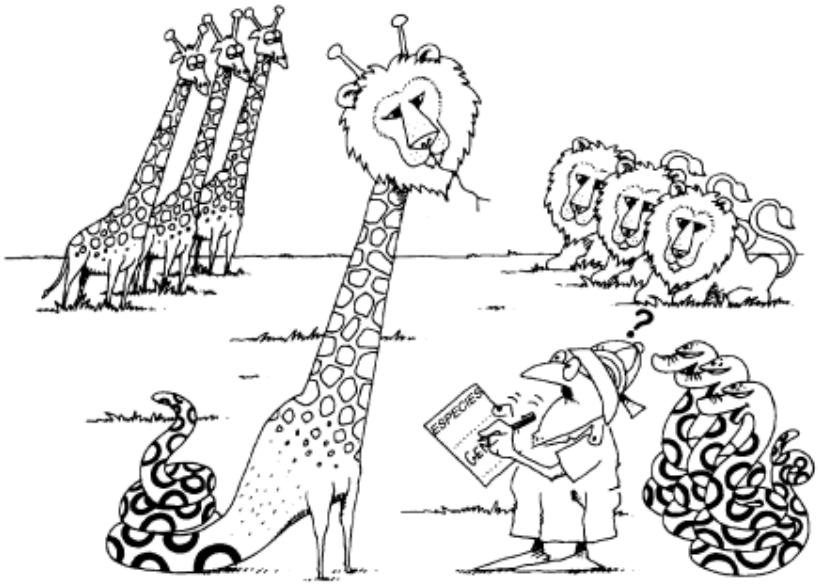
se en otra. Si se considera una clase equipo de club y un jugador, correspondientes a las entidades del mundo real que todos conocemos, la relación entre ambas es una agregación. Todos los objetos jugador de una instancia de equipo de club pueden ser destruidos, y sin embargo la instancia de equipo seguirá existiendo. En el mundo real, puede haber equipos de club sin jugadores. También es posible añadir nuevas instancias de jugadores a un objeto equipo de club ya creado. En el mundo real, pueden aparecer nuevos jugadores e incorporarse a equipos y un equipo puede prestar jugadores a otro.

Las **composiciones** son también una especialización de las asociaciones, vinculada asimismo a una relación del tipo todo-parte, pero con un vínculo absoluto y permanente. La composición indica que los objetos parte están contenidos físicamente en el objeto todo. Como consecuencia, los tiempos de vida de las partes se hallan fuertemente relacionados con el tiempo de vida del todo. Cuando se crea una instancia del objeto todo, se crea una instancia de cada uno de sus objetos parte; y cuando se destruye la instancia todo, se destruyen todas las instancias de los objetos parte. Un objeto parte no puede asignarse a un objeto todo con el cual no se haya creado.



La **agregación** puede o no denotar contención física. Por ejemplo un avión, se compone de alas, motores, tren de aterrizaje, etc. o sea, es un caso de contención física. Una persona puede tener un historial de estados civiles pero estos estados no son de ninguna manera parte física de la persona. Esta relación todo/parte es más conceptual por lo tanto menos directa que la agregación física de las partes que conforman un avión.

6.1.7 La clasificación



La clasificación es el medio por el que ordenamos el conocimiento. En el diseño orientado a objetos, el reconocimiento de la similitud entre las cosas nos permite exponer lo que tienen en común en abstracciones clave y mecanismos, y eventualmente nos lleva a arquitecturas más pequeñas y simples. No existe una receta para la clasificación. No existe a lo que se pueda llamar una estructura de clases *perfecta* ni el conjunto de objetos *correcto*. Al igual que en cualquier disciplina de la ingeniería, nuestras elecciones de diseño son un compromiso conformado por muchos factores que compiten. La identificación de las clases y objetos es la parte más difícil del diseño orientado a objetos. La experiencia muestra que la identificación implica descubrimiento e invención. Mediante el descubrimiento, se llega a reconocer las abstracciones clave y los mecanismos que forma el vocabulario del dominio del problema. Mediante la invención, se idean abstracciones generalizadas. Cuando se clasifica, se persigue agrupar las cosas que tienen una estructura común o exhiben un comportamiento común. La clasificación ayuda a identificar jerarquías de **generalización**, **especialización** y **agregación** entre clases.

Ejemplo de clasificación:

Se ha visto que un objeto es algo que tiene una frontera definida con nitidez. Sin embargo, las fronteras que distinguen un objeto de otro, son a menudo difusas. Por ejemplo, si se fijan en su pierna, ¿dónde empieza la rodilla y donde termina? En el reconocimiento del habla humana, ¿cómo saber que ciertos sonidos se conectan para formar una palabra y no son en realidad parte de otras palabras circundantes? En un procesador de texto, ¿constituyen los caracteres una clase o son las palabras completas una mejor elección? ¿Cómo tratar selecciones arbitrarias no contiguas de texto, y qué hay sobre las oraciones, párrafos o incluso documentos completos: son relevantes para el problema estas clases de objetos?

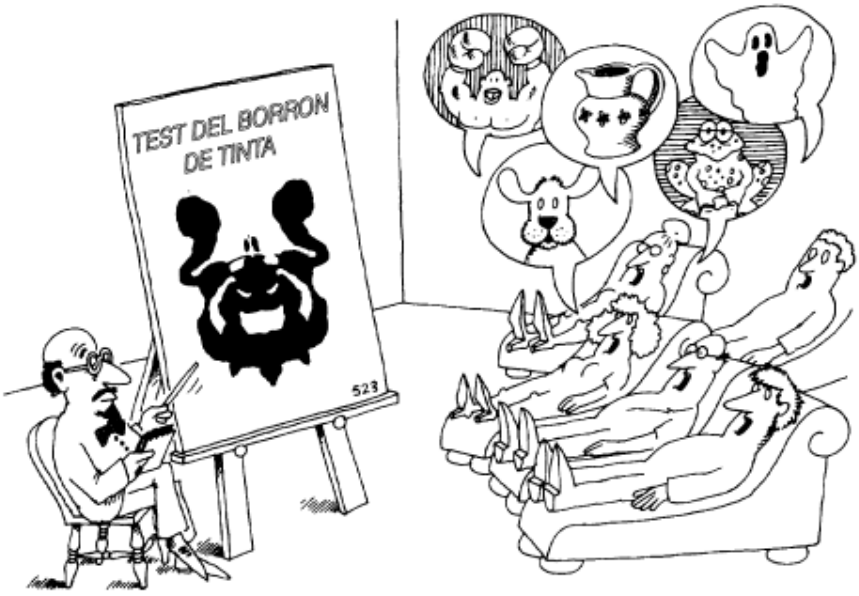
La clasificación es difícil puesto que existen paralelismos con los mismos problemas en el diseño orientado de objetos. Considérese por ejemplo la biología. Hasta el siglo XVIII

la opinión científica más extendida era que todos los organismos vivos podían clasificarse del más simple hasta el más complejo, siendo la medida de la complejidad algo muy subjetivo (¿por ejemplo, por qué será que los humanos fuesen situados en lo más alto de esta lista?). A mediados del mismo siglo, el botánico sueco *Carl Von Linneo* sugirió una taxonomía más detallada para categorizar los organismos de acuerdo con lo que se llama *géneros y especies*. Un siglo más tarde, *Darwin* propuso la teoría de que la selección natural fue el mecanismo de la evolución en virtud de la cual las especies actuales evolucionaron de otras más antiguas. La teoría de *Darwin* dependía de una clasificación inteligente de las especies. Como el propio *Darwin* afirma, los naturalistas *intentan disponer las especies, géneros y familias en cada clase en lo que se denomina el sistema natural*. Algunos autores lo toman a este sistema como un mero esquema para poner juntos aquellos objetos vivos que son más parecidos, y para separar los que son más distintos. En la biología actual, la clasificación denota el establecimiento de un sistema jerárquico de categorías sobre las bases de presuntas relaciones naturales entre organismos. La categoría más general en una taxonomía biológica es el reino, seguido en orden de especialización creciente por el filum, subfilum, clase, orden, familia, género y finalmente, especie. Históricamente un organismo concreto se sitúa en una categoría específica de acuerdo con su estructura corporal, características estructurales internas y relaciones evolutivas. Más recientemente, la clasificación se ha enfocado como la agrupación de organismos que comparten una herencia genética común: los organismos que tienen ADN similar se incluyen en el mismo grupo. La clasificación por ADN es útil para distinguir organismos que son estructuralmente similares, pero genéticamente muy diferentes. Para un informático, la biología parece una disciplina muy madura y con criterios bien definidos para clasificar organismos. Pero esto no es así. Como indican biólogos actuales, a nivel puramente de hechos, no se sabe ni siquiera dentro de un orden de magnitud cuántas especies de plantas y animales existen en el planeta; actualmente se han clasificado menos de 2 millones y las estimaciones del número total varían entre menos de 5 a 50 millones. Además, criterios diferentes para clasificar los mismos organismos arrojan resultados distintos. En definitiva, *todo depende de para qué quiere uno la clasificación*. Si se desea reflejar con precisión las relaciones genéticas entre las especies, ofrecerá una respuesta, pero si en vez de eso, se quiere decir algo sobre niveles de adaptación, la repuesta será distinta. La conclusión es que incluso en disciplinas rigurosamente científicas, la clasificación es altamente **dependiente de la razón por la que se clasifica**.

La clasificación inteligente es un trabajo intelectualmente difícil y la mejor forma de realizarlo es mediante un proceso incremental e iterativo. Primero los problemas se resuelven *ad hoc*. A medida que se acumula la experiencia, se va viendo que algunas soluciones funcionan mejor que otras y se transfiere informalmente una especie de folklore de persona a persona. Eventualmente las soluciones útiles se comprenden de forma más sistemática y se codifican y analizan. Esto permite el desarrollo de modelos que admiten una implantación automática y de teorías que permiten generalizar la solución. Esto a su vez da lugar a un nivel de práctica más sofisticado y nos permite atacar problemas más difíciles a los que con frecuencia se brinda un enfoque *ad hoc* comenzando el ciclo de nuevo. La naturaleza incremental e iterativa de la clasificación tiene un impacto directo en la construcción de jerarquías de clases y objetos en el diseño de un sistema de software complejo. En la práctica, es común establecer una determinada estructura de clases en fases tempranas del diseño y revisar entonces esa estructura a lo largo del tiempo. Sólo en etapas más avanzadas del diseño, una vez que se han construido los clientes que utilizan tal estructura se puede evaluar de forma significativa la calidad de la clasificación. Sobre la base de esta experiencia se puede decidir crear nuevas subclases a partir de otras existentes (derivación) o se puede dividir una clase grande en varias más pequeñas (factorización) o crear una clase mayor

uniendo otras más pequeñas (composición). Ocasionalmente se puede incluso descubrir aspectos comunes que habían pasado desapercibidos e idear una nueva clase (abstracción). Dos conclusiones:

- La clasificación es relativa a la perspectiva del observador que la realiza.
- La clasificación inteligente requiere una tremenda cantidad de perspicacia creativa.



Diferentes observadores pueden clasificar el mismo objeto de distintas formas.

Métodos para la clasificación de objetos y clases

Históricamente han existido tres aproximaciones generales a la clasificación:

- Categorización clásica
- Agrupamiento conceptual
- Teoría de prototipos

1. Categorización clásica

En la aproximación clásica a la categorización, todas las entidades que tienen una determinada propiedad o colección de propiedades en común forman una categoría. Tales propiedades son necesarias y suficientes para definir la categoría. Por ejemplo, las personas casadas constituyen una categoría, *se está casado o no*, y el valor de esta propiedad es suficiente para decidir a qué grupo pertenece un individuo. Por otra parte, las personas altas no forman una categoría, al menos que se determine un criterio absoluto por el que se distinga la propiedad *alto* de la propiedad *bajo*. Esta categorización emplea propiedades relacionadas como criterio de similitud entre objetos. Concretamente se puede dividir los objetos en conjuntos disjuntos dependiendo de la presencia o ausencia de una propiedad particular. En sentido general,

las propiedades pueden denotar algo más que meras características medibles, puede también abarcar ***comportamientos*** observables (hay animales que pueden volar y otros que no, y esta propiedad los puede distinguir). Las propiedades particulares que habría que considerar en una situación dada dependen mucho del dominio, por ejemplo, el color de un auto puede ser importante para el propósito de un control de inventario en una fábrica, pero no es relevante en absoluto para el software que controla los semáforos en una ciudad. Esta es la razón por la que se dice que no hay medidas absolutas para la clasificación aunque una estructura de clases determinada puede ser más adecuada para una aplicación que para otra. Algunas clasificaciones pueden verse mejor que otras, pero sólo respecto a nuestros intereses, no porque representan la realidad de forma más exacta o adecuada. De todas maneras parece prácticamente imposible proponer una lista de propiedades para cualquier categoría natural que excluya a todos los ejemplos que no están en la categoría e incluya a todos los que si están (p.e. la mayoría de los pájaros vuelan pero algunos no lo hacen pero siguen siendo pájaros – *¿o no?*). Estos son realmente problemas fundamentales de la categorización clásica, que el agrupamiento conceptual y la teoría de prototipos intentan resolver.

2. Agrupamiento conceptual

Es una variación más moderna del anterior y deriva en gran medida de los intentos de explicar cómo se representa el conocimiento. En este enfoque, las clases (agrupaciones de entidades) se generan formulando primero descripciones conceptuales de estas clases y clasificando entonces las entidades de acuerdo con las descripciones. Por ejemplo, se puede establecer un concepto como ***una canción de amor***. Éste es más bien un concepto que una propiedad porque la ***cantidad de amor*** de cualquier canción no es algo que se pueda medir empíricamente. Sin embargo, si se decide que cierta canción tiene más de canción de amor que de otra cosa, se la coloca en ésta categoría. Así, el agrupamiento conceptual representa más bien un agrupamiento probabilístico de los objetos.

3. Teoría de prototipos

La categorización clásica y el agrupamiento conceptual son suficientemente expresivos para utilizarse en la mayoría de las clasificaciones que se necesita en el diseño de sistemas de software complejos; sin embargo existen algunas situaciones en las que no resultan adecuadas. Existen algunas abstracciones que no tienen ni propiedades ni conceptos delimitados con claridad. Una categoría como un juego, no encajan en el molde clásico porque no hay propiedades comunes compartidas por todos los juegos. Aunque no hay una sola colección de propiedades que comparten todos los juegos, la categoría de los juegos está unida por *"parecidos familiares"*. Se puede observar que no hay fronteras fijas para la categoría juego. La categoría puede extenderse, pueden introducirse nuevos tipos de juegos siempre que se pareciesen a juegos anteriores de forma apropiada. Ésta es la razón por la que el enfoque se denomina *teoría de prototipos*: una clase de objetos se representa por un objeto prototípico y se considera un objeto como un miembro de esta clase si y solo si se parece a este prototipo de forma significativa. Por ejemplo considérense sillas de comedor con almohadón, sillones de peluquero, sillas plásticas, son sillas no porque compartan algún conjunto de propiedades definito-

rias con el prototipo, sino más bien porque mantienen suficiente parecido familiar con el prototipo. No hace falta que exista un núcleo fijo de propiedades de sillas prototípicas que compartan la silla de comedor y la de peluquero sino que ambas son sillas porque – cada una a su manera – está lo bastante cerca del prototipo. Las propiedades de interacción son sobresalientes entre los tipos de propiedades que cuentan a la hora de determinar si hay suficiente parecido familiar.

Aplicación de las teorías

Se identifican las clases y objetos en primer lugar de acuerdo con las propiedades relevantes para nuestro dominio particular. Aquí se hace hincapié en la identificación de las estructuras y comportamiento que son parte del vocabulario del domino del problema. Si este enfoque fracasa en la producción de una estructura de clases satisfactoria, hay que pasar a considerar la agrupación de objetos por conceptos. Aquí se concentra la atención en el comportamiento de objetos que colaboran. Si ambos intentos fallan al capturar nuestra comprensión del dominio del problema, entonces se considera la clasificación por asociación a través de la cual las agrupaciones de objetos se definen según el grado en el que cada una se parece a algún objeto prototípico.

6.2 DIAGRAMAS DEL UML

El UML documenta los sistemas de software que modela desde dos puntos de vista básicos: dinámico y estático. El modelado dinámico se refiere al comportamiento del sistema en tiempo de ejecución, en tanto que el modelado estático a la descripción de los componentes del sistema así como de sus relaciones. Esto no significa que los modelos deban estar separados en grupos diferenciados; lo que se debe hacer es presentar los sistemas como una colección de distintos puntos de vista (proporcionado por cada tipo de diagrama).

Los diagramas estructurales presentan elementos estáticos del modelo, tales como clases, paquetes o componentes; en tanto que los diagramas de comportamiento muestran la conducta en tiempo de ejecución del sistema, tanto visto como un todo como de las instancias u objetos que lo integran.

Hay que tener en cuenta que cada diagrama sirve para documentar un aspecto distinto del sistema; el criterio para usarlos es el de tener algo que decir, una historia sobre el sistema que se estudia y que debe ser contada; el tipo de diagrama que utilizará será el que dé mayor poder expresivo y la construcción de una serie de diagramas puede ser ordenada por un método de desarrollo utilizado. Algunos sistemas simples serán bien documentados con pocos diagramas, en tanto que algunos sistemas grandes bien podrían beneficiarse de un conjunto mayor. No se hacen diagramas porque un método lo exija sino que se hacen para contar algo importante del modelo, por lo que indicar que en un sistema haya que hacer tantos de tal o cual diagrama es una declaración sin fundamento alguno. Si bien existen varios tipos de diagramas para cada perspectiva, en la materia se verán solamente los siguientes:

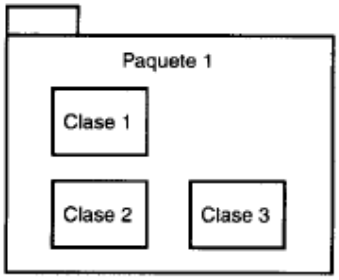
- Estáticos o estructurales:
    - Diagrama de clases

- Dinámicos o de comportamiento:
    - Diagrama de casos de uso
    - Diagrama de transición de estados

Elementos genéricos del UML

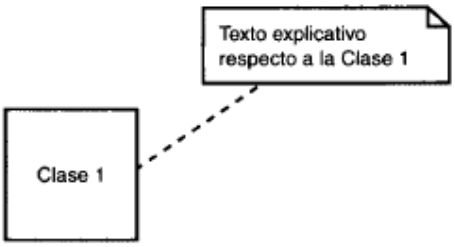
1. Los paquetes

Una de las características de UML es que comúnmente se requiere organizar los elementos de un diagrama dentro de un grupo. Tal vez se desee mostrar ciertas clases o componentes que son parte de un subsistema en particular. Para ello se agruparán en un ***Paquete*** el que se representa por una carpeta tabular como la siguiente:



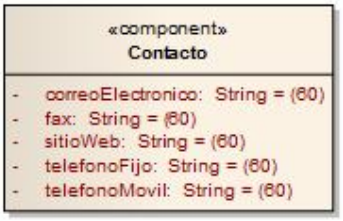
2. Las notas

Es frecuente también que en alguna parte del diagrama no se presenta una clara explicación del porqué algo está en ese lugar o la manera en que trabaja. En este caso se utiliza la ***nota*** UML. Es similar a un sticker pegado al gráfico donde se coloca la explicación. Se adjunta al elemento del diagrama conectándolo con una línea discontinua:



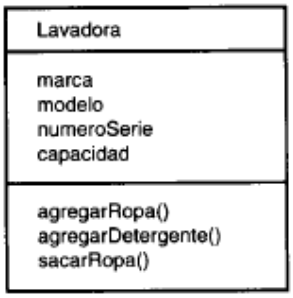
3. Los estereotipos

El UML brinda muchos elementos pero no un conjunto minucioso. A veces se requieren elemento hechos a medida. Los ***estereotipos*** permiten tomar elementos propios del UML y convertirlos en otros. Se representa como un nombre entre dos pares de paréntesis angulares. Su aplicación se verá posteriormente. Por ejemplo:



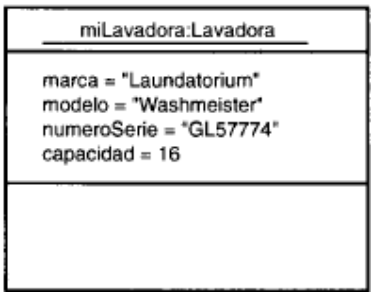
6.2.1 DIAGRAMAS DE CLASES

Supóngase la clase lavadora de la que se conoce la marca, el modelo, el número de serie y la capacidad junto con las acciones de agregar ropa, agregar detergente y sacar ropa. Seguramente también deberá tener un mecanismo para **fabricar** nuevas instancias a partir de ella, es decir, podrá crear nuevos objetos. La figura siguiente muestra la notación de UML que captura los atributos y acciones de una lavadora. Un rectángulo es el símbolo que representa a la clase y se divide en tres áreas. El área superior contiene el nombre, el área central los atributos y la inferior las acciones. Como convención, el nombre de la clase, como lavadora se escribe **Lavadora** y si constara de dos palabras se escribiría como **LavadoraIndustrial** y las características como número de serie se escribirán como **numeroSerie**. De igual manera se procede con las etiquetas de los métodos. De esta manera, la representación de la clase será:



El propósito de la orientación a objeto es desarrollar software que modele un esquema real del mundo. Entre más atributos y acciones se tomen en cuenta, mayor será la similitud del modelo con la realidad. Cabe entonces al diseñador, saber hasta dónde quiere llegar con la profundidad y que ésta sea relevante para la realidad que quiere modelar.

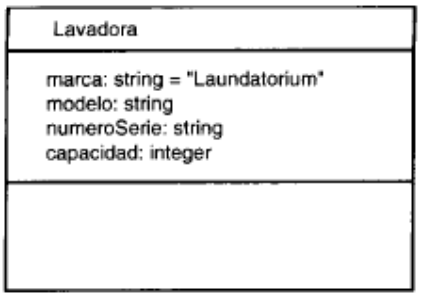
La instanciación de una clase es un objeto, entonces, considerando el ejemplo anterior:



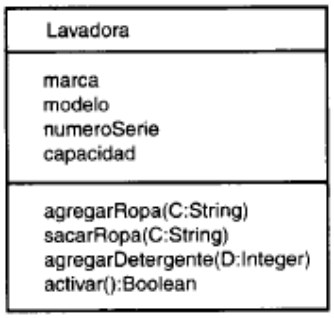
Obsérvese que el nombre de un objeto inicia con una letra minúscula. Se indica el nombre del objeto, dos puntos y luego el nombre de la clase a la que pertenece.

En las clases puede mostrarse además del tipo de dato de cada atributo, el valor por defecto que puede asignársele. Entre los posibles tipos se encuentran las cadenas (string), número de punto flotante (float), entero (integer), entero corto (short), y boolean, además de

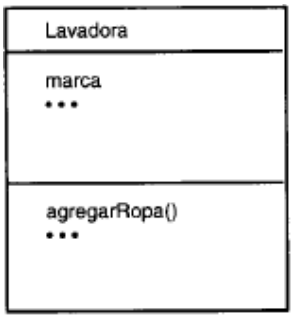
los tipos enumerados. Para indicar un tipo, se utilizan dos puntos para separar el nombre del atributo y su tipo. Por ejemplo:



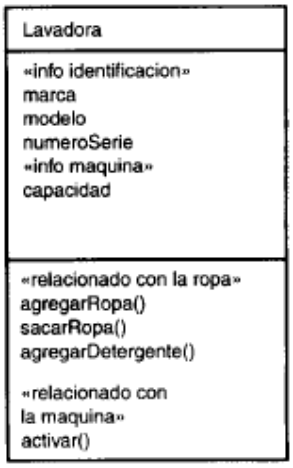
Así como es posible representar información adicional de los atributos, también se puede con las operaciones o métodos. En los paréntesis que preceden al nombre se puede mostrar el parámetro con el que funcionará la operación junto con su tipo de dato. La función devuelve un valor luego que finaliza su trabajo. En una función puede mostrarse el tipo de valor que regresará. Por ejemplo:



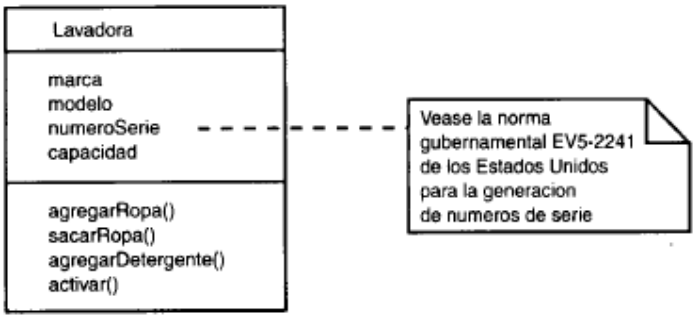
En los diagramas se mostrará más de una clase a la vez por lo que no resulta muy útil que siempre aparezcan todos los atributos y operaciones para no saturarlos demasiado. En lugar de ello se puede mostrar el nombre de la clase y dejar el área de atributos y/o de las operaciones vacías. En ocasiones es bueno mostrar algunos y en este caso seguirá a la lista de los mostrados un renglón con puntos suspensivos:



Si se tiene una larga lista de atributos u operaciones, se puede utilizar un estereotipo para organizarlas de forma que sea más comprensible. Un estereotipo es el modo en que el UML le permite extenderlo es decir, crear nuevos elemento que son específicos de un problema en particular que intente resolver. Los estereotipos se muestran con un nombre entre paréntesis angulares. Para una lista de atributos se puede utilizar un estereotipo como encabezado:



Pueden además agregarse mayor información en una clase mediante las notas adjuntas. Esta nota puede contener tanto una imagen como texto.



Construcción de los diagramas de clases

Las clases representan el vocabulario de un área de conocimiento. Las conversaciones con el cliente o un experto en el área dejarán entrever los sustantivos que se convertirán en clases (o atributos de una clase) en un modelo y los verbos se transformarán en operaciones. Se pueden utilizar diagramas de clase como una forma de estimular al cliente a que diga más respecto a su área y que ponga en evidencia cierta información adicional. Nótese que la manera en que se identifican las clases es similar a la utilizada para encontrar las entidades y sus atributos en el modelo conceptual de datos. Esto es válido para las asociaciones que se presentan entre las clases.

Sin las relaciones, un modelo de clases sería poco menos que una lista de cosas que representarían un vocabulario. Las relaciones muestran cómo se conectan los términos del vocabulario entre sí para dar una idea de lo que se está modelando. La asociación es la conexión conceptual fundamental entre clases. Cada clase en una asociación juega un papel y la multiplicidad muestra cuántos objetos de una clase se relacionan con un objeto de la clase asociada. Una asociación se representa como una línea entre los rectángulos de las clases con los roles y multiplicidades (cardinalidades). Una clase puede heredar atributos y operaciones de otra clase. La clase heredada es secundaria de la clase principal que es de la que se hereda. Se descubre la herencia cuando se encuentran clases en el modelo inicial que

tengan atributos y operaciones en común. Las clases abstractas sólo se proyectan como bases de herencia y no proporcionan objetos por sí mismas.

6.2.2 DIAGRAMAS DE CASOS DE USO

6.2.2.1 Conceptos

Las ideas estáticas ayudan a que el analista se comunique con el cliente, mientras que la dinámica ayuda a la relación con el grupo de desarrolladores. Tanto cliente como equipo de desarrollo es un conjunto importantísimo en el sistema, no obstante falta **el usuario**. Ni la idea estática ni la dinámica muestran el comportamiento del sistema desde el punto de vista del usuario siendo que es un punto clave. El modelado del sistema desde el punto de vista del usuario es el trabajo de los casos de uso. El caso de uso ayuda a los analistas a determinar la forma en que se utilizará un sistema. El caso de uso es como una colección de situaciones respecto al uso de un sistema. Cada escenario describe una secuencia de eventos. Cada secuencia se inicia por una persona, otro sistema, una parte del hardware o por el paso del tiempo. A tales entidades se las conoce como **Actores**. El resultado de la secuencia debe ser algo utilizable ya sea por el actor que la inició o por otro actor.

La visualización de los casos de uso permiten que el usuario al verlos pueda dar más información. Es un hecho que los usuarios saben más de los que dicen y este tipo de diagrama colabora en la comunicación. Además, la representación visual ayuda a combinar los diagramas de casos de uso con otros de otro tipo.

Para su representación gráfica, hay un actor que inicia el caso de uso y otro (posiblemente el mismo) que recibirá algo de valor de él. La representación gráfica es directa. Una elipse representa el caso de uso y en su interior se registra su nombre; una figura esquemática de una persona representa al actor y debajo de él su nombre. El actor que inicia se encuentra a la izquierda y el que recibe a la derecha. Una línea asociativa conecta a un actor con el caso de uso y representa la comunicación entre ellos. Uno de los beneficios del caso de uso es que muestra los confines entre el sistema y el mundo exterior. Generalmente los actores están fuera del sistema mientras que los casos de uso están dentro de él. Se utiliza un rectángulo con el nombre del sistema en algún lugar de su interior para representar las fronteras y este rectángulo tendrá en su interior los casos de uso definidos.

Suponga que se quiere hacer el diseño de la máquina expendedora de gaseosas. Para obtener el punto de vista del usuario final se entrevistarán usuarios potenciales respecto de la manera en que utilizarán la máquina. Cada entrevistado puede que le presente distintos escenarios para el mismo caso de uso. El caso de uso inicial será ***Comprar gaseosa***. El actor será un cliente que desea comprar una gaseosa. El escenario iniciará cuando el cliente inserte el dinero, posteriormente realizará una selección de cual gaseosa quiere y luego la máquina (suponiendo que tenga al menos una de las solicitadas) pondrá a alcance del cliente la gaseosa elegida. Además de la secuencia, hay otros aspectos del escenario que merecen

cierta consideración como la *precondiciones* y las *poscondiciones*. Para el ejemplo se tendrá: Precondición: *¿qué condiciones llevaron al cliente a iniciar el escenario en el caso de uso* ***Comprar gaseosa****?: La sed es la más obvia*. Poscondición: *¿qué se obtiene como resultado?: Cliente con la gaseosa en su poder*.

Cada caso de uso es una colección de escenarios y cada escenario es una secuencia de pasos. Estos pasos no aparecen en el diagrama. No se encuentran en notas adjuntas tampoco. Aunque nada lo prohíbe es mejor que los diagramas sean claros. Las secuencias de pasos se obtienen de documentación asociada al caso de uso que cliente y equipo de desarrollo tomarán como referencia. Cada caso de uso tiene su propio documento, de igual manera que cada escenario de caso de uso lo tendrá. La información que se requiere será:

- Actor que inicia el caso de uso
- Precondiciones
- Pasos en el escenario
- Poscondiciones
- Actor que se beneficia del caso de uso

También pueden enumerarse conjeturas del escenario (por ejemplo, que un cliente a la vez utilizará la máquina de gaseosas) y una breve descripción de una sola frase del escenario.

Supóngase ahora que la máquina no tenga gaseosas o bien que el cliente no tenga el importe exacto. Tales casos el diseño de la máquina deberá contemplarlos.

Para el caso en que la máquina se haya quedado sin gaseosa, existirá una ruta alternativa dentro del caso de uso ***Comprar gaseosa***. El cliente inicia el caso de uso al insertar dinero en la máquina y posteriormente hace una selección. La máquina no cuenta con ninguna lata de la gaseosa seleccionada por lo que mostrará un mensaje al cliente que indicará que no tiene de esa marca. Lo ideal podría ser que el mensaje le pida al cliente que haga otra selección. La máquina también debería dar la opción de devolver el dinero al cliente. La condición previa es un cliente con sed y el resultado es una lata de gaseosa o la devolución del dinero.

Para el caso que el cliente no tiene el importe exacto también existirá una ruta alternativa. El cliente inicia el caso de uso insertando el dinero y luego hace una selección. Se sume que la máquina tiene provisión de la marca elegida. En la máquina hay una reserva de moneda fraccionaria y devuelve la diferencia al despachar la lata. Si la máquina no cuenta con dinero para el vuelto, devolverá el dinero original y mostrará un mensaje que pida al usuario el importe exacto. La precondición sigue siendo la misma y la poscondición la lata de gaseosa y el cambio o bien el importe originalmente depositado.

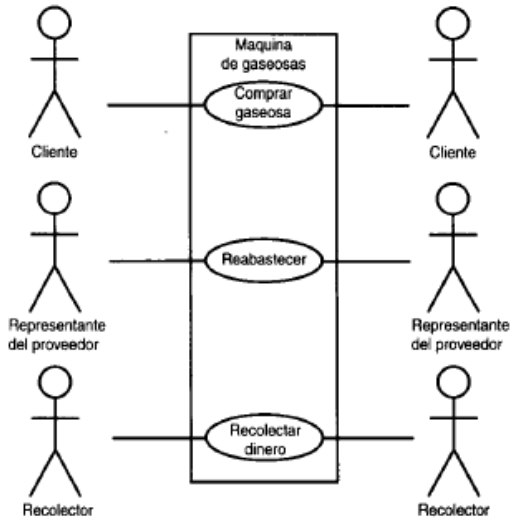
Existen otros usuarios que interactúan con la máquina y estos pueden ser el proveedor que tiene que reabastecer el stock en la máquina, el recolector de dinero, etc. Esto indica que hay que crear otros casos de uso para estos nuevos actores y que podrían ser ***Reabastecer*** y ***Recolectar dinero*** cuyos detalles surgirán durante las entrevistas con los proveedores y los recolectores.

Para el caso de uso ***Reabastecer*** el proveedor (o su representante) lo inicia por su decisión o bien porque habría pasado cierto lapso de tiempo de la última vez que lo hizo. El

representante del proveedor le quita el seguro a la máquina (mediante una llave y un cerrojo, pero eso entra dentro de la implementación), abre la puerta de la máquina y llena el compartimiento de cada marca hasta su capacidad. El representante también llena la reserva de moneda fraccionaria. Luego, cierra el frente de la máquina y vuelve a poner el seguro. La condición previa es la necesidad de reponer unidades y/o el intervalo de tiempo, y el resultado o poscondición es que el proveedor cuenta con nuevo conjunto de ventas potenciales.

Para el caso de ***Recolectar dinero*** el recolector inicia por su decisión o bien porque ha pasado cierto lapso de tiempo desde la última vez que hizo la última recolección. La persona deberá seguir la misma secuencia que en ***Reabastecer*** para abrir la máquina. El recolector sacará el dinero de la máquina y seguirá los pasos de ***Reabastecer*** para cerrar y poner el seguro a la máquina. La condición previa es la necesidad de retirar dinero y/o el intervalo de tiempo y el resultado es el dinero en las manos del recolector.

Considerando los casos de uso definidos y los actores asociados, el siguiente modelo representa tales integrantes:



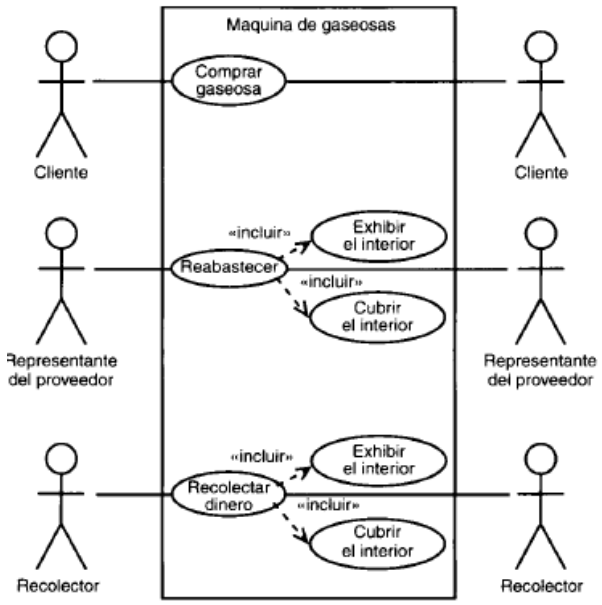
No se tienen en cuenta componentes internos de la máquina, mecanismos, controladores de cambio, etc. sino que lo que se intenta es ver la forma en que la máquina se comporta para con quien tenga que utilizarla.

El objetivo es obtener una colección de casos de uso que finalmente se mostrarán a las personas que diseñen las máquinas de gaseosas y a las personas que las construirán. Además, estos casos de uso reflejan lo que los clientes, recolectores y proveedores desean por lo que el resultado será una máquina que todos esos grupos puedan utilizar con facilidad.

6.2.2.2 Inclusión y extensión de los casos de uso

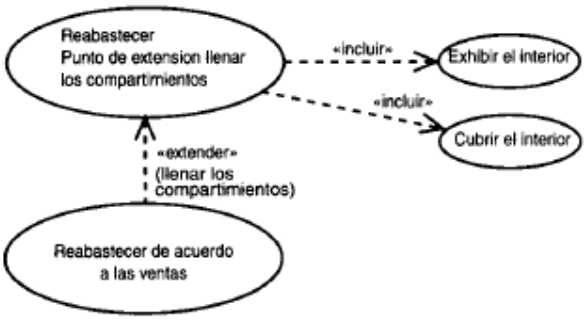
En los caso de uso ***Reabastecer*** y ***Recolectar dinero*** se observa que hay ciertos pasos en común. Ambos comienzan con abrir la máquina y finalizan con el cierre y su aseguramiento. En este caso se puede entonces eliminar esa duplicación de pasos para

ambos casos. La forma de hacerlo es tomar cada secuencia de pasos en común y construir un caso de uso adicional a partir de ellos. Se combinarán los pasos necesarios para *quitar el seguro* y *abrir la máquina* denominándolos ahora ***Exhibir el interior*** y los pasos *cerrar la máquina* y *asegurarla* en otro caso de uso llamado ***Cubrir el interior***. Con estos nuevos casos de uso, el caso de uso ***Reabastecer*** iniciaría con el caso de uso ***Exhibir el interior***. Luego el representante del proveedor seguiría los pasos ya indicados y concluiría con el caso de uso ***Cubrir el interior***. De forma similar, el caso de uso ***Recolectar dinero*** iniciaría con ***Exhibir el interior***, procedería como se indicó y finalizaría con el caso de uso ***Cubrir el interior***. Como se ve, ***Reabastecer*** y ***Recolectar dinero*** incluyen los nuevos casos de uso definidos.



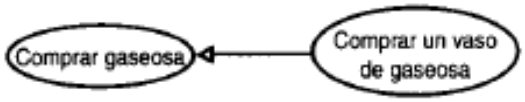
Es posible también utilizar un caso de uso de una forma distinta a la inclusión. En ocasiones se crea un caso de uso agregándole algunos pasos a un caso de uso existente. El caso de uso ***Reabastecer*** antes de colocar nuevas latas de gaseosas en la máquina, el representante del proveedor puede notar que hay marcas que se han vendido bien, así como que otras no lo han hecho. En lugar de solamente reabastecer todas las marcas, el representante podría sacar aquellas que no se han vendido bien y reemplazarlas por las que hay probado ser más populares. De esta forma tendría que indicar al frente de la máquina el nuevo surtido de marcas disponibles. Si se agregan estos pasos a ***Reabastecer***, se está en presencia de un nuevo caso de uso que podría llamarse ***Reabastecer de acuerdo a las ventas***. Este nuevo caso de uso es una extensión del original y se dice entonces que el nuevo caso de uso *extiende* al caso de uso base. La extensión sólo se puede realizar en puntos indicados de manera específica dentro de la secuencia del caso de uso base. A estos puntos se les conoce como puntos de extensión. En el caso de uso ***Reabastecer***, los nuevos pasos (anotar las ventas y abastecer de manera acorde) se darían luego que el representante haya abierto la máquina y esté listo para llenar los compartimientos de las marcas de gaseosas. En este ejemplo, el punto de extensión es *llenar los compartimientos.*

Para denotar la inclusión y la extensión se utilizan líneas de dependencia en donde se deberá mostrar el estereotipo **extender** o **incluir**.

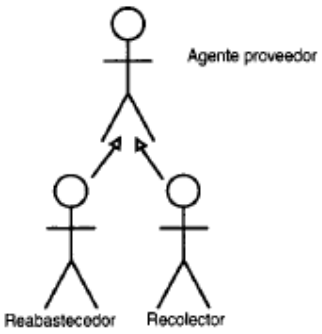


6.2.2.3 Generalización

Las clases puede heredarse entre sí, y esto también se aplica a los casos de uso. En la herencia de los casos de uso, el caso de uso secundario hereda las acciones y significado del primario, y además agrega sus propias acciones. Puede aplicar el caso de uso secundario en cualquier lugar donde se aplique el primario. En el ejemplo puede agregarse el caso de uso ***Comprar vaso de gaseosa*** que se puede heredar de ***Comprar gaseosa***. El caso de uso secundario tiene acciones como *obtener el vaso*, *agregar hielo* y *mezclar* además de las heredadas. La manera de representar la herencia es similar a lo expresado en las clases.



La relación de generalización también puede establecerse entre actores, así como entre casos de uso. Tanto el *Reabastecedor* como el *Recolector* son *Representantes del proveedor*; si se cambia el nombre de *Representante de proveedor* por ***Reabastecedor***, tanto éste como el ***Recolector*** serán secundarios:



6.2.2.4 Diagramas de caso de uso en el proceso de análisis

Las entrevistas al cliente deben iniciar el proceso. Estas entrevistas producirán diagramas de clases que serán como las bases del conocimiento para el dominio del sistema. Una vez que se conozca la terminología general del área del cliente se estará listo para hablar con los usuarios.

Las entrevistas con los usuarios comienzan en la terminología del dominio aunque deberán alternarse hacia la terminología de los usuarios. Los resultados iniciales de las

entrevistas deberán revelar a los actores y casos de uso de alto nivel que describirán los requerimientos funcionales en términos generales. Esta información establece las fronteras del sistema.

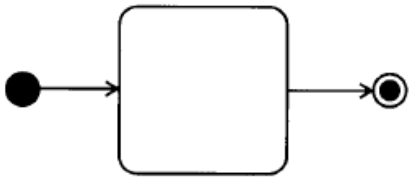
Las entrevistas posteriores con los usuarios profundizarán en estos requerimientos lo que dará por resultado modelos de casos de uso que mostrarán los escenarios y las secuencias detalladamente. Esto podría resultar en otros casos de uso que satisfagan las relaciones de inclusión y extensión. En esta fase, es importante haber logrado una gran comprensión del dominio ya que de lo contrario, podrían crearse demasiados casos de uso y demasiados detalles sin sentido.

6.2.3 DIAGRAMAS DE ESTADOS

Un diagrama de estados es una herramienta que permite caracterizar un cambio en un sistema o sea que los objetos que lo componen modificaron su ***estado*** como respuesta a los sucesos y al tiempo. Estos diagramas presentan los estados en los que puede encontrarse un objeto en un momento determinado junto con las transiciones entre estados, mostrándose además los puntos inicial y final de una secuencia de cambios de estado. También es conocido como **máquina de estados**.

Un diagrama de estados es totalmente distinto de uno de clase o de caso de uso. Los diagramas ya vistos modelan las clases, o comportamientos desde el punto de vista del usuario, mientras que un diagrama de estados **muestra las condiciones de un solo objeto**.

La simbología utilizada es la siguiente:



Se utiliza un rectángulo con los vértices redondeados para representar un estado, junto con una línea continua y una punta de flecha las que representan una transición. La punta de flecha apunta hacia el estado donde se hará la transición. La figura también muestra un círculo relleno que simboliza un punto inicial y la diana que representa un punto final.

Adicionalmente pueden agregarse detalles a la simbología. De la misma manera que se dividen las áreas de una clase (nombre, atributos y operaciones), puede dividirse el ícono de estado. En la parte superior irá el nombre, en el medio las variables de estado y la inferior las actividades.

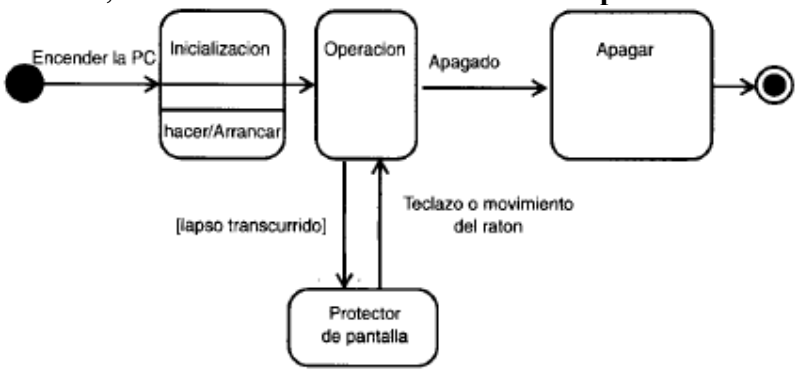
Las variables de estado no aportan demasiado. Las actividades constan de sucesos y acciones; las tres más utilizadas son ***entrada*** (qué sucede cuando el sistema entra al estado), ***salida*** (qué sucede cuando el sistema sale del estado) y ***hacer*** (qué sucede cuando el sistema está en el estado).

En el diagrama también se pueden agregar ciertos detalles a las líneas de transición. Puede indicar un suceso que provoque una transición y la acción que se ejecute. Los sucesos y las acciones se escriben cerca de la línea de transición separados por una diagonal (*suceso / acción*). La interfaz gráfica de usuario (GUI) con que se interactúa en una computadora, da ejemplos de detalles de la transición. Se asume en principio que la GUI puede tener tres estados:

- Inicialización
- Operación
- Apagar

Cuando se enciende el equipo (*Encender la PC*) se ejecuta un proceso de arranque y se desencadena un suceso que provoca que le GUI aparezca luego de una transición desde el estado de **Inicialización** y el arranque es una acción que se realiza durante tal transición. Como resultado de las actividades en el estado de inicialización la GUI entra al modo de **Operación**. Cuando se desea apagar la PC, se desencadena un suceso que provoca la transición hacia el estado de **Apagado** y con ello la PC se apaga.

Para completarlo se agregará que si se deja solo el equipo o si se realizara alguna actividad que no toque el ratón o el teclado aparecerá un protector de pantallas. Para decirlo en términos de cambio de estados sería que si ha pasado cierto tiempo sin que haya interacción con el usuario, la GUI hará una transición del estado **Operación** al uno nuevo.



El intervalo se especifica con algún panel de control del sistema operativo. Cualquier movimiento del ratón o presión sobre alguna tecla provocará la transición del estado **Protector de pantalla** al de **Operación**. Este intervalo constituye una *condición de seguridad* añadida.

Cuando la GUI está en estado de **Operación** muchas otras cosas ocurren aunque no sean evidentes en la pantalla. La GUI espera constantemente que alguien haga alguna

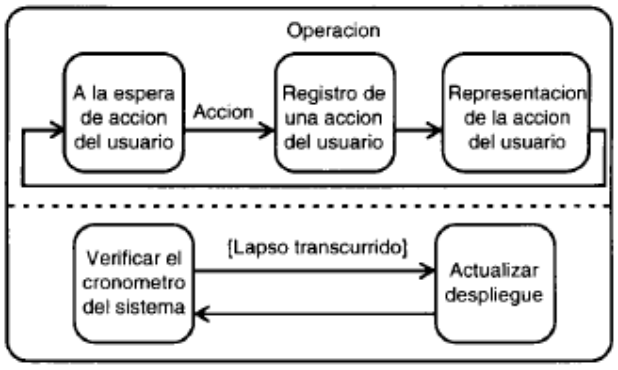
acción (presionar una tecla, accionar el ratón). Luego debe registrar esas acciones y modificar lo que despliega para reflejarlas en la pantalla (mover el puntero al mover el ratón o mostrar una letra **a** cuando se presiona la tecla **a**). De esta manera, la GUI atraviesa varios cambios de estado (subestados) mientras esté en **Operación**. Hay dos tipos de subestados: **secuenciales** y **concurrentes**.

Los **secuenciales** suceden uno detrás de otro, por ejemplo para el caso planteado, los subestados dentro del estado **Operación** podrían ser:

- A la espera de acción del usuario
- Registro de una acción del usuario
- Representación de la acción del usuario

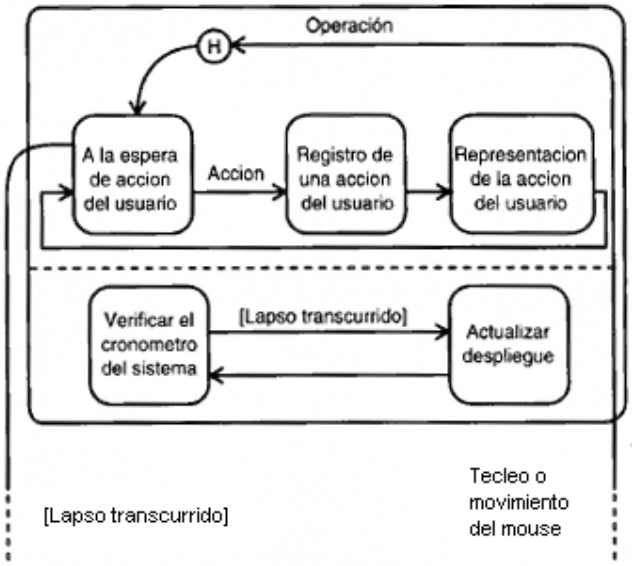
La acción del usuario desencadena la transición a partir de *A la espera de acción del usuario* hacia *Registro de una acción del usuario*. Las actividades dentro del *Registro* trascienden de la GUI hacia la *Representación de la acción del usuario*. Luego del tercer subestado, la GUI vuelve a iniciar *a la espera de acción del usuario*:

Los **concurrentes** suceden al mismo tiempo que los secuenciales. Para el ejemplo, la GUI no solamente aguarda a que se haga algo sino que también verifica el reloj del sistema y posiblemente actualice el despliegue de una aplicación luego de un intervalo específico (por ejemplo una aplicación que incluya un reloj en pantalla que tuviera que actualizar la GUI). Aunque cada secuencia es un conjunto de estados secuenciales, las dos secuencias son concurrentes entre sí.



Cuando se activa el protector de pantalla y luego se mueve el ratón para regresar al estado **Operación** la pantalla no debe retomar el estado inicial como si se hubiese encendido la PC, sino que tiene que mostrarse como se dejó antes que se activara el protector. Para ello, el diagrama de estados **histórico** captura esta idea. Existe un símbolo que muestra que un estado compuesto recuerda su subestado activo cuando el objeto trasciende fuera del

estado compuesto. El símbolo es la letra **H** encerrada en un círculo que se conecta por una línea continua con punta de flecha hacia el subestado a recordar.



Los diagramas de estados permiten a los desarrolladores comprender el comportamiento de los objetos de un sistema. Un diagrama de clases solamente muestra aspectos estáticos del sistema. Muestra jerarquías y asociaciones y le indican cuales son las operaciones pero no muestran ningún detalle dinámico de las operaciones. Los desarrolladores en particular, deben saber la forma en que los objetos se supone que se comportarán, ya que ellos son quienes tendrán que establecer tales comportamientos en el software. No es suficiente con implementar un objeto; los desarrolladores deben hacer que tal objeto haga algo. Los diagramas de estados se aseguran que no tendrán que adivinar lo que se supone qué harán los objetos. Con una clara representación del comportamiento del objeto, aumenta la probabilidad de que el equipo de desarrollo produzca un sistema que cumpla con los requerimientos.