Concepto del ciclo de vida de los sistemas y fases. Modelos de ciclo de vida.

Introducción

En este tema se va a dar una visión de lo que se llama ciclo de vida del software, así como distintos modelos de representación del mismo.

¿Para qué un ciclo de vida? En un departamento de sistemas de información de una empresa es necesario establecer lo que llamamos un marca de referencia común que pueda ser empleado por todas las personas que participan en el desarrollo de un proyecto informático: directivos, consultores externos e internos, jefes de proyecto, analistas, programadores, usuarios, etc.

En este marco de referencia deben estar claramente definidos los procesos, actividades y tareas a desarrollar.

Veamos primeramente dos definiciones publicadas por dos organismos internacionales:

- Norma IEEE 1074: Se entiende por ciclo de vida del software una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento software.
- Norma ISO 12207-1: Se entiende por modelo de ciclo de vida un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de requisitos hasta la finalización de su uso.

La Evolución del Software

El concepto de ciclo de vida se utilizó para modelar el proceso de ingeniería del software que, a su vez, apareció como solución a la llamada “crisis del software”.

Veamos un poco de historia. El desarrollo del software ha ido en paralelo con la evolución de los sistemas informáticos, los cuales han ido mejorando notablemente debido al aumento de la potencia del hardware, a la reducción de su tamaño y la disminución de su coste económico.

Siguiendo a Presmann podemos distinguir las siguientes etapas en la evolución del software:

- 1ª Etapa: Primeros años de la informática (1950-1965). El hardware sufrió numerosos cambios. Se desarrollaba software sin planificación y sin metodologías sistemáticas. En casi todos los sistemas se utilizaba programación en “batch”. La mayoría del software era desarrollado y utilizado por la misma persona.
- 2ª Etapa: (1965-1975). Aparición de la multiprogramación y de los sistemas multiusuarios. Como consecuencia de la interactividad de los sistemas aparecen nuevos tipos de aplicaciones. Surgen, asimismo, los sistemas de tiempo real. También los primeros gestores de BD.
- 3ª Etapa: (1975-1985). Aparecen los sistemas distribuidos, redes de área local LAN y de área global WAN. Hay una fuerte presión sobre los desarrolladores de software. Los ordenadores personales impulsan el crecimiento de muchas compañías de software.

- 4ª Etapa: (1985-). Tecnologías de cuarta generación. Tecnologías orientadas a objetos.

Características especiales del Software

Existen características propias del software que lo diferencian de otros productos:

- El software no se fabrica en un sentido clásico, sino que se desarrolla: Si bien existen similitudes con la fabricación del hardware, se trata de actividades fundamentalmente diferentes. Tanto en una como en otra la buena calidad se adquiere mediante un buen diseño pero la fabricación del hardware es muy diferente de la del software y puede introducir problemas de calidad que no existen o son fácilmente corregibles en el software. Ambas actividades requieren la construcción de un produce pero con métodos muy diferentes. Los costes del desarrollo del software están en la ingeniería por lo que no se pueden gestionar como si fueran clásicos proyectos de fabricación.
- El software no se “estropea”: El hardware se deteriora con el paso del tiempo y con el uso. Los errores no detectados del software provocarán fallos al principio de su vida. Sin embargo, una vez corregidos deberían desaparecer los fallos y no aparecer nuevos. No obstante la realidad suele ser diferente. El software sufre a lo largo de su vida modificaciones y al introducir estos cambios suelen producirse nuevos fallos que, a su vez, tienen que ser corregidos y así sucesivamente.
- La mayor parte del software se construye a medida, en lugar de ensamblar componentes como hace la industria: En la fabricación del hardware, la reutilización de componentes es una parte natural del proceso de ingeniería. En el mundo del software es algo que sólo ha comenzado a lograrse recientemente.

La “Crisis del Software”

Desde que se empezó a desarrollar software a gran escala empezaron a ser comunes una serie de problemas:

- La planificación resultaba ser muy imprecisa. Los plazos de entrega eran superados en la mayoría de los casos.
- El coste final de los proyectos era frecuentemente mucho mayor que el previsto.
- La productividad era muy baja.
- La calidad el producto entregado era, asimismo, muy baja.
- El cliente solía quedar insatisfecho del producto.
- El software era difícil de mantener.

Hay que añadir que el conjunto de problemas encontrados en el desarrollo del software no se limitan al software que “no funciona correctamente”. La llamada crisis abarca los problemas asociados a cómo desarrollar software, como mantener el volumen cada vez mayor de software existente y cómo poder mantenernos al corriente de la demanda creciente de software.

Ingeniería del Software

La ingeniería del Software surge de la necesidad de sistematizar el desarrollo del software afectado por la llamada “crisis del software”, aplicando principios de ingeniería para poder obtener software de calidad.

¿Qué es software de calidad? Si asumimos que el software cumple con la funcionalidad requerida, para que sea de calidad deberá tener las siguientes características:

- El software debe ser mantenible. Deberá estar escrito y documentado de forma tal que las modificaciones se puedan realizar con el menor coste posible. Esto es

fundamental ya que la mayor parte del coste asociado del software se produce después de su puesta en funcionamiento.
- El software debe ser fiable. Es decir, debe comportarse como esperan los usuarios y no debe fallar más de lo permitido por la especificación.
- El software debe ser eficiente.
- Debe ofrecer una interfaz de usuario apropiada.

Concepto del Ciclo de Vida y Fases

Podemos definir ciclo de vida de un sistema de información como el conjunto de etapas por las que atraviesa el sistema desde su concepción, hasta su retirada de servicio pasando por su desarrollo y explotación.

A veces también se habla de “ciclo de desarrollo”, que es un subconjunto del anterior que empieza en el análisis y finaliza con la entrega del sistema al usuario.

Existen diferentes modelos de ciclo de vida o sea distintas pautas a seguir en el desarrollo de los sistemas de información. Más adelante estudiaremos dos, el llamado modelo clásico o en cascada y el modelo en espiral.

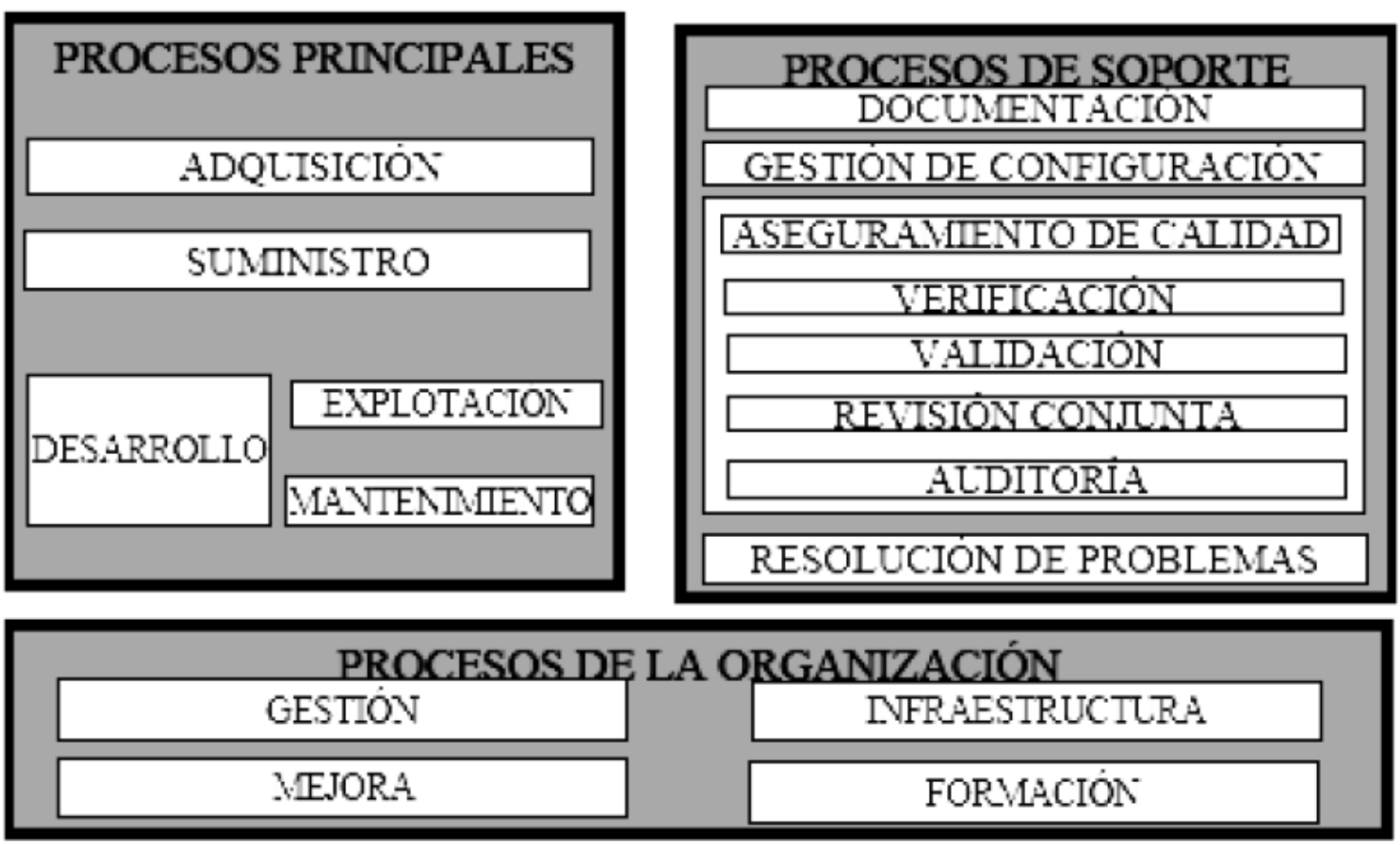
Tres son los objetivos básicos que cualquier modelo de ciclo de vida debe cubrir:

- Definir las actividades a realizar y su orden.
- Asegurar la consistencia con el resto de los sistemas de información de la organización.
- Proporcionar puntos de control para la gestión del proyecto (presupuesto y calendario).

Procesos del Ciclo de Vida Software

Según la Norma ISO 12207-1, las actividades a realizar durante el ciclo de vida del software se agrupan en cinco procesos principales, ocho procesos de soporte y cuatro procesos de la organización, así como un proceso especial que permite adaptar el ciclo de vida a cada proyecto en concreto.

A destacar que la norma no recomienda ningún modelo concreto de ciclo de vida, ni de gestión del software, ni detalla cómo realizar ninguna de las actividades.



Procesos Principales

Son aquellos que resultan útiles a las personas que inician o realizan el desarrollo, la explotación o el mantenimiento del software a lo largo del ciclo de vida. Estas personas son los compradores, los proveedores, el personal de desarrollo, los usuarios y el personal encargado del mantenimiento del software.

- **Proceso de adquisición**: Contiene las actividades y tareas que el comprador, el cliente o el usuario realizan para adquirir un sistema o un producto software. Aquí están incluidos la preparación y publicación de una solicitud de ofertas, la selección del proveedor del software y la correspondiente gestión de los procesos desde la adquisición hasta la aceptación del producto.
- **Proceso de suministro**: Contiene las actividades y tareas que el suministrador o proveedor realiza. Comienzan con la preparación de una propuesta para responder a una petición de oferta de un comprador o con la firma de un contrato con el comprador para proporcionarle un producto software. Trata, asimismo de la identificación de los procedimientos y de los recursos necesarios para gestionar y garantizar el éxito del proyecto, incluyendo el desarrollo de los planes del proyecto y la ejecución de dichos planes hasta la entrega del producto software al comprador.
- **Proceso de desarrollo**: Contiene las actividades de análisis de requisitos, diseño, codificación, integración, pruebas e instalación y aceptación. Vamos a resumir someramente estas actividades:
  - *Análisis de requisitos del sistema*: Aquí son especificados todos los requisitos del Sistema de Información, funciones y capacidades que debe cumplir, requisitos de seguridad, interfaces, de mantenimiento, etc.
  - *Diseño de la arquitectura del sistema*: Se identifican los principales componentes hardware y software.
  - *Análisis de los requisitos de software*: Se establecen dichos requisitos, incluyendo el nivel de calidad que debe cumplir el sistema.

    - *Diseño de la arquitectura del software*: El diseñador debe transformar el análisis anterior en una arquitectura en la que se puedan identificar sus componentes principales.
    - *Diseño detallado del software*: Aquí se realiza un diseño detallado de cada componente software, de las BD y manuales de usuario.
    - *Codificación y pruebas unitarias*: Se desarrollan y se documentan los componentes del punto anterior. Finalmente se realizan las pruebas unitarias de cada uno de ellos para asegurarse de que cumplen los requisitos exigidos.
    - *Pruebas de integración*: Se integran los componentes del software realizando las correspondientes pruebas.
    - *Prueba del software*: Las pruebas se planifican y diseñan de forma sistemática para poder detectar el máximo número y variedad de defectos con el mínimo consumo de tiempo y esfuerzo.
    - *Integración del sistema*: Aquí se realizan las pruebas conjuntas de los elementos hardware y software.
    - *Implantación del software desarrollado en el entorno de explotación final*. Cuando se sustituya a un software ya existente, puede ser recomendable un período de tiempo en el que convivan los dos sistemas.
    - *Proceso de aceptación del software*.
- **Proceso de explotación**: Comprende la propia explotación del software y el soporte operativo a los usuarios del sistema.
- **Proceso de mantenimiento**: Aparece cuando, tarde o temprano, el software requiere modificaciones, bien por errores, necesidades de mejora, etc.

Procesos de Soporte

Sirven de apoyo al resto de procesos y pueden aplicarse en cualquier punto del ciclo de vida.

- **Proceso de documentación**: Comprende todas las actividades que permiten desarrollar, distribuir y mantener la documentación necesaria para todas las personas involucradas: consultores, jefes de proyecto, analistas, programadores, usuarios, etc.
- **Proceso de gestión de la configuración**: Controla las modificaciones y las versiones de los elementos de configuración del software del sistema.
- **Proceso de aseguramiento de la calidad**: Comprueba que los procesos y los productos software del ciclo de vida cumplen con los requisitos especificados y se ajustan a los plantes establecidos.
- **Proceso de verificación**: El objetivo es demostrar la consistencia, completitud y corrección del software entre las fases del ciclo de desarrollo de un proyecto (por ejemplo, si el código es coherente con el diseño). Este proceso puede ser responsabilidad de una empresa de servicios y, en este caso se conoce como proceso de verificación independiente.
- **Proceso de validación**: El objetivo es determinar la corrección del producto final respecto a las necesidades del usuario. Al igual que el anterior, este proceso puede ser ejecutado por una organización de servicios, denominándose proceso de validación independiente.
- **Proceso de revisión conjunta**: Para evaluar el estado del software y sus productos en una determinada actividad del ciclo de vida o una fase de un proyecto. Las revisiones conjuntas se celebran tanto a nivel de gestión como a nivel técnico del proyecto a lo largo de todo su ciclo de vida. Un mecanismo habitual de revisión son las reuniones y la responsabilidad es generalmente compartida entre un grupo de personas pertenecientes a la organización.
- **Proceso de auditoría**: Permite determinar, en los hitos preestablecidos, si se han cumplido los requisitos, los planes y, en suma, el contrato.
- **Proceso de resolución de problemas**: Permite analizar y solucionar los problemas, sean éstos diferencias con los requisitos o con el contrato. Aporta un medio oportuno

y documentado para asegurar que los problemas detectados son analizados y solucionados.

Procesos de la Organización

Son los utilizados por una organización para llevar a cabo funciones como la gestión, formación del personal o procesos de mejora continua.

- **Proceso de gestión**: Contiene las actividades y las tareas genéricas que puede emplear una organización que tenga que gestionar sus procesos. Incluye actividades como la planificación, el seguimiento y control, la revisión y evaluación.
- **Proceso de infraestructura**: Establece la infraestructura necesaria para cualquier otro proceso: hardware, software, herramientas, técnicas, etc para el desarrollo, explotación y mantenimiento.
- **Proceso de mejora**: Para mejorar los procesos del ciclo de vida del software.
- **Proceso de formación**: Para mantener al personal con la adecuada formación, lo que conlleva el desarrollo del material de formación, así como la implementación del plan de formación de la organización.

Proceso de Adaptación

Sirve para realizar la adaptación básica de la norma ISO 12207-1 respecto a los proyectos software. Como es sabido, las variaciones en las políticas y procedimientos de la organización, los métodos y estrategias de adquisición, el tamaño y complejidad de los proyectos, los requisitos del sistema y los métodos de desarrollo, entre otros, influencian la forma de adquirir, desarrollar, explotar o mantener un sistema.

Dado que los procesos se aplican durante el ciclo de vida del software, y además se utilizan de diferentes formas por las diferentes organizaciones y con distintos puntos de vista y objetivos, es preciso comprender los procesos, las organizaciones y sus relaciones bajo diferentes puntos de vista:

- Contrato: El comprador y el proveedor negocian y firman el contrato, empleando los procesos de adquisición y suministro.
- Gestión o dirección: El comprador, el proveedor, el desarrollador, el operador y el personal de mantenimiento gestionan sus respectivos procesos en el proyecto software.
- Explotación: El operador proporciona el servicio de explotación del software a los usuarios.
- Ingeniería: El desarrollador o el personal de mantenimiento llevan a cabo sus respectivas tareas de ingeniería para producir o modificar los productos de software.
- Soporte: Los grupos de soporte (el de gestión de la configuración, el de aseguramiento de la calidad, el de auditoría, etc) proporcionan servicios de apoyo a otros grupos en el cumplimiento de tareas únicas y específicas.

Modelo en Cascada

Este modelo nació durante los años setenta y supuso un gran avance con respecto a los modelos que habían sido utilizados hasta entonces. Este modelo se compone de una serie de fases que se suceden secuencialmente, generándose en cada una de las fases resultados que constituyen la entrada de la fase siguiente. Estas fases pueden diferir, pero suelen comprender:

- Planificación
- Especificación de Requisitos
- Diseño
- Codificación

- Pruebas e Integración
- Implantación y Aceptación
- Mantenimiento

La fase de especificación de requisitos es conocida también como análisis funcional, la fase de diseño se denomina análisis orgánico y la fase de codificación se llama programación.

El número de fases es irrelevante, lo que caracteriza verdaderamente a este modelo es la secuencialidad entre las fases y la necesidad de completar cada una de ellas para pasar a la siguiente. El sistema está terminado cuando se han realizado todas las fases.

El modelo en cascada ayudó a eliminar muchos de los problemas que se planteaban antes de su utilización, además ha sido la base para la normalización y la adopción de estándares. A medida que ha sido utilizado se han detectado en él debilidades e inconsistencias que se han intentado corregir con diversas modificaciones y extensiones al modelo inicial.

Fases del Modelo en Cascada

Vamos a analizar cada una de las posibles fases:

Planificación

De esta fase depende en gran medida un desarrollo efectivo en lo referente a costos y plazos de entrega.

Hay que fijar los siguientes puntos:

- Ámbito del trabajo a realizar.
- Recursos necesarios.
- Tareas a realizar.
- Referencias a considerar.
- Coste aproximado del desarrollo del proyecto.
- Formación del equipo de desarrollo.
- Ordenación y calendario de las actividades.

La planificación se llevará a cabo con un nivel de detalle adecuado a la complejidad, tamaño y grado de estructuración del proyecto. Para proyectos de gran tamaño la planificación es imprescindible y en ocasiones sirve para determinar el modelo de ciclo de vida a seguir en el proyecto.

La estimación de recursos, costes y calendario se determina a partir de la experiencia acumulada por parte del jefe de proyecto y de la información histórica de otros proyectos realizados dentro o fuera de la organización. Esta es una de las fases más difíciles de realizar, pero en la actualidad se cuenta con técnicas y herramientas automatizadas para el control y la gestión del proceso de producción de los sistemas de información.

Especificación de Requisitos – Análisis Funcional

Una vez terminada la planificación del proyecto, la fase de análisis de requisitos es la primera del proceso de desarrollo del sistema. En esta fase es preciso analizar, entender y documentar el problema que el usuario trata de resolver con el sistema de información o aplicación a desarrollar.

Es necesario especificar en detalle las funciones, objetivos y restricciones del sistema propuesto para que el usuario y los desarrolladores puedan tomar estas especificaciones como punto de partida para acometer el resto del sistema.

El proceso de recogida de requisitos es la tarea más delicada de esta fase en la cual el analista del sistema debe llegar a comprender el dominio de la información y adaptar las necesidades del usuario a unas especificaciones formales listas para poder ser utilizadas por los desarrolladores.

Se trata de definir “qué” debe hacer el sistema identificando la información a procesar, las funciones a realizar, el rendimiento deseado del sistema, las interfaces con otros sistemas o las restricciones de diseño entre otros aspectos. Es fundamental en esta fase la participación e implicación del usuario del sistema.

Para el análisis de las necesidades a cubrir y los requisitos a satisfacer por el sistema, su priorización, comprobar que el sistema se ajusta a las necesidades del usuario y plantear alternativas viables no solo a nivel técnico sino desde el punto de vista de costes y riesgos, hay que utilizar todas aquellas técnicas o elementos a nuestro alcance como, por ejemplo, realización de entrevistas con los usuarios, utilización de información referente al sistema actual si es que existe, utilización de técnicas de diagramación para facilitar la comprensión del sistema, técnicas de análisis coste-beneficio, técnicas de prototipado rápido o técnicas de análisis estructurado.

Las tareas asociadas a esta fase y los resultados que se obtienen serán independientes del entorno tecnológico del sistema de información.

Diseño – Análisis Orgánico

A partir, como siempre, de las especificaciones de la fase anterior y una vez elegida la mejor alternativa, se debe comenzar a crear la solución al problema descrito atendiendo a aspectos de interfaz de usuario, estructura del sistema y de decisiones sobre la implantación posterior.

Esta fase aborda el “cómo”, es decir, deberá diseñar las estructuras de datos, la arquitectura del sistema, los detalles que permitan la codificación posterior y las pruebas a realizar.

Para el diseño del sistema habrá que trasladar las especificación de requisitos a un conjunto de representaciones ya sean gráficas, tabulares o basadas en lenguajes que constituirán la estructura de datos lógica y física, la arquitectura y los procedimientos.

Otras cuestiones que se abordan en esta fase son los requisitos de comunicaciones, algoritmos, seguridad y control. Al igual que en las fases anteriores la de diseño conlleva una documentación en la que se recogen sus resultados. En esta fase hay que tener en cuenta el entorno del sistema referente a hardware y software de base.

Codificación – Programación

En esta fase se traducen las especificaciones de diseño a un lenguaje de programación capaz de ser interpretado y ejecutado por el ordenador. Existen lenguajes de distintos grados de complejidad o eficacia y la utilización de uno u otro determinará la forma de trabajo de esta fase. En todo caso, el lenguaje vendrá determinado por el entorno lógico del sistema.

El programador deberá velar por la claridad de su estilo para facilitar cualquier interpretación posterior de los programas. Asimismo se respetarán los estándares de la organización en cuanto a nomenclatura y formato. Es imprescindible que los programas incorporen comentarios escritos que ayuden a su comprensión y que se acompañen de la documentación externa necesaria que describa su objeto, los algoritmos que incluye, sus entradas y salidas y cualquier otro elemento relevante.

Muchas son las técnicas aplicables a la programación, como, por ejemplo, las técnicas estructuradas ampliamente extendidas desde hace años. Más reciente es la generación

automática de código, que a partir de especificaciones formales algunas herramientas CASE (Computed Aided Software Engineering) facilitan con un mayor o menor grado de optimización, según los casos, código en los lenguajes de programación más utilizados.

Pruebas e Integración

Una vez que los programas han sido desarrollados, es preciso llevar a cabo las pruebas necesarias para asegurar la corrección de la lógica interna de los mismos y comprobar que cubren las funcionalidades previstas.

La integración de las distintas partes que componen la aplicación o el sistema es precisa en proyectos complejos o de grandes dimensiones que hayan sido descompuestos por razones de facilidad de gestión y control. La integración debe solucionar posibles problemas de interacción y garantizar el buen funcionamiento del conjunto.

En esta fase se debe proporcionar la documentación que ilustre los procedimientos de usuario en la utilización y funcionamiento del sistema y si fuera preciso se organiza un plan de formación para usuarios con el material didáctico necesario.

Como en las fases anteriores existen técnicas y herramientas para la realización de las tareas de esta fase.

Implantación y Aceptación

En esta fase se trata de conseguir la aceptación final del sistema por parte de los usuarios del mismo y llevar a cabo las actividades necesarias para su puesta en producción.

Para ello se tomarán como punto de partida los componentes del sistema probados de forma unitaria e integrada en la fase anterior y se probarán una vez más esta vez con el fin de verificar que cumplen los requisitos de usuario y que el sistema es capaz de manipular los volúmenes de información requeridos en los tiempos y velocidades deseados, que las interfaces con otros sistemas funcionan, etc.

En estas pruebas participará el usuario que si está conforme deberá aceptar formalmente el sistema.

Mantenimiento

Esta fase comienza una vez que el sistema es entregado al usuario y continúa mientras permanece activa su vida útil. El mantenimiento puede venir propiciado por:

- Errores no detectados previamente.
- Modificaciones, mejoras o ampliaciones solicitadas por los usuarios.
- Adaptaciones requeridas por la evolución del entorno tecnológico o cambios normativos.

En el primer caso se habla de mantenimiento “correctivo” puesto que debe solventar defectos en el sistema. El segundo caso se denomina mantenimiento “perfectivo” puesto que se produce una modificación de los requisitos iniciales aumentando las funcionalidades. El último de los casos se conoce por mantenimiento “adaptativo”, con el paso del tiempo es muy posible que la situación inicial respecto de la cual se concibió el sistema cambie.

Para la realización del mantenimiento se siguen los mismos pasos que para la realización del sistema, pero en el contexto de un sistema existente. Es fundamental que las variaciones producidas en esta fase queden reflejadas en todas las fases anteriores y no simplemente en la fase de codificación. De lo contrario esta práctica conduciría a sistemas intratables al cabo de varias modificaciones sin actualización de la documentación afectada.

La fase de mantenimiento lleva asociada su propia documentación reflejando los cambios, su objeto, la fecha, el autor y cualquier dato que pueda ayudar al control de dichos cambios o a procesos de mantenimiento posteriores.

La Documentación

El modelo de ciclo de vida en cascada está regido por la documentación, es decir, la decisión del paso de una fase a la siguiente se toma en función de si la documentación asociada a esa fase está completa o no.

El concepto de documentación debe entenderse en sentido amplio como todos los productos resultantes de las tareas realizadas en cada fase ya sean informes, programas, juegos de pruebas, etc, se podría definir la documentación como aquello que se construye y ha de mantenerse durante la vida del sistema.

A pesar de posibles críticas a esta orientación se recogen a continuación las características que deben incluir los documentos asociados a cada fase. No hay que olvidar que el objetivo final es construir con éxito un sistema de información y que la documentación nos ayudará a conseguirlo pero no es un fin en sí misma.

En la fase de planificación se elaborará documentación que tendrá el carácter de marco básico de referencia del proyecto y deberá incluir:

- Descripción y alcance de las tareas que se van a llevar a cabo.
- Identificación de los principales métodos, instrumentos y procedimientos de trabajo que se utilizarán.
- Procedimientos de seguimiento y control de los trabajos y mecanismos de revisión y aprobación de los mismos.
- Calendario de tareas y su ordenación temporal.
- Asignación de recursos.
- Organización de las personas que intervienen en el proyecto.

En la fase de especificación de requisitos será necesaria la existencia de documentación en el que se recojan las necesidades del usuario respecto al sistema: funcionalidades, rendimientos, limitaciones y restricciones, los interfaces de usuario. La descripción de los requisitos de los usuario debe ser descrita de forma que se pueda verificar su cumplimiento mediante inspecciones o pruebas. Este documento reflejará el punto de vista del analista del sistema respecto de las especificaciones que el usuario ha realizado, si la comprensión por parte del analista no es adecuada el usuario debe rechazar el documento. Este documento es crucial porque sobre sus presupuestos se construirá el sistema de información, por tanto se detallará la naturaleza de la información, su contenido y su estructura lógica. La representación de las operaciones y procesos, sus entradas y salidas y cualquier característica relevante a nivel funcional referente al entorno tecnológico del sistema.

En la documentación asociada a la fase de diseño se profundizará más, llegando a describir los componentes del sistema: estructuras de datos, unidades de tratamiento o módulos de procesamiento e interfaces al máximo nivel de detalle. Se concretará la descripción técnica y cuestiones relacionadas con la implantación del sistema: arquitectura general, fichero y/o BD, pantallas, informes o comunicaciones con otros sistemas.

La fase de codificación debe proporcionar como documentación, el código fuente de todos los módulos incluidas las funciones auxiliares, el código para la creación de estructuras de datos e interfaces externas y cualquier tipo de módulos o rutinas relacionadas con el sistema. Además del código de cada módulo en el soporte físico y formato de representación previamente establecido por el usuario, se acompañará el listado del código con sus comentarios internos y comentarios externos sobre la definición de las

estructuras de datos, de los algoritmos, sobre el manejo de excepciones y la gestión de errores.

La fase de pruebas e integración llevará documentación que describa el plan de pruebas a nivel unitario e integrado y los resultados de dichas pruebas. La fase de implantación y aceptación vendrá documentada con el plan de pruebas del sistema a nivel global y sus resultados. Por último si se realizan modificaciones en la fase de mantenimiento deberán reflejarse en la documentación correspondiente que haya podido ser afectada.

La documentación asociada al sistema no estaría completa sin un manual de usuario que contenga la descripción funcional de todos los procedimientos para facilitar la operativa del sistema al usuario. Recogerá los procedimientos de instalación, administración y gestión de la configuración, operaciones especiales, funcionamiento, ayudas incorporadas, tratamiento de errores, comandos y sentencias de control. A veces también puede incluir una guía de referencia rápida con el resumen de todas estas instrucciones.

Crítica del Modelo

Las principales críticas al modelo se centran en sus características básicas, es decir, secuencialidad y utilización de los resultados de una fase para acometer la siguiente de modo que el sistema sólo se puede validar cuando está terminado.

Los proyectos reales raramente siguen el flujo secuencial que propone el modelo. Siempre ocurren interacciones y en las últimas fases sobre todo se pueden realizar en paralelo algunas áreas como por ejemplo: codificación y pruebas. Una aplicación del modelo en sentido estricto obligaría a la “congelación” de los requisitos de los usuarios, supuesto este completamente alejado de la realidad. El modelo no contempla la posibilidad de realimentación entre fases.

El modelo en su formulación pura no prevé revisiones o validaciones intermedias por parte del usuario, así los resultados de los trabajos sólo se ven al final de una serie de tareas y fases de tal forma que si se ha producido un error en las primeras fases éste sólo se detectará al final y su corrección tendrá un costo muy elevado, puesto que será preciso rehacer todo el trabajo desde el principio. El modelo no dispone de resultados parciales que permitan validar si el sistema cumple con los requisitos desde las primeras fases, dándose el caso de sistemas perfectamente formalizados y documentados que no cumplen los requisitos del usuario.

Extensiones al Modelo en Cascada

Actualmente, después de la experiencia obtenida con la aplicación del modelo en cascada y gracias a avances tecnológicos como por ejemplo lenguajes de cuarta generación o herramientas CASE existen modelos de ciclo de vida alternativos al modelo en cascada. En la práctica se llegan a realizar incluso modelos mixtos. En este punto no se tratarán estos casos, sino que se citarán algunas innovaciones aplicables al modelo en cascada que permiten mejorar algunas de las deficiencias del modelo y aumentar su eficacia:

- Utilización de herramientas CASE y otras facilidades. Al hablar de las fases del ciclo de vida se mencionaban las técnicas estructuradas, en principio dichas técnicas no se utilizaban en el modelo en cascada, sin embargo, hoy en día no se plantea la realización de un sistema “artesanalmente”.
- Introducción de una fase intermedia entre especificación de requisitos y el diseño denominado diseño rápido que sirva para validar las especificaciones por parte del usuario, estableciéndose un proceso iterativo de modificación de la especificación hasta que el usuario esté satisfecho con la misma.
- Técnicas de diseño en grupo. Es deseable que el usuario pueda participar al máximo en la definición de los requisitos puesto que se evitarán errores y confusiones ya en las primeras etapas.

- Utilizar técnicas de análisis de riesgos y de coste-beneficio para pasar a la fase siguiente y abandonar planteamientos rígidos de paso a la fase siguiente cuando se ha completado la documentación.
- Dotar de autonomía al jefe de proyecto para que de acuerdo con su experiencia y las características del proyecto decida comenzar una fase sin haber terminado la anterior al 100%, pero que le permite, por ejemplo, realizar una maqueta para la validación por parte del usuario.
- Codificación y pruebas de los módulos de más alto nivel en primer lugar, seguida de la de los módulos más detallados o de más bajo nivel. Esta aproximación puede propiciar el diálogo entre el analista y el usuario en fases posteriores a la especificación de requisitos estableciendo así un proceso de realimentación entre las fases de Implantación y Especificación de requisitos.
- Realización de fases en paralelo como codificación y pruebas. La codificación se puede así ver beneficiada por los resultados de las pruebas y detección de errores.
- Reutilización de código ya probado. A veces con pequeñas modificaciones se pueden llegar a utilizar programas existentes.
- Revisiones de código. Se trata de inspecciones para localizar lo más pronto posible dentro del ciclo todos los errores de diseño y codificación.

Modelo en Espiral del Ciclo de Vida

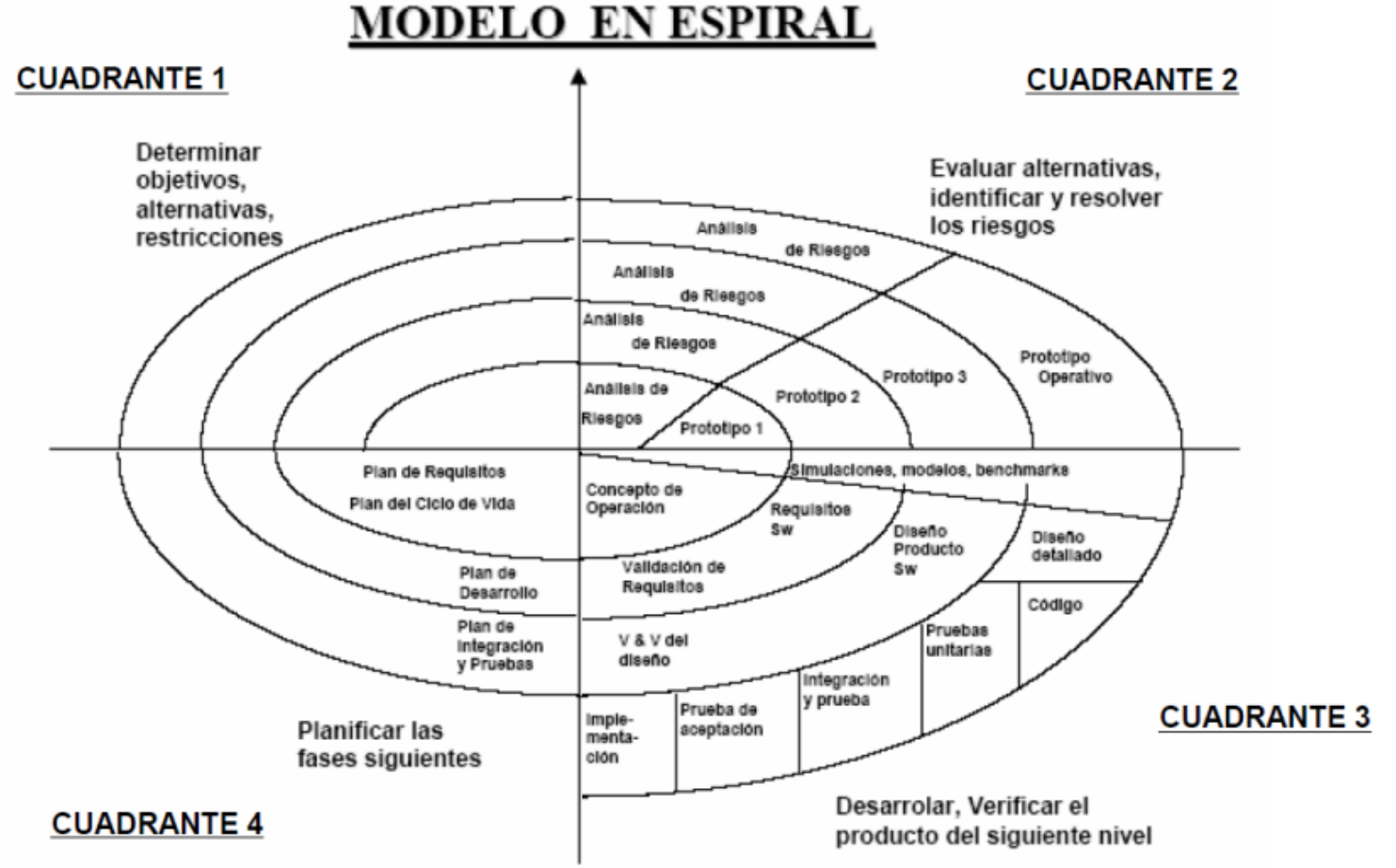
Introducción

A partir de la experiencia de la aplicación del modelo en cascada se desarrolló el modelo en espiral del ciclo de vida del software y se implementó en algunos grandes proyectos.

En este modelo subyace el concepto de que cada ciclo implica una progresión que aplica la misma secuencia de pasos para cada parte del producto y para cada uno de sus niveles de elaboración, desde la concepción global de la operación hasta la codificación de cada programa individual.

El modelo en espiral se ilustra en la siguiente figura. En ella podemos apreciar:

- *La dimensión radial:* representa el coste acumulativo en el que se ha incurrido en las etapas realizadas hasta el momento actual.
- *La dimensión angular*: representa el progreso hecho en completar cada ciclo de la espiral.



Describamos a continuación como sería un típico ciclo de espiral. Considerando cada cuadrante de izquierda a derecha en el sentido de las agujas del reloj.

CUADRANTE 1: Cada ciclo en espiral empieza con la indentificación de:

- Los objetivos de la parte del producto que va a ser elaborada (rendimiento, funcionalidad, disponibilidad para acomodarse a nuevos cambios, etc.)
- Las alternativas para implementar esta parte del producto (diseño A, diseño B, compra del software ya desarrollado, reutilización de un software ya existente, etc).
- Las restricciones impuestas: costes, calendario de realización, interfaces, etc.

CUADRANTE 2: Aquí se evalúan las opciones relativas a los objetivos y las restricciones.

Este proceso, con frecuencia, identificará áreas de incertidumbre que son fuentes significativas de riesgo. Esto llevaría implícito la formulación de una estrategia económicamente efectiva para resolver las fuentes de riesgo: prototipado, simulación, benchmark, modelos analíticos o combinaciones de éstas y otras técnicas de resolución de riesgos.

Si los riesgos de rendimiento o los riegos del interfase de usuario tienen mucha más importancia que los riesgos de desarrollo de programas o los riesgos de interfase de control interno, entonces el paso siguiente podría ser un paso de desarrollo evolutivo: un esfuerzo mínimo para especificar la naturaleza global del producto, un plan para el siguiente nivel de prototipado y el desarrollo de un prototipo más detallado para continuar la resolución de las mayores fuentes de riesgo.

Si este prototipo es operacionalmente útil y suficientemente robusto para servir como una base de bajo riesgo para el evolución del producto, entonces, los subsiguientes pasos dirigidos por el riesgo (risk-drive) podrían ser una serie de prototipos cada vez más evolucionados, moviéndonos hacia la derecha de la figura. Así, consideraciones de riesgo, pueden llevar a la realización del proyecto utilizando sólo un subconjunto de todos los pasos potenciales en el modelo.

CUADRANTES 3 y 4: Si los esfuerzos previos de prototipado han resuelto ya todos los riesgos de rendimiento o los riesgos de interface de usuario y dominan los riesgos de desarrollo en programas o los riesgos de control de interface, el paso siguiente sería el desarrollo según el modelo en cascada. Cada nivel de especificación del software en la figura, entonces, seguido por un paso de validación y planificación del siguiente ciclo.

Esta implementación, dirigida por el riesgo, de un subconjunto del modelo en espiral, permite al modelo acomodarse a cualquier mezcla de estrategias de desarrollo de software: orientado por las especificaciones, orientado por la simulación, orientado por transformaciones automáticas o cualquier otro enfoque de desarrollo.

En tales casos, la estrategia mixta apropiada se escoge, considerando, la relación relativa de magnitudes de los riesgos y la efectividad relativa de las distintas alternativas en la resolución de estos riesgos. De forma similar, consideraciones de gestión de riesgo, permiten determinar la cantidad de tiempo y esfuerzo que debe dedicarse a otras actividades del proyecto tales como: planificación, gestión de la configuración, garantía de la calidad, verificación formal, y prueba.

Un aspecto importante del modelo espiral, como en otros muchos modelos, es que cada ciclo se completa por una revisión que involucra a las personas u organizaciones principales relacionadas con el proyecto. Esta revisión cubre todas las actividades desarrolladas durante el ciclo previo, incluyendo los planes para el próximo ciclo y los recursos que se requieren para llevarlos a cabo.

El principal objetivo de la revisión es asegurar que todas las partes implicadas están de acuerdo con el camino a seguir en la siguiente fase.

Los planes para las fases sucesivas pueden incluir también el desarrollo del producto por medio de incrementos sucesivos o la división en componentes que pueden ser desarrollados por distintas personas u organizaciones. En este último caso, aparecen unos ciclos espirales paralelos (uno para cada componente) añadiendo una tercera dimensión al concepto presentado en la figura.

Además, la etapa de revisión y compromiso puede extenderse, desde una simple revisión informal del diseño de un programa a una revisión de los requerimientos principales implicando, en este caso, a clientes, usuarios, desarrolladores y organizaciones de mantenimiento.

Ejemplo de Aplicación del Modelo en Espiral

El modelo en espiral se aplicó en un proyecto muy complejo: la definición y desarrollo del Sistema de Productividad de Software (TRW-SPS) un entorno integrado de ingeniería de software de la empresa TRW. El objetivo era mejorar la productividad de las tareas de desarrollo de software realizadas por la empresa.

En primer lugar se realizó un “ciclo 0” de la espiral para determinar la viabilidad de conseguir un incremento significativo de productividad en el desarrollo software.

Ciclo 0: Estudio de viabilidad

Participaron cinco personal, a tiempo parcial, durante un período de dos a tres meses. Durante este ciclo los objetivos y las restricciones se consideraron a un nivel muy alto, expresándoles en términos cualitativos (“mejora significativa”, “coste razonable”, etc).

Se consideraron alternativas en cuatro áreas: gestión de los proyectos, gestión del personal, tecnología e instalaciones.

Como áreas de riesgo, se consideró la posibilidad de que la compañía realizase una inversión importante para encontrarse con:

- Mejoras de productividad escasamente significativas.
- Mejoras potenciales incompatibles con aspectos de la cultura de la empresa.

El análisis de riesgos condujo a la conclusión de que se podrían obtener significativas mejoras en la productividad, a un coste razonable, por medio de un conjunto integrado de iniciativas.

Ciclo 1: Concepción de la operación

En este ciclo se invirtieron más recursos (12 meses hombre en lugar de los 2 meses hombre del Ciclo 0); se consideraron objetivos más específicos (conseguir el doble de productividad en la producción de software en 5 años a un coste máximo de 10.000$ por persona); surgieron nuevas restricciones como la preferencia por la utilización de productos desarrollados por la propia empresa, en especial la red local de TRW.

Para las áreas de riesgo también se fue más específico que en el Ciclo 0 (“comprobación que la red TRW LAN ofrecía una relación precio/rendimiento dentro de la restricción de 10.0000$ por persona”). Para la resolución de riesgos se realizaron tareas más extensivas, tales como la realización de benchmarks y análisis de un prototipo de la TRW LAN.

La fase de compromisos no se limitó a la aceptación del plan. Se acordó la aplicación del entorno de desarrollo de software producido a un proyecto piloto que implicaba a más de 100 personas. Se decidió la formación de un comité de seguimiento para garantizar la coordinación de las distintas actividades y para evitar que el prototipo de entorno de desarrollo no se optimizase, excesivamente, en función de las características del proyecto en el que se iba a probar. Se recomendó que no solamente se desarrollase el prototipo, sino que también se elaborase una especificación de requerimientos y un diseño siguiendo una orientación orientada al riesgo.

Ciclo 2: Especificación de los requerimientos de alto nivel

Se tomaron las decisiones al comienzo del ciclo al observarse que muchos de los requisitos del sistema dependían de la decisión sobre el SO a utilizar y de si el sistema a elaborar iba a ser un sistema orientado al host o totalmente portable. Se decidió escoger UNIX y un sistema orientado a un host UNIX.

Otros ciclos posteriores

Las etapas posteriores se realizaron según las características generales de la implantación del modelo en este caso:

- Desarrollo de especificaciones, postergando la elaboración de los elementos de software de bajo riesgo hasta que no se hubieran estabilizado los elementos de software de alto riesgo.
- Incorpora la construcción de prototipos como técnica de reducción de riesgos en cualquiera de las etapas del proyecto.
- Permite la “vuelta atrás” a etapas anteriores cuando se encuentran alternativas más atractivas o se identifican nuevas fuentes de riesgo que requieren solución.

Evaluación del Modelo

Ventajas

La ventaja principal del modelo en espiral es que su rango de opciones acomoda las buenas características de los otros modelos de desarrollo de software, y su procedimiento dirigido por el riesgo, evita muchas de sus dificultades.

En situaciones apropiadas, el modelo en espiral es equivalente a uno de los modelos de proceso existentes.

Si un proyecto tiene un riesgo bajo en áreas tales como el establecimiento de una interfaz de usuario no adecuada o en el cumplimiento de requerimientos rigurosos de ejecución y si, por el contrario, tiene un alto riesgo en la predicción y control del presupuesto y del calendario de elaboración, entonces estas consideraciones conducen el modelo en espiral a uno equivalente al modelo en cascada.

El modelo en espiral tiene otras ventajas adicionales:

- Concentra su atención en opciones que consideran la reutilización de software existente. Los pasos implicados en la identificación y evaluación de alternativas potencian estas opciones.
- Permite preparar la evolución del ciclo de vida, crecimiento y cambios del producto software.
- Proporciona un mecanismo de incorporación de objetivos de calidad en el desarrollo de producto software. Este mecanismo se deriva del énfasis puesto en la identificación de todos los objetivos y restricciones durante cada una de las vueltas de la espiral.
- Es especialmente adecuado para la temprana eliminación de errores y alternativas poco atractivas.
- No implica procedimientos separados para el desarrollo y la mejora del software.
- Proporciona un marco viable para integrar los desarrollos de sistemas software-hardware. El centrar la atención en la gestión del riesgo y en la eliminación de alternativas no atractivas lo antes posible y con el menor coste es, igualmente, aplicable al software y al hardware.
- Se adapta al diseño y programación orientada a objetos y posiblemente con estos métodos es cuando permite obtener mejores resultados: el modelo en espiral potencia la utilización de desarrollos incrementales, y cuando sea necesario, la reelaboración de partes ya desarrolladas. La abstracción, encapsulación, modularidad y jerarquización, elementos fundamentales de los métodos orientados a objeto, facilitan los desarrollos incrementales y hacen menos traumáticas las reelaboraciones.

Dificultades

Adaptar su aplicabilidad al software contratado

El modelo en espiral actualmente trabaja bien en desarrollos de software internos pero necesita un trabajo adicional para adaptarlo al mundo de los contratos de adquisición del software.

Los desarrollos internos de software tienen bastante flexibilidad y libertad para acomodarse a confirmaciones etapa por etapa; para diferir confirmaciones a determinadas opciones; para establecer miniespirales con las que resolver caminos críticos; para ajustar niveles de esfuerzo, o para acomodar prácticas tales como el prototipado, o el desarrollo evolutivo.

En el mundo de la adquisición de software es muy difícil conseguir estos grados de flexibilidad y libertad sin perder responsabilidad y control y es muy difícil definir contratos que no especifiquen por adelantado los productos a obtener.

Recientemente, se ha progresado en el establecimiento de mecanismos de contratación más flexibles pero todavía se necesitan mejoras para conseguir que los compradores se sientan completamente cómodos utilizándolos.

Dependencia de la experiencia en la evaluación de riesgos

El modelo en espiral da un papel relevante a la habilidad de los desarrolladores de software para identificar y gestionar las fuentes de riesgo del proyecto. Un buen ejemplo de esto es la especificación dirigida por el riesgo en el modelo en espiral. En ella se debe llegar a un alto nivel de detalle en los elementos de alto riesgo dejando, los de bajo riesgo, para una elaboración en etapas posteriores.

Otro problema es que la especificación será, en exceso, dependiente de las personas. Por ejemplo, un diseño producido por un experto puede ser implantado por no expertos; en este caso, el experto que no necesita mucha documentación, debe producir suficiente documentación adicional para que los no expertos no se extravíen.

Con una aproximación convencional, dirigida por la documentación, el requerimiento de llevar todos los aspectos de la especificación a un nivel uniforme de detalle elimina algunos problemas potenciales y permite una adecuada revisión de algunos aspectos por revisores inexpertos. No obstante, esto produce pérdidas de tiempo a los expertos que deben buscar los aspectos críticos en medio de un gran volumen de detalles no críticos.

El Plan de Gestión de Riesgo

Incluso si una organización no está lista para adoptar el procedimiento completo en espiral, una característica técnica del mismo que puede ser fácilmente adaptada a cualquier modelo de ciclo de vida proporciona muchos de los beneficios de dicho procedimiento. Se trata del Plan de Gestión del Riesgo.

Este plan, básicamente, asegura que en cada proyecto se haga una identificación temprana de sus factores de riesgo más altos; desarrolla una estrategia para resolver los factores de riesgo; identifica y establece una agenda para resolver nuevos factores de riesgo a medida que afloran y, por último, muestra los progresos respecto al plan en revisiones mensuales.

El plan de gestión de riesgo y las técnicas para gestión de riesgo de Software facilitan adaptar concepto del modelo en espiral a los procedimientos de adquisición y desarrollo de software más asentados. Se pueden sacar las siguientes cuatro conclusiones:

- El modelo en espiral, por su naturaleza de estar dirigido por el riesgo, es más adaptable a un amplio rango de situaciones que los procedimientos dirigidos por la documentación, tales como el modelo en cascada o los procedimientos dirigidos por el código, tales como el modelo de desarrollo evolutivo. Es particularmente aplicable a sistemas de software muy grandes y complejos.
- El modelo en espiral ha tenido éxito en su mayor aplicación realizada hasta ahora: el desarrollo del TRW-SPS. Proporciona la flexibilidad necesaria para acomodar un rango de alternativas técnicas y objetivos de usuario muy dinámico.
- El modelo en espiral no está, todavía, tan elaborado como los modelos más establecidos. Por esta razón, el modelo en espiral puede ser aplicado por personal experto, pero necesita elaboración posterior en áreas como contratación, especificaciones, puntos de control, revisiones, calendarios e identificación de áreas de riesgo para ser completamente aplicable en todas las situaciones.
- Algunas implementaciones parciales del modelo en espiral como el Plan de Gestión del Riesgo, son compatibles con la mayoría de los modelos de proceso actuales y son muy útiles para superar las mayores fuentes de riesgo de proyectos.

Tabla de los 10 factores de riesgo del software más importantes:

Factor de riesgo	Técnicas de gestión del riesgo
1. Escasez de personal	Formar una plantilla con personal de gran talento, integrado en el trabajo, integrado en equipo, con alta moral y formado, incluyendo la gente clave.
2. Presupuestos y calendarios no realistas	Estimación de costes y plazos detallada usando diversas fuentes; desarrollo incremental, reutilización de software, depuración de requisitos.
3. Desarrollo de funciones de software no adecuadas	Análisis de la organización, análisis del problema, entrevistas con usuarios, prototipos, desarrollo temprano de manuales de usuario.
4. Desarrollo de una interfaz de usuario no adecuada	Análisis de tareas, prototipos, guiones, caracterización de los usuarios (funcionalidad, estilo, carga de trabajo).
5. Recubrimiento dorado	Depuración de requerimientos, prototipos, análisis coste-beneficio, diseño ajustado al coste.
6. Continuos cambios de requerimientos	Alto umbral de cambio, desarrollo incremental (dejar los cambios para posteriores incrementos).
7. Componentes adquiridos en el exterior sin la calidad necesaria	Prueba, inspecciones, verificación de referencias, análisis de compatibilidad.
8. Tareas realizadas externamente que no tienen el nivel adecuado	Verificación de referencias, auditorías previas a la aceptación, contratos con penalización.
9. Problemas de rendimiento en tiempo real	Simulación, pruebas, modelado, prototipado, instrumentación, afinamiento.
10. Ir más allá del estado actual de las posibilidades de la informática	Análisis técnico, análisis coste-beneficio, prototipos, verificación de referencias.

Tabla del Plan de Gestión del Riesgo del Software:

Plan de Gestión del Riesgo del Software
1. Identificar los 10 factores del proyecto de mayor riesgo.
2. Presentar un plan para resolver cada factor de riesgo.
3. Actualizar mensualmente la lista de factores de mayor riesgo, planes y resultados.
4. Resaltar el estado de los factores de riesgo en las revisiones mensuales del proyecto. Comparar con el estado de situación de los meses anteriores.
5. Iniciar las acciones correctoras apropiadas.

Bibliografía

- Scribd (Ricardo Costanzi)

Gestión del proceso de desarrollo: objetivos, actores y actividades. Técnicas y prácticas de gestión de proyectos.

Introducción a la Gestión del Proceso de Desarrollo

Los fundamentos de gestión consisten en determinar el tamaño del producto (incluyendo funcionalidad, complejidad y otras características del producto), asignando los recursos apropiados a un producto de ese tamaño, creando un plan para aplicar esos recursos y luego controlando y dirigiendo los recursos para impedir que el proyecto se desvíe. En algunos casos los altos cargos delegan explícitamente estas tareas de gestión a los

responsables técnicos, y en otros casos simplemente las dejan vacantes, ocupándose de ellas un responsable o desarrollado motivado.

Definiciones

- **Proceso**. Conjunto de actividades con un objetivo, que transforman un conjunto de entradas en un conjunto de salidas, agregando valor.
- **Proceso de Software**. Proceso que transforma Requerimientos de Procesamiento de Información en Elementos de Software (componentes computacionales y documentos) que satisfacen los requerimientos de procesamiento de información, ofreciendo servicios a una organización.
- **Proceso de desarrollo de software**. Proceso cuyo objetivo es generar un nuevo sistema informático que satisfaga un conjunto de requerimientos iniciales de procesamiento de información en una organización.
- **Procesos de Administración de Proyectos**. Procesos orientados a organizar y guiar al equipo de desarrollo para cumplir los compromisos negociados con el cliente (fechas de entrega y costes).
- **Procesos de Ingeniería de Software**. Procesos orientados a obtener componentes de software (código y documentos) entregables al cliente, sin errores y con el menor esfuerzo posible.

Características de la Gestión del Proceso de Desarrollo

Las características principales de la gestión del proceso de desarrollo son las siguientes:

- El producto a desarrollar es intangible.
- El producto tiene su propia flexibilidad.
- La ingeniería de software no es reconocida como una disciplina de la Ingeniería con el mismo estatus de la mecánica, eléctrica, matemáticas, etc.
- El proceso de desarrollo de software no está estandarizado.

De estas características se deduce la importancia de la propia gestión, ya que la Ingeniería de software es una actividad económica importante, que está sujeta a restricciones económicas. No hay que olvidar que los proyectos bien gestionados a veces fallan. Los proyectos mal gestionados siempre fallan.

Causas de Fallos en los Proyectos

Dentro de las principales causas por las que puede fallar un proyecto, se encuentra el hecho de que los componentes del proyecto no respetan o no conocen bien las herramientas y las técnicas del análisis y diseño de sistemas, además de esto puede haber una mala gestión y dirección del proyecto.

Además existen una serie de factores que pueden hacer que el sistema sea mal evaluado:

- Requerimientos Incompletos.
- Usuarios no Involucrados.
    - Carencia de Recursos.
- Expectativas Irreales.
- Falta de soporte ejecutivo.
- Cambios de Requerimientos.
- Falta de planificación.
- Obsoleto antes de ser completado.
- Carencia de supervisión por parte de la gerencia de TI.
- Desconocimiento de la tecnología.
- No resuelve problemas de negocio.
- Requerimientos planificados de manera irreal.
- Carencia de entrenamiento en Administración de Programas.

- Mala Estimación.

Aunque estos factores pueden influir de manera muy trascendente en la mala realización de un proyecto, generalmente están acompañados de otro tipo de problemas. Pero, ¿cuáles de estos errores de gestión de proyectos ocasionan que no se cumplan los requisitos, que se sobrepase los tiempos de entrega o se aumenten repetidas veces los costes?

La respuesta a esta pregunta puede ser hallada en dos fuentes principalmente, deficiencias en las herramientas y las técnicas de análisis del diseño de sistemas o la mala gestión de los proyectos.

En el caso de las necesidades no satisfechas o no identificadas, el error puede aparecer debido a que se omiten datos durante el desarrollo del proyecto, es por esto que es muy importante no saltar ninguna etapa del ciclo de vida del desarrollo de sistemas.

Otra causa de insatisfacción de necesidades es la mala definición de las expectativas de un proyecto en sus orígenes, ya que si no están bien definidos los requerimientos máximos y mínimos que el proyecto debe satisfacer desde el comienzo, los desarrolladores verán afectados su trabajo por el *síndrome de las necesidades que crecen* el cual les dejará hacer cambios en el proyecto en cualquier momento sin detenerse a pensar si esos cambios serán buenos para el proyecto como un todo, por supuesto todas estas modificaciones acarrearan alteraciones en los costes y en los tiempos de entrega.

El coste de un proyecto puede aumentar durante el desarrollo de este debido a varias causas:

- Para comenzar un proyecto generalmente se exige un estudio de viabilidad en el cual no se incluyen datos completamente precisos de la cantidad de recursos que cada tarea consumirá, y es en base a este estudio que se hacen estimaciones de los recursos totales que el proyecto va a necesitar.
- El uso de criterios de estimación poco eficientes por parte de los analistas.
- El aumento en los tiempos de entrega debido a que los directores del proyecto en ocasiones no gestionan bien los tiempos de entrega de cada tarea del proyecto y cuando tienen un retraso no son capaces de alterar los plazos de entrega finales creyendo que podrán recuperar el tiempo perdido, pero no siempre es posible acelerar otras tareas para ahorrar tiempo en la entrega final.

Para evitar todos estos problemas, se debe tener al mando del proyecto un director que conozca las herramientas de diseño y análisis de sistemas y tenga una buena formación en las funciones de dirección.

Objetivos de la Gestión del Proceso Desarrollo

Los objetivos de la gestión son al final objetivos de calidad, es el primer paso de cualquier metodología de mejora, estos se pueden definir respondiendo la pregunta…

¿Cuáles son los puntos que queremos mejorar en la gestión?

Seguramente habrá muchos puntos que son susceptibles de mejora, sin embargo hay que considerar solo unos pocos y sobre todo aquellos que sean los que más nos interesa modificar.

Al establecer los objetivos debemos procurar definirlos de manera clara, concreta y deben ser cuantificables.

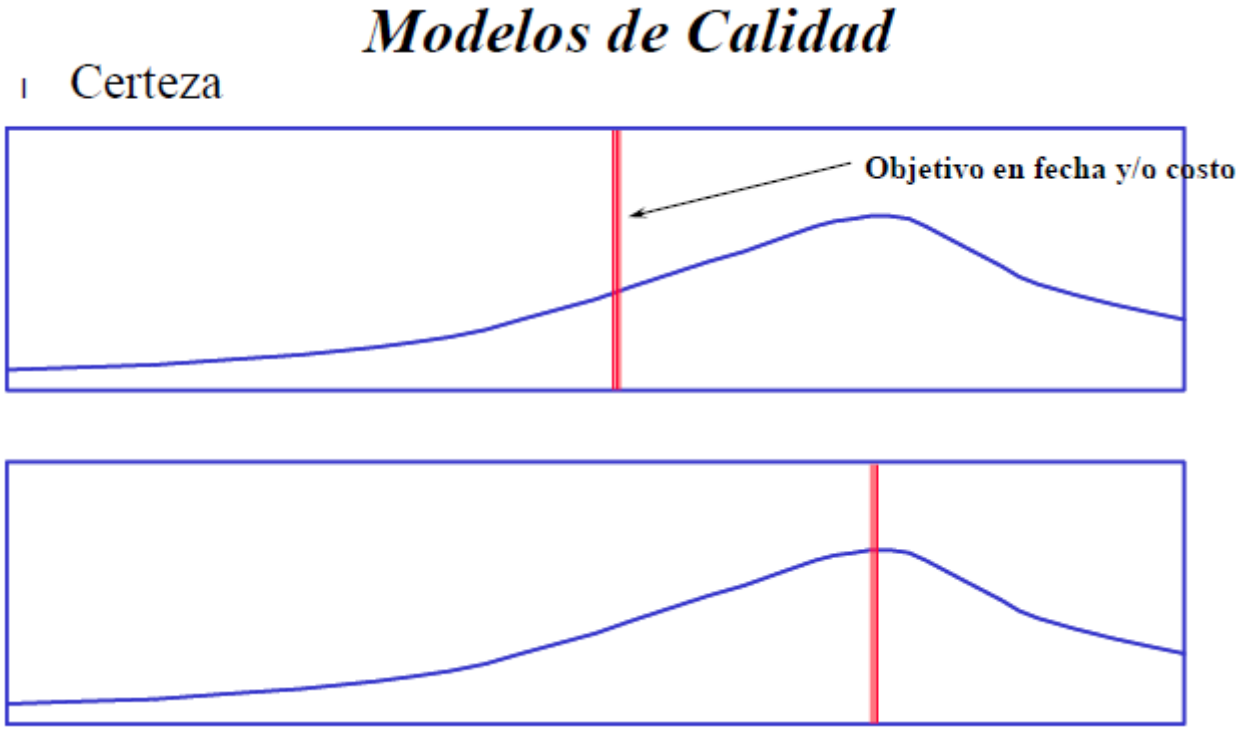
Básicamente estos podrían ser:

- Reducir la diferencia entre la fecha real y la fecha acordada.

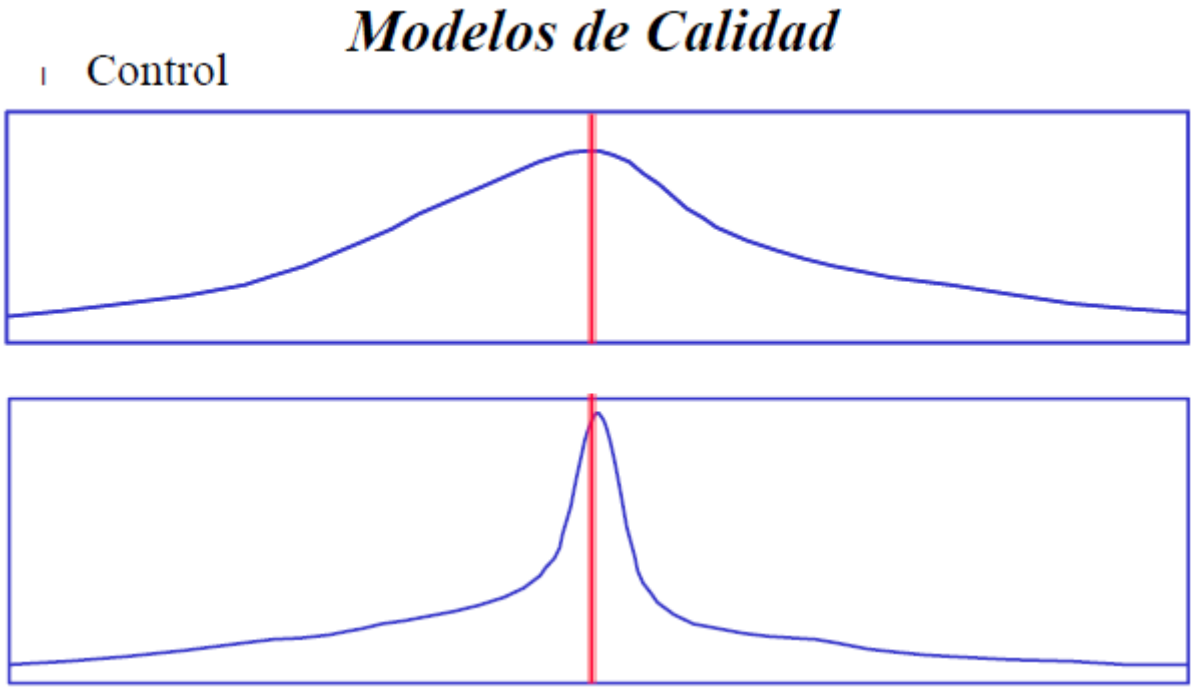
- Reducir la diferencia entre el esfuerzo real y el esfuerzo acordado.
- Reducir el número de errores funcionales y no funcionales de los sistemas en entornos de producción (tendencia a cero errores).
- Aumentar la productividad de los equipos de desarrollo (Relación productos-esfuerzo global de proyectos).

En mejora de procesos hay tres cosas que esperamos que ocurran:

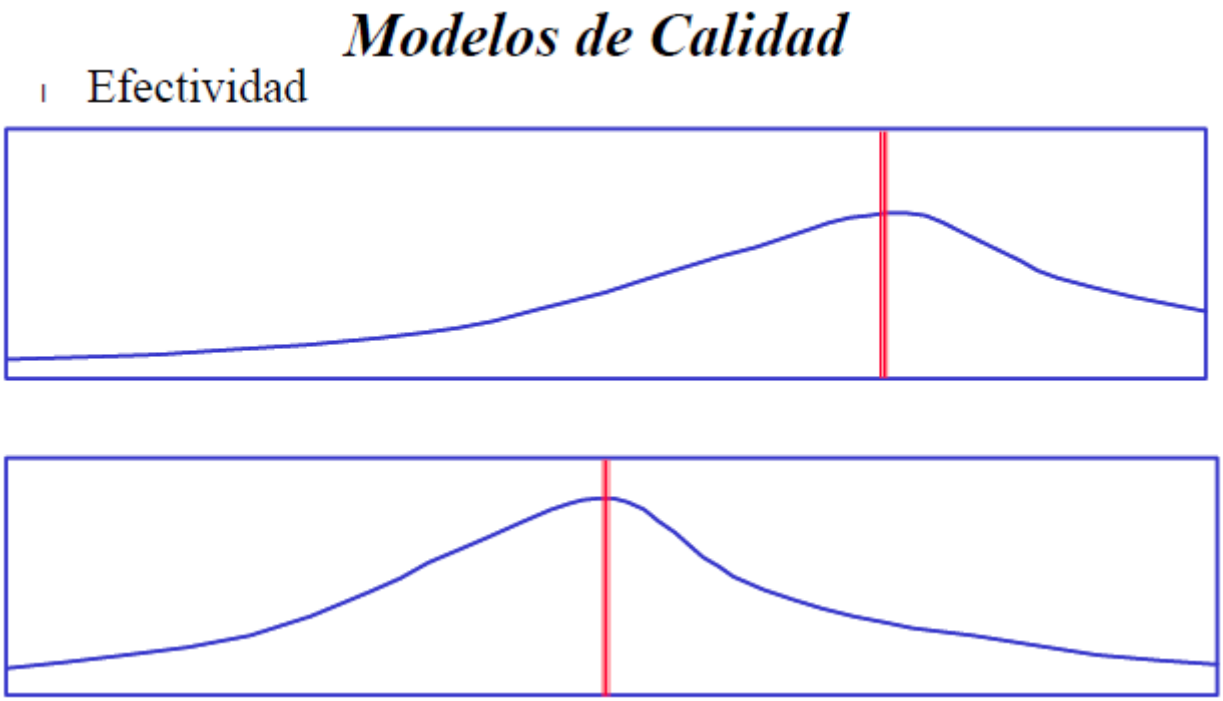
En la figura se muestra como en principio se establecen objetivos de fecha y costo que no se cumplen. La mayor parte del trabajo no está dentro del objetivo (a la izquierda de la gráfica). En la parte de abajo lo que ocurre es que el objetivo que se establece se acerca más a la realidad del desempeño del equipo de trabajo que desarrolla las tareas, volviendo más certera la estimación.



En la siguiente figura lo que esperamos que se transforme es el control. Aquí se muestra como (en la parte de arriba) una parte importante de los resultados, a pesar de estar centrados en los objetivos, se sale de los objetivos. Disminuir (abajo) esta curva significa que nuestro proceso está bajo control.



En la siguiente figura, con más certeza en la información para fijar objetivos y con más control en nuestros procesos, podemos mejorar la efectividad con objetivos más agresivos y con altas posibilidades de cumplirlos.



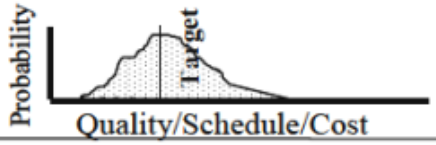
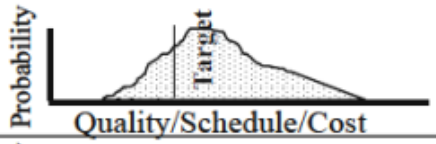
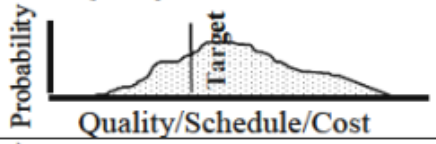
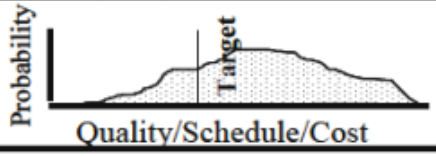
El Modelo CMM (Capacity Maturity Model)

Existen diferentes modelos de calidad entre los cuales se encuentran: CMM, SPICE, Bootstrap y Thrillium, los cuales se concentran en evaluar la capacidad de los procesos de software y la madurez de las organizaciones de software. Estos modelos constituyen un marco de referencia que permite calificar a las organizaciones de desarrollo de software.

Está comprobado que ISO9000 no es adecuado para evaluar capacidad de procesos de software, es por eso que el mismo ISO creó el proyecto SPICE.

El modelo CMM ubica a las organizaciones en uno de cinco niveles de madurez según se muestra en la figura:

- En el nivel 1 la organización es reactiva, los administradores se dedican a resolver crisis inmediatas y los tiempos calendario y los costes son excedidos básicamente porque no están basados en estimaciones reales. Cuando hay metas calendario agresivas, regularmente la funcionalidad y calidad del producto son comprometidos a fin de cumplir con las fechas. No existe un proceso definido y cuando un proyecto sale bien, no hay manera de reproducir su forma de trabajo en proyectos subsecuentes.
- En el nivel 2 ya hay un proceso definido, se tienen identificadas las entradas y salidas de cada etapa y se establecen controles en las entradas y salidas de cada etapa.
- En el nivel 3 se define el cómo de cada etapa, habiendo probado en la VIDA REAL las técnicas y descubierto la mejor forma de aplicarlas. Se identifican los controles internos necesarios para cada etapa.
- En el nivel 4 se aplican los controles internos de cada etapa y se llevan a cabo mediciones y estimaciones en base a información estadística.
- Finalmente en el nivel 5 se lleva a cabo un proceso de mejora continua REAL en el que se hace reingeniería de procesos.

Nivel	Características	Desempeño predecible
5 Optimizado	La mejora de los procesos es institucionalizada	
4 Administrado	Producto y proceso son cuantitativamente controlados	
3 Definido	Ingeniería de software y administración de procesos son definidos e integrados	
2 Repetible	Administración de proyecto existe; desempeño repetible	
1 Inicial	El proceso es informal y ad hoc; el desempeño es impredecible	

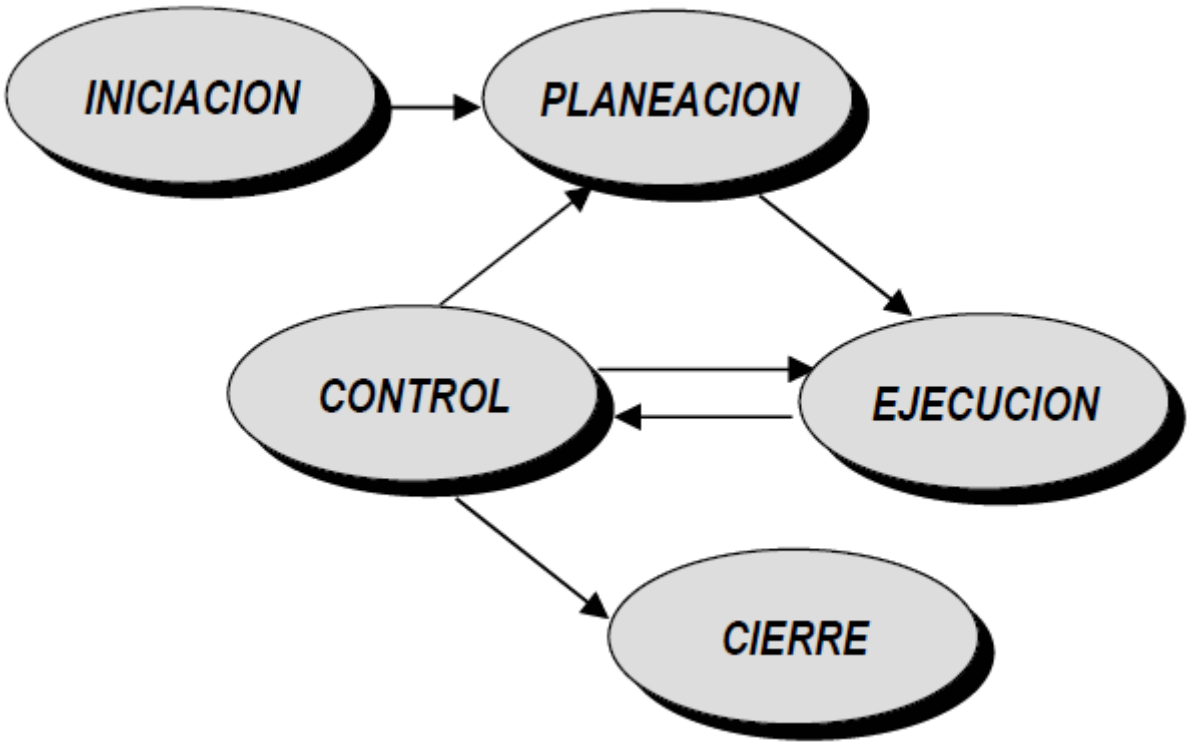
Actividades de Gestión

Son las actividades que permiten asegurar que el software se lleva a cabo a tiempo y de acuerdo a la planificación. Así como de acuerdo a los requerimientos del software:

- Planificación de las actividades del equipo de desarrollo dentro del proyecto.
- Obtener los recursos (tanto físicos como humanos) necesarios para la ejecución del proyecto.
- Organizar funciones y responsabilidades de las personas dentro del equipo de desarrollo.
- Revisar cumplimento de planes y compromisos.
- Supervisar/auditar la ejecución de las actividades dentro del desarrollo y revisar las características necesarias de los productos que se generan dentro del proyecto.

- Administrar/controlar los cambios en los productos generados dentro del proceso de desarrollo.
- Medir y registrar el desempeño durante la ejecución del proyecto.
- Anticipar posibles problemas durante la ejecución del proyecto y prevenirlos.
- Evaluar y retroalimentar el desempeño de los miembros del equipo de desarrollo.

Estas actividades se pueden agrupar como se muestra en el diagrama general de grupos de procesos:



Todos los proyectos, que se gestionan como tales, tienen una serie de fases comunes, no tanto porque se realicen tareas iguales, sino porque el objetivo de cada fase con relación al producto a obtener es común a cualquier proyecto.

Así tenemos dos grandes fases: Planificación y Ejecución. Estas fases se subdividen en otras menores. Veamos cada una de ellas por separado.

Iniciación

El origen de un proyecto suele ser difuso. Normalmente alguien identifica un problema o una necesidad. Este problema-necesidad hace muy interesante el nacimiento de un proyecto, ya que podemos observar como ante el problema que se plantea unos gerentes lo ven como un impedimento para alcanzar sus metas, mientras otros, pensando que el mismo problema también la tienen sus competidores, lo ven como una oportunidad para dar una solución correcta y posicionarse mejor en el mercado.

Ya sea visto como problema u oportunidad, lo primero que hay que hacer es obtener una descripción clara de éste. La pregunta clave a responder es: *¿Cuál es el problema, o dónde está la oportunidad?* Evidentemente aquí hay que trabajar con los usuarios, directores de empresa y clientes, pues ellos son los que conocen su negocio y será de ellos de quien tendremos que obtener la información para responder a esta pregunta.

La definición del problema suele ocupar muy poco tiempo, por esto muchas veces no se le da la importancia central que tiene. Hay que tener en cuenta que todo el proyecto se basará en esta definición y es mejor que quede clara. La definición del problema debe ser revisada por todos los implicados en el problema: usuarios, directivos y clientes.

Normalmente al definir el problema debemos hurgar en la organización, sus objetivos y fines. También debemos, una vez clarificado el problema, identificar los beneficios que se obtendrán si lo solucionamos. Hay que evitar “las soluciones en busca de un problema”, es decir cuando alguien ha visto una aplicación en marcha, o un sistema, y quiere algo similar. Muchas veces se esconde la idea intuitiva de que aquello resolverá un problema o generará una oportunidad. Lo mejor es sacar a flote el problema o la oportunidad y entonces definirlo en términos claros.

También es peligrosa la situación en la que los únicos interesados en el problema y su solución son los implicados en el proyecto. Muchas veces los técnicos desean aplicar nuevas técnicas o herramientas y organizan un proyecto en torno a éstas.

En todo caso lo que se debe hacer es buscar en la empresa, identificando alguna aplicación que no sea compleja y que sea útil a los objetivos de la misma.

Los siguientes puntos nos dan una idea de la forma de pensar, así como las tareas a realizar durante esta fase:

- Estudiar el sistema actual.
- Discutir y analizar lo que se desea obtener.
- Clarificar las áreas de la empresa que se verán afectadas.
- Definir el problema y sus componentes, aclarando: qué es fundamental, qué es deseable y qué es opcional.
- Visualizar el producto o sistema a proporcionar, así como su adaptación a la organización.
- Identificar al responsable del proyecto.
- Crear una declaración clara de lo que se va a hacer.
- Obtener el sí de los implicados: “Sí, tenemos exactamente ese problema”.

En todas las fases y en esta de forma especial se debe estimar los costes previsibles del proyecto y, sobre todo, el coste de la siguiente fase, la planificación.

En muchas organizaciones, una vez definido el problema, éste se añade a la lista de los problemas pendientes de resolución. Así un comité de dirección selecciona el próximo problema a resolver, o sistema a desarrollar.

Planificación

El objetivo de toda planificación es la de clarificar el problema a solucionar, definir el producto a obtener, o servicio a proporcionar, estimar los costes económicos en que va a incurrir, así como los recursos humanos y de cualquier otro tipo que se requieran para alcanzar la meta.

La función principal es la de atender a las necesidades que aparecerán a lo largo del desarrollo, anticipando el curso de las tareas a realizar, la secuencia en que se llevarán a cabo, los recursos y el momento en que serán necesarios. Hay que tener en cuenta que normalmente hay más bienes o servicios que desearíamos obtener, que recursos disponibles para obtenerlos, por lo que las empresas deben seleccionar entre varias alternativas. Así una mala definición de un proyecto puede provocar que la empresa comprometa sus recursos en un bien del que hubiera podido prescindir en favor de un sustituto más económico.

La planificación del proyecto es la fase en la que se deberán identificar todas las cosas necesarias para poder alcanzar el objetivo marcado. En esta fase se han de concretar los tres cimientos sobre los que se apoyará el desarrollo de todo el proyecto, estos son:

- Calidad: viene dadas por las especificaciones.
- Coste económico: valorado en el presupuesto.

- Duración: asignada en el calendario de trabajo.

Así como en la fase anterior nos centrábamos en identificar el problema, aquí tenemos que identificar diferentes soluciones y los costes asociados a cada una de ellas.

Aunque muchos autores separan el análisis de la aplicación de la propia planificación, por entenderse que la primera es una técnica, mientras que la planificación es una tarea de gestión, cronológicamente se han de realizar de forma simultánea, aunque, se debería partir de una especificación seria del problema, antes de planificar las tareas, costes y recursos necesarios para desarrollar la aplicación.

Otro asunto es que cada trabajo que se realiza se debe planificar antes de acometerlo. Así, antes de realizar el análisis se deberá hacer una planificación de los trabajos asociados a éste, pero difícilmente se podrá realizar la planificación de todo el proyecto.

Las tareas a realizar para planificar el proyecto, las podemos agrupar en:

- Estimar el tamaño de la aplicación a desarrollar.
- Estimar el coste en recursos humanos.
- Identificar las tareas a realizar.
- Asignar recursos a cada tarea.
- Crear un calendario de las tareas.
- Realizar un estudio económico.
- Reunir todo en un documento, Estudio de viabilidad.

Estas tareas se realizan de forma secuencial o iterativa entre ellas. Esta sería una iteración de las tareas:

```
Establecer las restricciones del proyecto
hacer las suposiciones iniciales de los parámetros del proyecto
while el proyecto no termina o ha sido cancelado loop
Describe la planificación de tiempos del proyecto
Inicia las actividades de acuerdo a la planificación
Espera (a que se lleve a cabo el desarrollo)
Revisa el progreso del proyecto
Revisa los parámetros estimados del proyecto
Actualiza la planificación del proyecto
Renegocia las restricciones del proyecto y los tiempos de entrega
if (aparecen problemas) then
inicia una revisión técnica y sus posibles soluciones
end if
end loop
```

Las actividades en un proyecto deben ser organizadas para producir resultados tangibles para que la administración pueda juzgar el progreso.

Los “Milestones” son los puntos finales de alguna actividad. Los “deliverables” son los resultados del proyecto que serán entregados a los clientes. El proceso de “cascada” permite una definición precisa de los “milestones”.

Ejecución

En esta fase, se trata de llevar a cabo el plan previo. Se verá fuertemente influida por la planificación. Una mala planificación, llevará a una mala ejecución, ya que si se planifica que costará menos tiempo del real, los usuarios presionarán a los desarrolladores, con lo que éstos trabajarán en peores condiciones, del mismo modo, si se planifica un coste inferior, los administradores de la empresa presionarán al personal del proyecto, con lo que éstos trabajarán con más estrés.

Esta fase se caracteriza fundamentalmente porque en ella se ha de organizar el equipo de desarrollo, los mecanismos de comunicación, la asignación de roles y de responsabilidades a cada persona. Tareas fundamentales son:

- Identificar las necesidades de personal, que aunque ya venían de la fase de planificación, habrá que ajustarla a las disponibilidades actuales.
- Establecimiento de la estructura organizativa.
- Definir responsabilidades y autoridad.
- Organizar el lugar de trabajo. En muchas ocasiones el comienzo de un proyecto tiene tareas como instalación de equipamientos, acondicionamiento de locales, …
- Puesta en funcionamiento del equipo. Cuando las personas que van a trabajar en un proyecto no se conocen, es oportuno organizar reuniones más o menos informales para que se conozcan, esto evitará malentendidos y conflictos durante la ejecución del proyecto.
- Divulgación de los estándares de trabajo y sistemas de informes. Al comenzar el proyecto, las personas están más receptivas que cuando se encuentran en un trabajo rutinario o cuando el objetivo se transforma en algo obsesivo. Ésta es una razón de peso para introducir los nuevos métodos de trabajo. Es posible que sea el cliente el que marque los estándares.

Control

En este momento, ya tenemos el proyecto con su calendario, etc, las especificaciones claras, los recursos y personas en situación de trabajo. Las personas deben llevar a término cada una de las tareas que se les ha asignado en el momento que se le haya indicado. El objetivo principal en esta fase es; establecer visibilidad adecuada del progreso real del proyecto, de manera que la administración pueda tomar acciones efectivas cuando la ejecución del proyecto de software se desvía significativamente de los planes.

Por su parte el responsable del proyecto debe realizar las siguientes actividades:

- Revisar cumplimiento de planes y compromisos.
- Tomar medidas del rendimiento.
- Revisar los informes que le llegan de los empleados.
- Mantener reuniones para identificar los problemas antes de que aparezcan.
- Administrar / controlar los cambios en los productos generados dentro del proceso de desarrollo.
- En caso de desviaciones poner en práctica las acciones correctivas necesarias.
- Coordinar las tareas.
- Motivar y liderar a los empleados.
- Recompensar y disciplinar.

Cierre

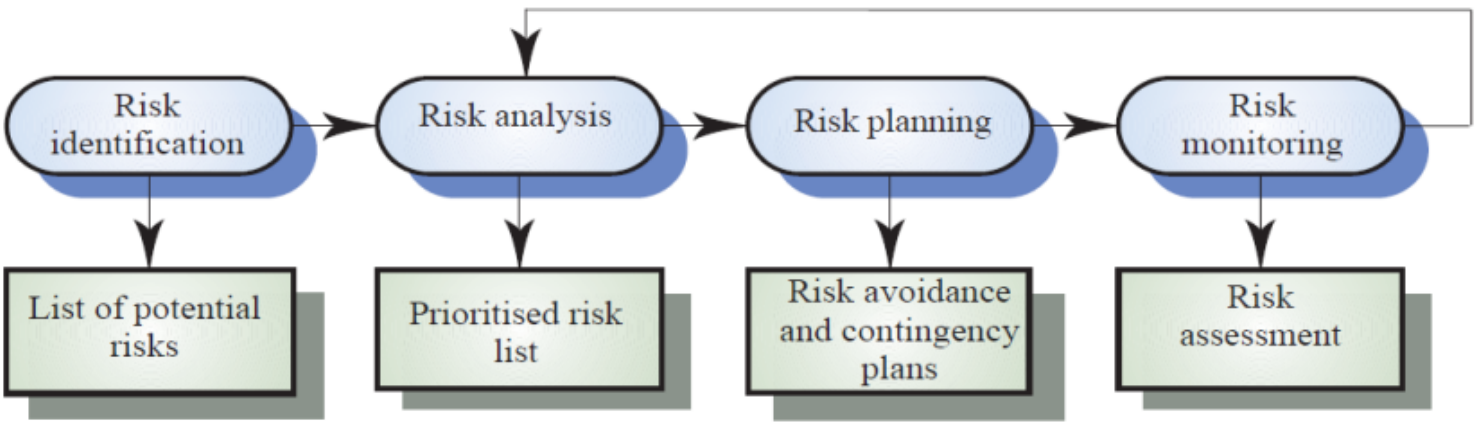
Ésta fase es la opuesta a la de puesta en marcha. En ésta se trata primero dar por finalizado el proyecto y entregar el producto, o dejar de producir el servicio encomendado.

Las actividades a realizar son las siguientes:

- Hacer entrega definitiva del producto al cliente.
- Revisar las desviaciones del proyecto, identificar causas e indicar formas diferentes de actuación en futuros proyectos.
- Reasignar el personal a los nuevos proyectos o reintegrarlos en los departamentos de partida.
- Es interesante documentar las relaciones entre los empleados para futuros proyectos.

Gestión de Riesgos

Gestión de riesgos concierne con la identificación de riesgos y la escritura de planes para minimizar el efecto de estos en el proyecto.



Un riesgo se relaciona con la probabilidad de que ocurra alguna circunstancia adversa al proyecto:

- Los riesgos de un proyecto afectan a la planificación o a los recursos.
- Los riesgos del producto afectan a la calidad o al desempeño del software por desarrollarse.
- Los riesgos del negocio son aquellos que afectan a la organización que desarrolla el software.

Identificación de los Riesgos

Identifica riesgos en el proyecto, en el producto y en el negocio:

- Riesgos en la tecnología.
- Riesgos en la gente.
- Riesgos organizacionales.
- Riesgos en los Requerimientos.
- Riesgos de estimación.

Análisis de Riesgos

Calculo de la posibilidad de que ocurran estos riesgos y de sus consecuencias:

- Determina la probabilidad y la seriedad de cada riesgo.
- Las probabilidades pueden variar entre muy alta, alta, moderada, baja o muy baja.
- Los efectos de los riesgos pueden ser: catastróficos, serios, tolerables o insignificantes.

Planificación de Riesgos

Trazar planes para evitar o minimizar el efecto de los riesgos. Considera cada riesgo y desarrolla una estrategia para manejarlo:

- Estrategias de evasión: La probabilidad de que el riesgo que presente se minimizara.
- Estrategias de minimización: El impacto del riesgo en el producto o en el proyecto se reducirá.
- Planes de contingencia: Si el riesgo se presenta, el plan de contingencia se encarga de tratar este riesgo.

Monitorización de Riesgos

Monitorizar los riesgos durante el proyecto:

- Determina regularmente cada riesgo identificado y decide si es probable o no que se presente.
- Determina si los efectos que produciría el riesgo ha cambiado.
- Cada riesgo clave debe discutirse en las reuniones de avance del proyecto.

Desarrollo en Fases

El proceso de desarrollo de software no es solamente escribir líneas de código, compilar y ejecutar. Lo anterior es sólo una etapa (importante) de dicho proceso. En un proceso, se debe definir quién hace qué cosa cuando y cómo para alcanzar un cierto objetivo. En la ingeniería de software el objetivo principal es construir un producto de software o mejorar alguno ya construido, tomando en cuenta los requerimientos de los clientes (usuarios). Un proceso, provee de una guía para el desarrollo eficiente de un software de calidad. Tal proceso es una guía para todos los participantes en el desarrollo (usuarios, desarrolladores, responsables de proyecto, etc.) y permite construir software más ordenado y con un tiempo de vida relativamente largo.

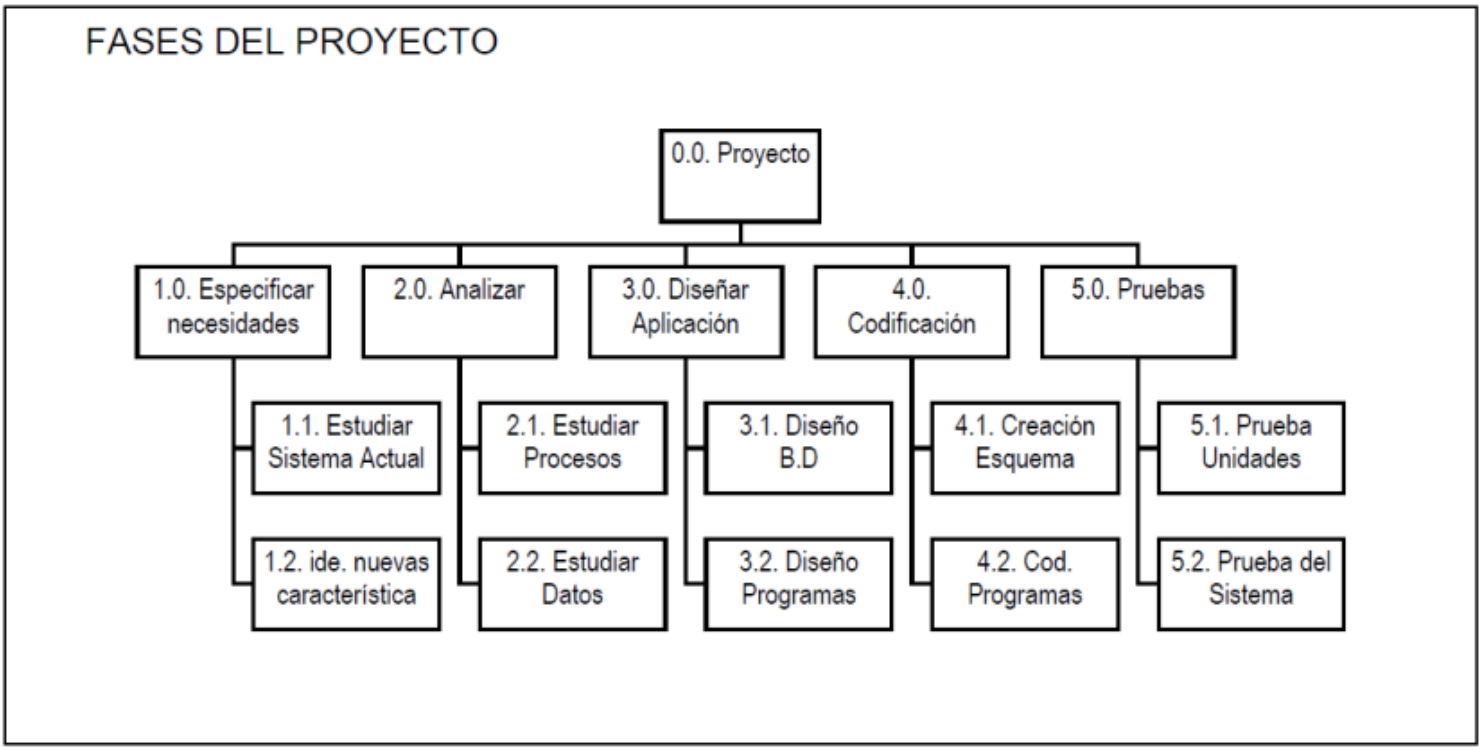
Para realizar un proyecto, empezaremos por ver cuales son los objetivos que queremos alcanzar y luego pensaremos qué cosas tenemos que hacer para alcanzar estos fines. Esta descomposición pasará por identificar las fases de nuestro proyecto y el esfuerzo a aplicar en cada una de ellas. A su vez estas fases se descompondrán en tareas. También tendremos que marcar unos puntos (hitos) de control que nos permitan saber si el proceso va de acuerdo a lo previsto.

Normalmente todas las fases y tareas terminan en la generación de uno o varios documentos. A éstos se les llama *entregables*. Este nombre se debe a que pasan de manos

del desarrollador a manos del controlador del proyecto o cliente. En los proyectos informáticos se suele asociar los hitos a la consecución de un entregable.

Descomposición en Actividades del Proyecto (WBS)

Empezaremos por ver la herramienta que se utiliza a la hora de descomponer y documentar el trabajo de un proyecto, como un conjunto de tareas. Habitualmente se le conoce como WBS (Work Breakdown Structure) que literalmente significa estructura de descomposición del trabajo. Es un método de representar de forma jerárquica los componentes de un proceso o producto. Puede ser utilizado para documentar la descomposición de un proceso, la descomposición de un producto, o de forma híbrida.



Entregables de un Proyecto Informático

Los entregables son: “Productos que, en un cierto estado, se intercambian entre los clientes y los desarrolladores a lo largo de la ejecución del proyecto informático”.

Los entregables se clasifican como relativos al objetivo y relativos a la gestión del proyecto. Son relativos al objetivo todos aquellos documentos que hacen referencia exclusivamente al sistema de información y al subsistema informático en desarrollo. Pertenecen a este conjunto los requisitos del sistema, la especificación del sistema, la documentación del diseño, el código fuente, los programas ejecutables, los manuales de usuario, etc.

Los entregables relativos a la gestión del proyecto hacen referencia a aquellos documentos que se refieren a la situación en que se encuentra un proyecto, previsiones de costes, gastos realizados, informe sobre entornos de trabajo, etc, siendo su objetivo el poder controlar el proyecto. Pertenecen a esta clase la planificación del proyecto, los presupuestos, los documentos de control de la planificación o de la calidad, los estudios de riesgos durante el desarrollo, etc.

Se deberá definir de forma clara el conjunto mínimo de entregables necesario para dar por terminada cada fase de desarrollo. Aunque algunos entregables se desarrollan a lo largo de varias tareas.

Los entregables nos proveen de:

- Un conjunto de componentes que formarán el producto una vez finalizado el desarrollo.
- Los medios para medir el progreso y la calidad del producto en desarrollo.
- Los documentos necesarios para la siguiente etapa.

Entregables más Usuales

Dado que como hemos visto los entregables juegan un papel central en el desarrollo de un subsistema informático, vamos a listar los más importantes:

- Estudio de viabilidad:
  - Descripción breve del sistema propuesto y sus características.
  - Descripción breve de las necesidades del negocio en el sistema propuesto.
  - Propuesta de organización del equipo de desarrollo y definición de responsabilidades.
  - Estudio de los costes, que contendrán estimaciones groseras de planificación y fechas, tentativas, de entrega de los productos.
  - Estudio de los beneficios que producirá el sistema.
- Análisis:
  - Captura de requisitos:
    - Análisis del sistema actual (si existe).
    - Requisitos nuevos de los usuarios.
    - Descripción del sistema propuesto.
  - Especificación del sistema:
    - Descripción del sistema (DFD’s, etc.).
    - Requisitos de datos.
    - Requisitos de telecomunicaciones.
    - Requisitos de hardware.
    - Plan de pruebas de integración.
  - Diseño:
    - Descripción detallada del sistema, contendrá:
      - Programas, módulos reutilizables y objetos.
      - Ficheros y bases de datos.
      - Transacciones.
      - Diccionario de datos.
      - Procedimientos.
      - Carga del sistema y tiempos de respuesta.
      - Interfaces, tanto humanos como de máquinas.
    - Descripción de los controles del sistema propuestos.
    - Diseños alternativos recomendados.
    - Estándares de programación y diseño de programas, recomendados.
    - Técnicas de implementación recomendadas: codificación propia, compra de paquetes, contratación externa, etc.
    - Plan de pruebas de programas.
  - Codificación:
    - Documentación del diseño final del sistema y de cada programa.
    - Diagramas definitivos del sistema y de los programas.
    - Descripción detallada de la lógica de cada programa.
    - Descripción de las Entradas y Salidad (ficheros, pantallas, listados, etc.).
    - Listado de los programas, conteniendo comentarios.
    - Cadenas de ejecución si es necesario (JCL, scripts, etc).
    - Resultado de las pruebas de cada unidad.
    - Resultado de las pruebas de cada programa.
    - Resultado de las pruebas de la integración.
    - Guía para los operadores del sistema.

        - Programa de entrenamiento de los operadores.
        - Manual de usuario del sistema.
    - Pruebas:
        - Plan de pruebas del sistema (actualizado).
        - Informe de los resultados de las pruebas.
        - Descripción de las pruebas, el resultado esperado, resultado obtenido y acciones a tomar para corregir las desviaciones.
    - Instalación:
        - Planes detallados de contingencias de explotación, caídas del sistema y recuperación.
        - Plan de revisión post-instalación.
        - Informe de la instalación.
        - Carta de aceptación del sistema.
    - Mantenimiento:
        - Listado de fallos detectados en el sistema.
        - Listado de mejoras solicitadas por los usuarios (si no dan lugar a nuevos proyectos).
        - Traza detallada de los cambios realizados en el sistema.
        - Actas de las revisiones regulares del sistema y aceptación de los niveles de soporte.

A todos estos documentos hay que añadir en todas las fases documentos con la estimación y planificación de la próxima fase y del resto del proyecto. También habrá que ir actualizando el índice de todo el material relacionado.

Descomposición en Fases del Desarrollo de una Aplicación

La descomposición por fases (actividades) se basa en referencias históricas de la empresa que asocian una cantidad media de horas de trabajo a una actividad concreta, de modo que dado un proyecto concreto podemos estimar la cantidad de esfuerzo que se dedicará a esa actividad. En ésta se ha de tener en cuenta el tipo de proyecto, el lenguaje de desarrollo y la madurez de la organización.

Podemos plantear la descomposición desde el enfoque de entregables y asociar las tareas a la producción de un entregable concreto. Este enfoque tiene la ventaja de que la culminación de una tarea indica que ha concluido un producto y viceversa. Dado que, como veremos, no es aconsejable el tener tareas que duren más de una semana, se plantean problemas con algunos entregables que cuestan más.

El planteamiento de descomponer por procesos o actividades puede resultar más natural en algunos casos. Es más fácil conseguir tareas acotadas en el tiempo. Tiene la desventaja de que el proyecto no será tan fácil de controlar ya que en muchos casos será la palabra de los realizadores la única constancia de que la tarea está terminada o al “90%”.

En cualquier caso, los proyectos se planifican con dos horizontes, el de la próxima fase y el del proyecto completo. En el horizonte de la próxima fase se realiza con mayor nivel de detalle, mientras que según se alejan las fases se aplica un menor nivel de detalle.

La descomposición del proyecto con mayor nivel de refinamiento no puede basarse en datos recogidos de forma analítica, sino que hace falta una aportación personal de los miembros del equipo de trabajo, tanto para identificar tareas como para asignarles esfuerzos. Se suele aconsejar el trabajo en grupo donde todos puedan aportar sus conocimientos y experiencias previas.

Hay que tener en cuenta que si identificamos las tareas y se las imponemos a los desarrolladores, éstos funcionarán en una situación de sumisión lo que puede tener efectos perniciosos tanto para los plazos de entrega como para la calidad del software. Por otra parte el dejar que sean los propios desarrolladores los que identifiquen tareas y

recursos, dentro de un marco razonable (puntos de función) les llevará a una situación de compromiso personal, pasando a interiorizar los objetivos y como consecuencia obtendremos mejores resultados.

La tarea fundamental de los desarrolladores es escuchar a los clientes o usuarios y traducir sus requisitos a un lenguaje comprensible por la máquina, de modo que el subsistema informático se adapte a las necesidades expresadas. Así para cualquier tarea podremos encontrar las siguientes subtareas:

- Documentarse, Buscar o Investigar.
- Organizar, Escribir Documentos.
- Verificar, Comprobar.
- Revisar, Actualizar Documentos.
- Entregar, Finalizar.

Además de lo anterior hay que tener en cuenta que al ir desarrollando el sistema obtenemos información que nos será útil a la hora de identificar nuevas tareas. Así, el análisis estructurado nos provee de una descomposición del proyecto por productos: transacciones, archivos, entradas, salidas, etc. El Diseño de programas nos descompone el sistema por módulos, el Diseño de BD descompone por tablas, archivos, etc, y los diseños de interfaz de pantallas, listados, mensajes, etc. Así, por ejemplo, una entrada puede ser que requiera de una reunión con el usuario, un estudio de ésta y la posterior presentación y aprobación de la propuesta a desarrollar.

Tareas Usuales de un Proyecto Informático

- Estudio de viabilidad:
  - Analizar el sistema propuesto y escribir una descripción.
  - Definir y documentar posibles tipos de sistemas.
  - Hacer un análisis de coste de sistemas similares.
  - Hacer una estimación del tamaño del sistema, la planificación y los costes (tener en cuenta los entregables más importantes).
  - Definir cualitativa y cuantitativamente los beneficios del sistema propuesto.
  - Realizar una planificación inicial del plazo de recuperación de la inversión.
  - Realización de una estimación detallada de costes, planificación, recursos, etc, de la siguiente fase (Análisis).
  - Asignar director del proyecto.
  - Composición del documento de estudio de viabilidad.
  - Presentación del documento de viabilidad a la dirección para su aprobación.
- Análisis:
  - Captura de requisitos:
    - Definir el ámbito del sistema propuesto.
      - Funciones
      - Usuarios
      - Restricciones
    - Entrevista a todos los usuarios propuestos y actuales:
      - Determinar:
        - Utilización del sistema actual.
        - Deficiencias del sistema actual.
        - Requisitos nuevos del sistema.
      - Documentar:
        - Descripción del sistema actual.
        - Deficiencias del sistema actual.
    - Producir el documento de requisitos del nuevo sistema:
      - Incluir:
        - Requisitos del usuario priorizados.
        - Resoluciones sobre las deficiencias del sistema actual.

        - Producir una lista de los beneficios tangibles e intangibles (un refinamiento de la lista del estudio de viabilidad).
        - Realización de una estimación detallada de costes, planificación, recursos, etc, de la siguiente fase (Especificación del sistema).
        - Producir una estimación revisada de costes, planificación, recursos, etc, para el resto del proyecto.
        - Producir el documento de definición de requisitos, esta tarea incluye la construcción de un prototipo.
        - Realizar una revisión final del documento de requisitos.
        - Toma la decisión de continuar o no con el proyecto.
        - Definir las responsabilidades en la próxima fase para el director, miembros del equipo de desarrollo y otros.
    - Especificación del sistema:
        - Definir el tipo de sistema propuesto: Transformar las restricciones físicas, ambientales y operacionales a características del sistema. Por ejemplo: ¿Sistema basado en transacciones? ¿Distribuido o centralizado?¿Estaciones de trabajo o terminales?
        - Esquematizar el sistema propuesto: Transformar los requerimientos del usuario de la fase anterior en unas especificaciones funcionales (DFD, Organigramas, etc).
        - Construir el diccionario de datos (DD): Describir todos los elementos del DFD incluyendo funciones y datos; asegurarse de que todas las relaciones inter-funcionales y entre datos sean documentadas. Si existe DD de la empresa, hacerlo compatible con el que estamos realizando.
        - Revisar y expandir el análisis de coste beneficio: Actualizarlo con la información nueva y verificar que los beneficios esperados se mantienen y que el plazo de recuperación de una inversión sigue siendo aceptable.
        - Realización de una estimación detallada de costes, planificación, recursos, etc, de la siguiente fase (Diseño del sistema).
        - Producir una estimación revisada de costes, planificación, recursos, etc, para el resto del proyecto.
        - Producir el documento de especificación del sistema.
        - Realizar una revisión final del documento de especificación del sistema.
        - Tomar la decisión de continuar o no con el proyecto.
        - Definir las responsabilidades en la próxima fase para el director, miembros del equipo de desarrollo y otros.
- Diseño:
    - Producir el diseño global del sistema, contendrá:
        - Definir los programas y sus principales funciones.
        - Definir los principales flujos de datos entre programas y funciones.
        - Diseñar el esquema de datos lógico y físico.
        - Definir las fronteras con paquetes software, si existen.
        - Definir los entornos de hardware y software, proponiendo alternativas.
    - Localización de paquetes software: Buscar paquetes software apropiados que puedan implementar parte, o toda la funcionalidad requerida del sistema de forma rentable y que, si se implementa, ofrezca un entorno compatible con los objetivos de la organización. (Puede realizarse antes del diseño, o de forma simultánea a la tarea anterior).
    - Desarrollar un diseño detallado del sistema, para cada alternativa de diseño planteada:
        - Crear una descripción narrativa detallada del diseño para todo el sistema y cada una de sus partes (programas, funciones y datos).
        - Actualizar el diccionario de datos.
        - Definir los componentes hardware específicos (Capturadores de datos, sistemas de comunicación, etc) y sus funciones.
        - Validar el diseño con las especificaciones del sistema.
        - Documentar el entorno hardware y software necesarios para esta alternativa.

    - Revisar y expandir el análisis de coste beneficio para cada alternativa:
        - Actualizar con la información nueva.
        - Verificar que los beneficios esperados se mantienen y que el plazo de recuperación de la inversión sigue siendo aceptable.
    - Evaluar las alternativas de diseño, para cada alternativa, documentar:
        - Requerimientos de usuario que se alcanzan con esta alternativa.
        - Nivel de aceptación esperado de los usuarios.
        - Realización de una estimación detallada de costes, planificación, recursos, etc, de la siguiente fase (Codificación) con esta alternativa.
        - Producir una estimación revisada de costes, planificación, recursos, etc, para el resto del proyecto.
        - Alternativa recomendada.
    - Desarrollo de un plan de test del sistema:
        - Crear datos de entrada del test.
        - Producir el listado de los resultados esperados.
        - Producir el listado de los criterios de test.
        - Desarrollar la planificación de test del sistema.
    - Desarrollar un plan de test diferenciado para cada alternativa.
    - Identificar las necesidades de entrenamiento y documentación de los usuarios. Definir las guías de:
        - Documentación completa de usuario.
        - Manuales de operador.
        - Documentos y planificación de formación para usuarios y operadores.
    - Producir el documento de diseño del sistema.
    - Realizar una revisión final del documento de diseño del sistema.
    - Tomar la decisión de continuar o no con el proyecto.
    - Recomendar una alternativa.
    - Definir las responsabilidades de la próxima fase para el director, miembros de los equipo de programación y test, así como de otros implicados.
- Codificación:
    - Producir un plan de trabajo:
        - Creación de la lista detallada de tareas necesarias para realizar la codificación y test de todos los componentes del sistema.
        - Producir una planificación para las tareas anteriores con las fechas más tempranas y más tardías, así como la asignación de responsabilidades.
        - Instaurar los procedimientos para recoger los progresos y estados del proyecto.
        - Instaurar los procedimientos para recoger tiempos, si resulta apropiado.
        - Obtener la aprobación del plan de trabajo por parte de la dirección.
    - Realización del diseño detallado de cada programa:
        - Diseñar detalladamente los diagramas:
            - De estructura de los programas.
            - De estructura de los ficheros.
            - Pantallas, informes, y otras composiciones.
            - Esquemas de la base de datos.
            - Composición de las tablas y sus diseños.
        - Pseudocódigo de la lógica del programa (Dependerá de los métodos de diseño utilizados).
    - Codificar, documentar y pasar los test en cada programa:
        - Codificar el programa.
        - Realizar las pruebas de unidad, hasta que los programas se adapten a las especificaciones descritas en las etapas anteriores.
        - Actualizar todo lo necesario en el sistema y en el DD de la organización.
    - Realizar el test de integración:
        - Poner todos los programas probados en la librería de pruebas de integración.
        - Realizar el test de integración de cada programa.
        - Documentar todos los resultados del test de integración.

    - Terminar los manuales de operador y usuario, así como los de formación.
    - Realización de una estimación detallada de costes, planificación, recursos, etc, de la siguiente fase (Prueba del sistema).
    - Producir una estimación revisada de costes, planificación, recuros, etc, para el resto del proyecto.
    - Confeccionar el documento de diseño de programas y codificación.
    - Realizar revisiones del documento de diseño de programas y codificación.
    - Obtener los resultados finales de la integración completa del sistema y de las pruebas de integración.
    - Definir las responsabilidades en la próxima fase para el director, miembros del equipo de test, así como de otros implicados.
- Pruebas:
    - Realizar el test del sistema:
        - Hacer el test de sistema de acuerdo al documento de test del sistema.
        - Verificar la operatividad de los manuales de usuario y operador, utilizándolas en los cursos de formación de los usuarios y operadores que realicen el test del sistema.
        - Verificar los documentos de entrenamiento de usuarios y operadores, utilizándolos en los cursos de formación de los usuarios y operadores que realicen el test del sistema.
        - Documentar completamente los resultados del test del sistema.
    - Revisar la planificación de instalación:
        - Disponibilidad de los recursos.
        - Revisión de los factores de contingencia que puedan afectar a la instalación:
            - Procesos especiales de final de mes y fin de año.
            - Vacaciones y fiestas.
        - Disponibilidad de soporte por parte de otros proveedores.
        - Revisión final del calendario de instalación.
    - Esbozar el plan de contingencia ante caídas del sistema:
        - Criterios para las caídas.
        - Identificación de recursos para contingencias.
        - Horario para recuperaciones o abandonos.
    - Desarrollar un acuerdo de nivel de servicio:
        - Criterios de rendimiento de usuario, precisión y volumen.
        - Criterios de apoyo de los proveedores:
            - Tiempo medio entre fallos.
            - Tiempo medio de reparación.
        - Criterios de calidad del sistema.
        - Frecuencia con la que se medirán los criterios.
    - Producir los documentos de test en la entrega.
    - Revisión y aprobación de los documentos de entrega.
    - Aprobación de la documentación del sistema.
        - Documentación de programas.
        - Manuales de operador.
        - Manuales de usuario.
        - Manuales de formación.
        - Documentación de ayuda.
    - Aprobación del plan de instalación.
    - Aprobación de los planes de contingencia, recuperación y caídas.
    - Finalización del sistema completamente probado:
        - Documento de finalización del desarrollo del sistema.
        - Documento de finalización de los usuarios.
        - Documento de finalización del CPD.
        - Documento de finalización de garantía de calidad.
        - Documento de finalización de finanzas.
- Instalación:
    - Instalación de hardware y software nuevo.

    - Formar a los primeros usuarios y operadores.
    - Desarrollar los planes de contingencia, recuperación y caída.
    - Desarrollar los procedimientos de mantenimiento y versiones.
    - Establecer procedimientos para:
        - Versiones regulares.
        - Versiones de emergencia.
        - Versión por configuración, si existen diferentes tipos de hardware.
    - Llevar a cabo cualquier conversión de datos necesaria.
    - Llevar a cabo la instalación del sistema nuevo a producción:
        - Instalación completa desde cero.
        - Instalación en paralelo.
        - Instalación por fases.
    - Planificar y programar las revisiones post-instalación. Establecer los criterios de:
        - Rendimiento del sistema.
        - Calidad del sistema.
        - Satisfacción del usuario.
        - Calidad y facilidad de Gestión de: manuales de usuario y operador, formación de usuarios y operadores e información y datos producidos.
        - Fluidez de la instalación.
        - Costes de desarrollo, instalación, operaciones y mantenimiento. Establecer planificación y calendario de las revisiones, asegurando la disponibilidad del personal y documentación.
    - Llevar a cabo las revisiones post-instalación:
        - Crear el informe de la revisión post-instalación.
        - Obtener la aprobación firmada de los informes de:
            - Usuarios finales del sistema.
            - Operadores del sistema.
            - Auditoria y garantía de la calidad.
            - Desarrollo de sistemas.
            - Soporte de sistemas y mantenimiento.
            - Finanzas.
        - Obtener la carta de aprobación del sistema.
    - Establecer el calendario para otras revisiones post-instalación si es necesario.
- Mantenimiento:
    - Implementar los cambios del sistema:
        - Utilizar los procedimientos de implementación de versiones.
        - Implementar versiones de emergencias.
    - Asegurarse de que el sistema continúa solucionando las necesidades de los usuarios:
        - Utilizar los acuerdos de niveles de soporte, en estos acuerdos se establecen los requerimientos de soporte y objetivos de funcionamiento:
            - Revisiones regulares de requerimientos del nivel de acuerdo.
            - Revisiones regulares de como el sistema está alcanzando sus objetivos.
        - Llevar a cabo revisiones regulares del sistema:
            - Utilizar los procedimientos y contenido de las revisiones post-instalación.

Estas tareas se han enumerado a modo de lista de comprobación, de forma que serán los desarrolladores los encargados de identificar las tareas apropiadas a cada proyecto así como los recursos necesarios, teniendo en cuenta la estimación previa del esfuerzo.

Tareas y Funciones de los Distintos Agentes

Son personas y organizaciones que participan activamente en el proyecto o cuyos intereses pueden ser afectados positiva o negativamente tanto por el resultado de la ejecución del proyecto como por su terminación exitosa.

Los principales *agentes* en cada proyecto pueden ser:

- Jefe de proyecto. Responsable de la planificación y ejecución el proyecto.
- Equipo de desarrollo. Encargado de realizar el proyecto.
- Cliente. Es el que arriesga su dinero en el desarrollo, es decir, el que pagará por el sistema.
- Usuarios. Personas que utilizarán el sistema a nivel operativo y que normalmente pertenecen al cliente. Nos dan pistas sobre el problema a nivel de funcionamiento. Son responsables de que el sistema funcione de manera eficiente.

El Jefe de Proyecto

La misión del jefe de proyecto es:

- Con el cliente:
  - Dejarlo satisfecho.
  - Incrementar su competitividad y/o desempeño interno a través de la solución que se le entregue.
- Con el negocio:
  - Lograr rentabilidad.
  - Aprovechar recursos al máximo.
- Con los recursos humanos:
  - Crecimiento profesional.
  - Satisfacción interna y externa.

Al jefe de proyecto se le concede una amplia autoridad sobre los recursos del proyecto y puede adquirir nuevos recursos ya sea dentro o fuera de la organización. Todo el personal del proyecto está bajo su autoridad mientras dure el proyecto. Debe combinar conocimiento técnico en la materia además de habilidades de dirección para poder dirigir a todo el personal del proyecto.

Las interacciones que tiene el jefe de proyecto dentro de su organización son:

- Con su equipo de trabajo.
- Con el ejecutivo de cuenta.
- Con el departamento de calidad.

Las interacciones que tiene el jefe de proyecto con el cliente son:

- Con el usuario operativo.
- Con el departamento de sistemas.
- Con el experto funcional del negocio.
- Con otras áreas.

Responsabilidades del Jefe de Proyecto

- Conocer los criterios de negociación acordados con el cliente.
- Notificar al ejecutivo de cuenta los cambios al alcance para que los renegocie adecuadamente.
- Evaluar el desempeño de cada persona en base al cumplimiento de los compromisos acordados.
- Informar a la dirección de manera justificada y en tiempo sobre la planificación de la asignación y liberación de recursos.
- Informar los requerimientos de formación, evaluaciones y vacaciones de los miembros del equipo de trabajo al director.
- Informar de avances e incidentes del proyecto.
- Asegurar que el proyecto sea rentable y productivo.

- Dar seguimiento al plan de trabajo y corregir desviaciones a tiempo.
- Asegurar la obtención de los recursos materiales indispensables para desarrollar el proyecto.

Autoridad del Jefe de Proyecto

- Organizar al equipo de trabajo como sea más conveniente y productivo.
- Definir las expectativas de crecimiento para los miembros del equipo de trabajo.
- Implementar las medidas necesarias para que las expectativas de cada persona se cumplan.
- Prescindir de los servicios de un miembro del equipo de trabajo, si hay motivos para apoyar esta decisión.

Bibliografia

- Scribd (tfandos)

Planificación del desarrollo. Técnicas de planificación. Metodologías de desarrollo. La metodología Métrica.

Técnicas de Planificación

Introducción a la Planificación del Desarrollo

El objetivo de la Planificación del proyecto Software es proporcionar un marco de trabajo que permita al gestor hacer estimaciones razonables de recursos, costos y planificación temporal. Estas estimaciones se hacen dentro de un marco de tiempo limitado al comienzo de un proyecto de software, y deberían actualizarse regularmente a medida que progresa el proyecto. Además las estimaciones deberían definir los escenarios del mejor caso, y peor caso, de modo que los resultados del proyecto pueden limitarse.

El Objetivo de la planificación se logra mediante un proceso de descubrimiento de la información que lleve a estimaciones razonables.

Existen diversos tipos de planes:

Plan	Descripción
Plan de Desarrollo	Describe la metodología a utilizar en el desarrollo del proyecto.
Plan de Calidad	Describe los procedimientos de calidad, y los estándares a utilizar en el proyecto.
Plan de Validación	Describe el enfoque los recursos y la planificación utilizada por la validación.
Plan de Mantenimiento	Predice los requerimientos de mantenimiento del sistema, los costes de mantenimiento y el esfuerzo.
Plan de Desarrollo Personal	Describe como se adquirirán y desarrollarán los conocimientos y habilidades del personal.

Aunque hay que tener en cuenta que la planificación conlleva una serie de problemas añadidos:

- Es difícil estimar la longitud y dificultad de las tareas, por lo que la estimación del coste es más difícil.
- La productividad no es proporcional al número de personas trabajando en una tarea.
- Incluir personal en un proyecto en avance, retrasa el proyecto por sobrecargas en la comunicación.
- Lo inesperado siempre sucede. Es necesario considerar siempre contingencias.

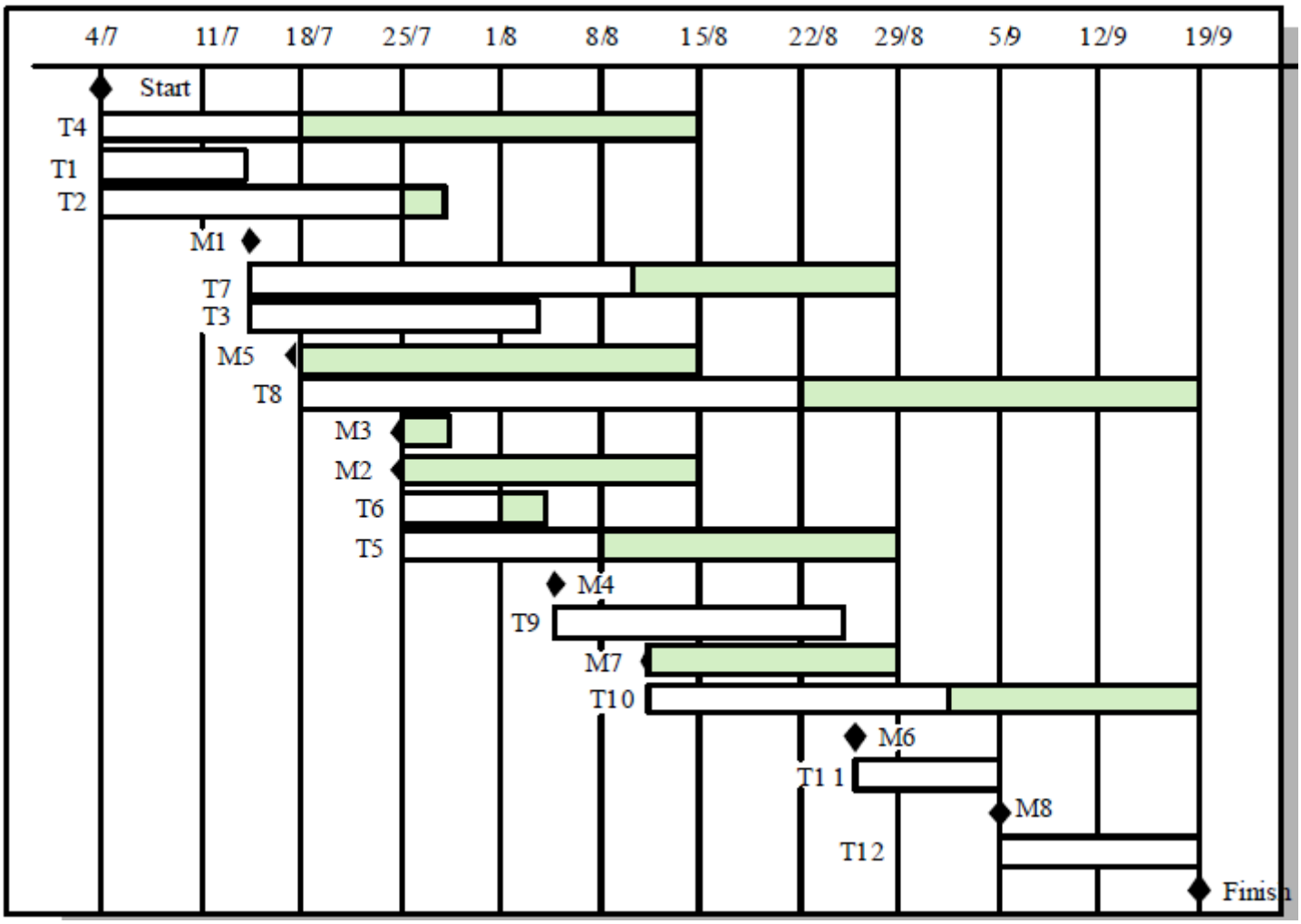
Herramientas Básicas Usadas en la Planificación

Aunque desde la antigüedad se han realizado proyectos de gran envergadura como por ejemplo la construcción de edificios públicos, guerras, viajes, etc, no es hasta principios de este siglo cuando aparece el conocido diagrama de Gantt en el que se refleja de forma esquemática las tareas, su duración y las fechas en que se deberán realizar. Trabajando sobre este diagrama el director de proyecto realizaba planificaciones y seguimiento de un proyecto.

Dada la evolución tecnológica los seres humanos cada vez abordamos proyectos más complejos, pero por otra parte creamos técnicas más evolucionadas, completas y automáticas para gestionar estos proyectos. La construcción del misil Polaris, así como la solución de los problemas en la gestión de la producción de Dunlop llevaron al desarrollo de las técnicas conocidas como PERT (Técnica para la Evaluación y Revisión de Programas) y CMP (Método del Camino Crítico) que aportan a la programación de proyectos técnicas matemáticas.

Estas técnicas surgieron de la necesidad de obtener algoritmos automatizables que ayudasen a los gestores de proyectos complejos en la construcción de calendarios (programas). En el siguiente punto veremos mediante un ejemplo como se aplica en el CPM.

Diagrama de Gantt.

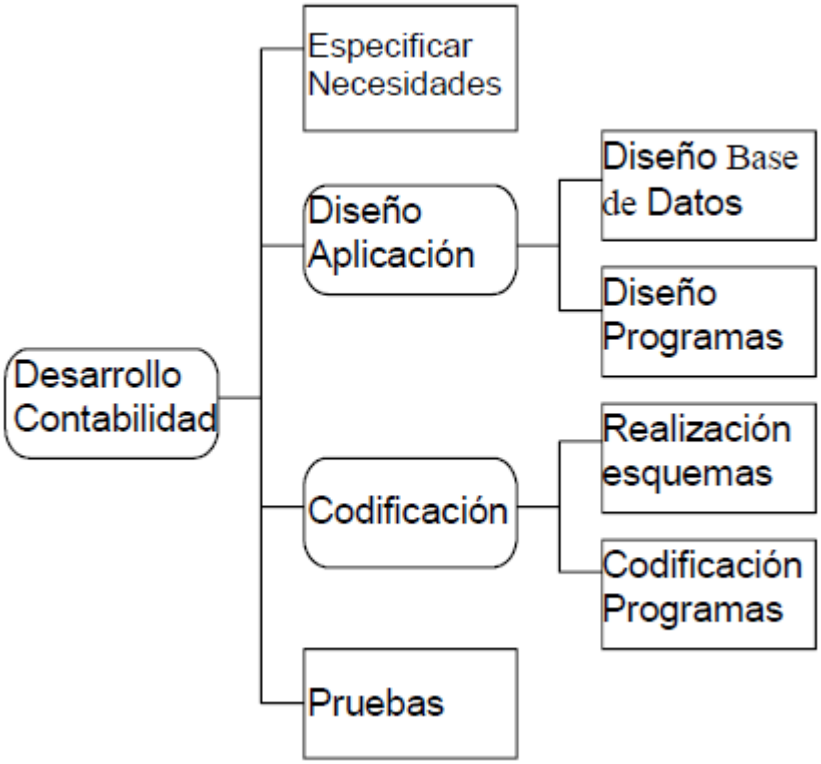


Aplicación de la Técnica CPM

El CPM se realiza sobre un proyecto en cuatro etapas, a continuación se describe cada una de ellas.

Etapa 1. Identificar las Tareas

Se deben identificar cada una de las tareas que forman parte del desarrollo del proyecto.



Si queremos realizar el proceso de forma manual, rellenaremos una ficha por cada actividad identificada. El formato de la ficha será el que se muestra en la figura.

Especificación de tarea

Número: 3.1.
Nombre: Diseño B.D.
Descripción: Se diseñara la base de datos, partiendo del modelo entidad-relación propuesto en el análisis y con el objetivo de tener un sistema funcionando sobre DB2.
Esfuerzo Estimado: 2 semanas/hombre
Entregables: Estructura de implementación de la B.D.
...: ...

Etapa 2. Añadir Recursos y Tiempos

A cada actividad se le asignarán recursos (personas, material, equipos, etc) y tiempo estimado para su realización, completando la ficha.

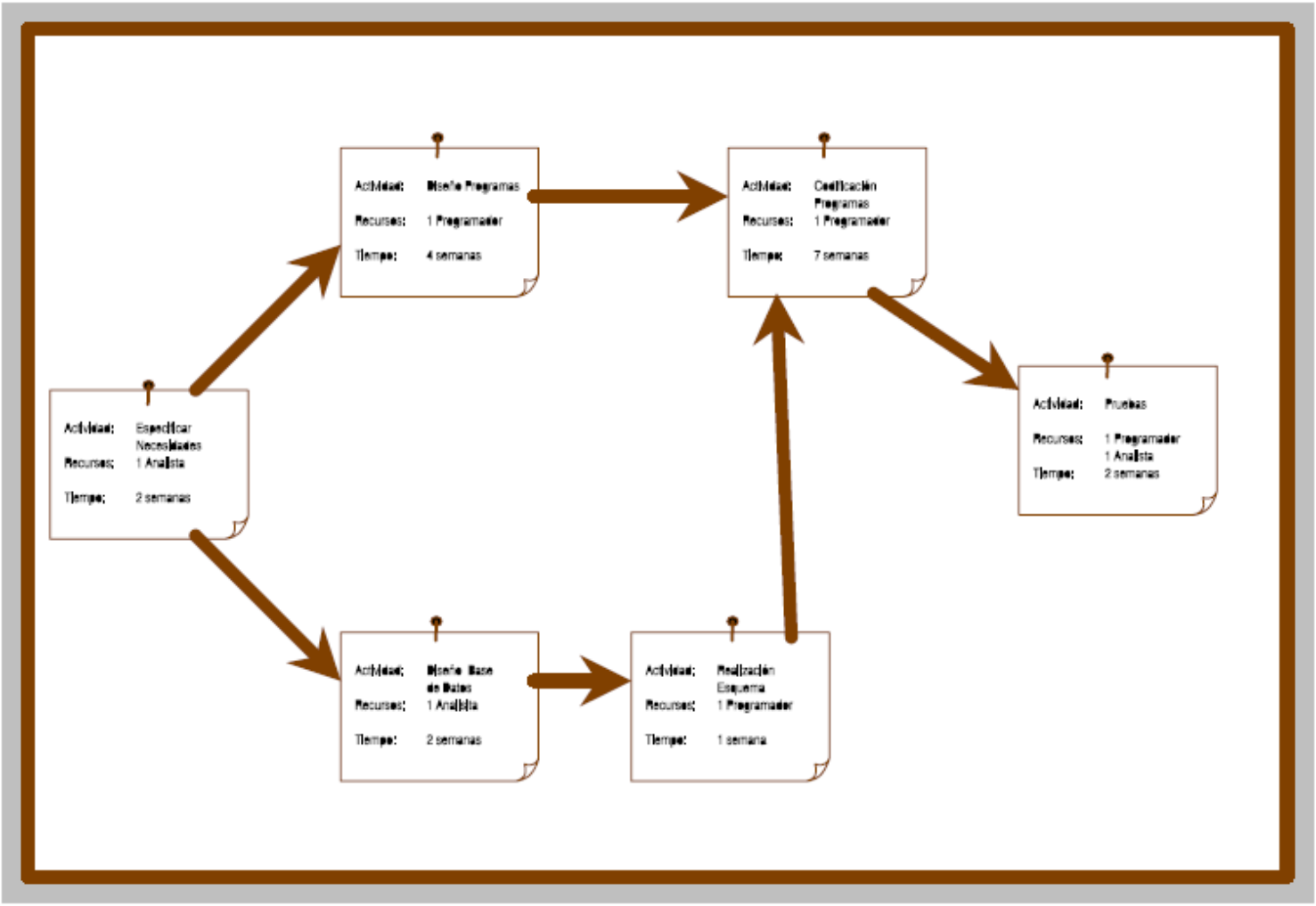
Etapa 3. Ordenar las Tareas

En esta etapa se tienen que organizar las tareas en base al orden técnico de ejecución. Así sabemos que hay que hacer las especificaciones antes de diseñar el programa. Nos podemos plantear las siguientes preguntas para ordenar las tareas:

- ¿Qué se puede hacer ahora?
- ¿Qué debe haberse hecho antes de esto?

- ¿Qué podría hacerse a la vez?
- ¿Qué debe seguir a lo que hacemos ahora?

Si se trata de calcular el Camino Crítico de forma manual será interesante pinchar todas las tareas en un tablero de corcho, señalando mediante cuerdas la ordenación de las tareas. Esta representación es conocida como *red de procedencia*, aunque su apariencia es diferente al gráfico PERT en algunos programas informáticos se describe erróneamente con este nombre.



Si tenemos un diagrama complejo, y queremos realizar los cálculos de forma manual se puede utilizar el método que se describe a continuación.

Este diagrama se compone de nodos y arcos, similares a las pegatinas comentadas anteriormente. Los nodos representan a las tareas y la información necesaria para calcular sus fecha de realización. Los arcos indican las precedencias entre tareas.

Vamos a representar cada nodo (tarea) como se ve en la figura.

<table>
<tr><td colspan="2">Etiqueta actividad</td><td colspan="2">Duración</td></tr>
<tr><td>Inicio temprano</td><td colspan="2" rowspan="2">DESCRIPCIÓN DE LA ACTIVIDAD</td><td>Final temprano</td></tr>
<tr><td>Inicio tardío</td><td>Final tardío</td></tr>
<tr><td colspan="2">Máximo tiempo disponible</td><td colspan="2">Holgura</td></tr>
</table>

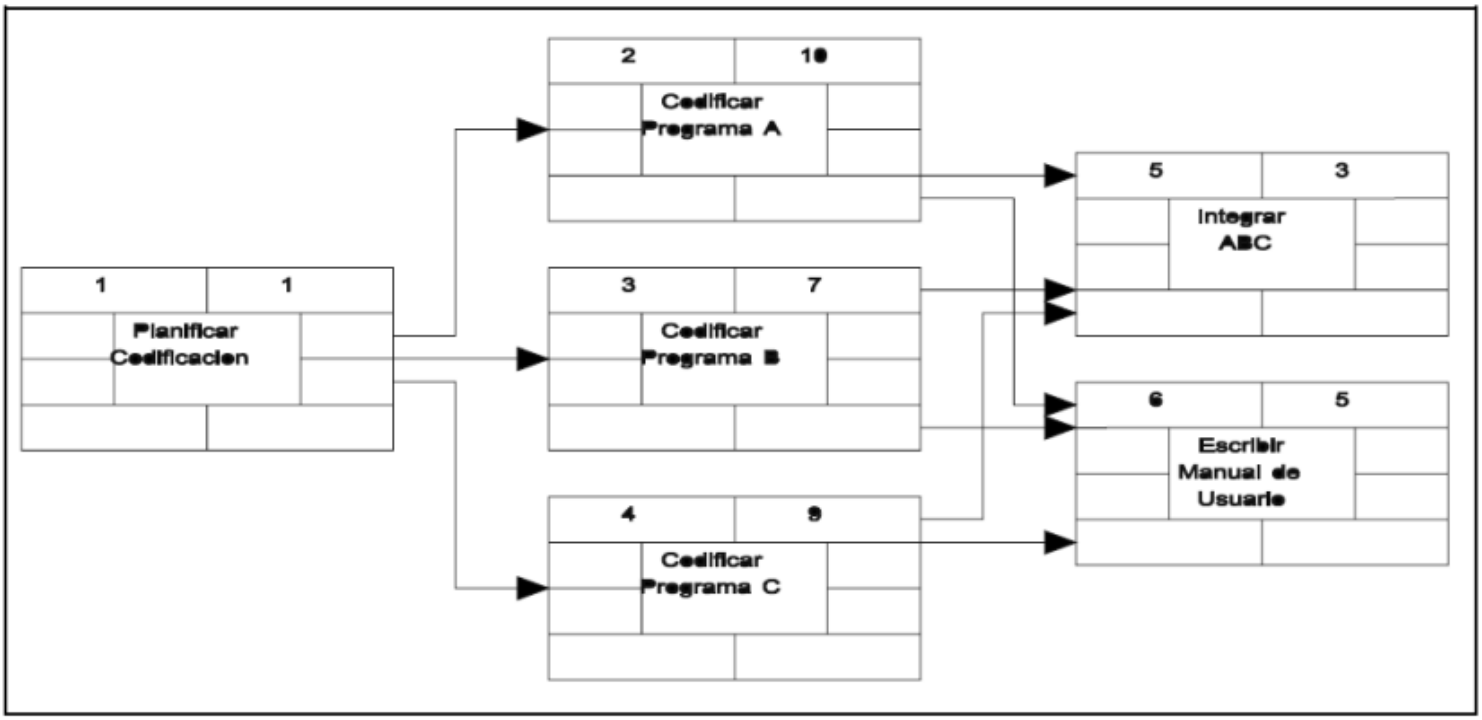
Donde:

- DESCRIPCIÓN DE LA ACTIVIDAD es el nombre que le hemos dado a la actividad. Por ejemplo: Codificación Programa A.

- Etiqueta actividad es un número arbitrario y que identifica unívocamente a cada actividad.
- Duración es el tiempo que calculamos que se tardará en completar la tarea, teniendo en cuenta el esfuerzo y los recursos asignados a la tarea. Por ejemplo una tarea que estimamos requerirá seis días-hombre de esfuerzo, si se realiza entre tres personas podría tener una duración de dos días.
- Inicio temprano es la fecha en que se podrá comenzar la tarea si no se retrasa ninguna otra.
- Final temprano es, en el caso de iniciarse la tarea en el inicio temprano, lo antes que puede finalizar, respetando su duración.
- Inicio tardío es la fecha más retrasada en la que puede comenzar la tarea para que se pueda completar el proyecto en la fecha marcada como final del proyecto.
- Final tardío es la fecha más retrasada en la que puede terminar la tarea para que se pueda completar el proyecto en la fecha marcada como final del proyecto.
- Máximo tiempo disponible es el tiempo máximo que puede durar una tarea en caso de comenzar en su Inicio temprano y concluir en su Final tardío.
- Holgura es la diferencia entre el Máximo tiempo disponible y su Duración.

Etapa 4. Cálculo del Camino Crítico

Una vez tenemos todas las tareas con sus respectivas duraciones y las precedencias pasamos a dibujar una red en la que aparezca para cada tarea una caja similar a la vista en el punto anterior con casi todos los campos vacíos. Entre ellas aparecerán los arcos indicando precedencias, tendremos algo similar a la figura.



Ahora calculamos las fechas tempranas. Para esto seguimos los siguientes pasos:

- En aquellas tareas que no tienen ningún predecesor se le asigna a *Inicio temprano* el valor 0.
- Si la tarea tiene predecesoras y todas estas tienen calculado su *Final temprano* se toma como *Inicio temprano* el máximo de todos ellos.
- El *Final temprano* de cada tarea se calcula como el *Inicio temprano* más la *Duración*.

Repetiremos estos pasos hasta que todas las tareas tengan sus fechas tempranas.

Para calcular las fechas tardías procederemos con los pasos que se describen a continuación:

- Se obtiene la fecha de finalización de proyecto más tardía. Esta puede venir dada por algún tipo de razones externas o puede que se nos pida que el proyecto termine lo antes posible, en este caso la fecha de finalización más tardía será el máximo de los "*Final temprano*" de todas las tareas.
- A aquellas tareas que no sean predecesoras de ninguna otra se les asigna como *Final tardío* la fecha de finalización más tardía del punto 4.
- El *Inicio tardío* se calcula restando al *Final tardío* la *Duración*.
- En aquellas tareas que son predecesoras de otras se calcula el *Final tardío* como el mínimo de los *Inicio tardío* de las tareas de que es predecesora.

Los otros dos campos de cada tarea: Máximo tiempo disponible y Holgura se calculan mediante las siguientes fórmulas:

```
Máximo tiempo disponible = Final tardío - Inicio temprano
Holgura = Máximo tiempo disponible - Duración
```

Obtención del Camino Crítico

Llamamos camino crítico de una planificación al conjunto de tareas que tienen *Holgura* cero. Siempre que se solicita que el proyecto tenga la duración mínima tendremos un camino crítico.

Se le llama camino crítico porque suele ser un camino que parte de una tarea que no tiene predecesoras y atraviesa el grafo por tareas con holgura cero hasta terminar en una tarea que no es predecesora de ninguna otra. Puede darse el caso de que con el "camino crítico" se puedan construir varias secuencias, partiendo de tareas sin predecesoras y se alcancen tareas sin sucesoras.

A las tareas del camino crítico se les llama tareas críticas y esto se debe a que un retraso en cualquiera de ellas lleva a un retraso del final del proyecto.

Diferencia Fundamental entre el CPM y el PERT

Aunque en principio son similares los algoritmos de ambos métodos, la asignación de duraciones de las tareas en el PERT es algo más elaborada. En lugar de realizarse una sola estimación se realizan tres estimaciones:

- "tm" tiempo medio que se estima para la actividad.
- "to" tiempo optimista, el que resultaría de ir todo muy bien.
- "tp" tiempo pesimista, el que resultaría si todo fuese mal en esta tarea.

A la tares se le asigna como duración el resultado de:

```
Duración = (to + 4 tm + tp) / 6
```

Por otra parte el grafo se construye de forma dual a la vista. Los arcos modelan las actividades o tareas, mientras que los nodos modelan la relación de precedencia de las tareas. Así un nodo indica que los arcos que llegan a él anteceden a los que salen de él.

Uso de Aplicaciones para la Planificación Control de Proyectos

Como hemos indiciado estos algoritmos se hicieron pensando en el uso de sistemas de cómputo automático, así que no es de extrañar que existan muchas aplicaciones que den soporte a éstos. Entre las más conocidas que funcionan sobre PC están el CA-SuperProject y el Microsoft Project.

Metodologías de Desarrollo: La Metodología Métrica 3

Introducción a la Metodología Métrica 3

Antes de describir una metodología, es necesario aclarar ciertas cuestiones, tales como:

- ¿Qué es una metodología?
- ¿Para qué sirve?

Por que la respuesta a estas preguntas va a ser crucial para poder entender y evaluar toda la exposición siguiente sobre la metodología Métrica 3.

¿Qué es una Metodología?

Según el Diccionario de la Lengua Española de la Real Academia, metodología es “Ciencia del método” o “Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal”. Aunque existe otra un poco más certera; “Modo de decir o hacer con orden una cosa”, parece que esto aclara algo más, lo que veremos en el tema es un modo o manera concreto (Métrica 3) de hacer con orden una cosa (desarrollo de sistemas de información).

El desarrollo de sistemas de información es una tarea de resolución de problemas, en la que el problema a resolver consiste en traducir:

- La situación-problema a un sistema de representación útil para nuestros propósitos (análisis de requisitos).
- La representación resultante de la fase anterior en otra que vaya acercando los objetos, conceptos y relaciones a una forma inteligible por nuestro dominio objetivo (máquinas, personas, otros sistemas de información); de esta manera realizaremos el diseño, desde el cual, en un proceso de traducción final, codificaremos los programas de ordenador, las instrucciones para los usuarios, las interfaces con otros sistemas, etc.

¿Para qué sirve?

- En primer lugar como guía; nos dice lo que tenemos que hacer, cómo, cuándo y quién tiene que hacerlo; además, lo hace de manera completa; podremos saltarnos los pasos que consideremos convenientes siempre que comprendamos la estructura del método y podamos evaluar la conveniencia de utilizar atajos.
- En segundo lugar, determina los puntos del proceso en los que debemos detenernos y comprobar cómo vamos, algo bastante importante y que evita la propagación de los errores a través del proceso.
- En tercer lugar, permite que los conocimientos adquiridos en el desarrollo de sistemas de información se plasmen en el método que la organización utiliza para ello mediante su continua revisión y adaptación y pasen a ser patrimonio de la organización y no solo de las personas que llevan a cabo la tarea.

Objetivos

La metodología MÉTRICA Versión 3 ofrece a las Organizaciones un instrumento útil para la sistematización de las actividades que dan soporte al ciclo de vida del software dentro del marco que permite alcanzar los siguientes objetivos:

- Proporcionar o definir Sistemas de Información que ayuden a conseguir los fines de la Organización mediante la definición de un marco estratégico para el desarrollo de los mismos.

- Dotar a la Organización de productos software que satisfagan las necesidades de los usuarios dando una mayor importancia al análisis de requisitos.
- Mejorar la productividad de los departamentos de Sistemas y Tecnologías de la Información y las Comunicaciones, permitiendo una mayor capacidad de adaptación a los cambios y teniendo en cuenta la reutilización en la medida de lo posible.
- Facilitar la comunicación y entendimiento entre los distintos participantes en la producción de software a lo largo del ciclo de vida del proyecto, teniendo en cuenta su papel y responsabilidad, así como las necesidades de todos y cada uno de ellos.
- Facilitar la operación, mantenimiento y uso de los productos software obtenidos.

Características

- La nueva versión de MÉTRICA contempla el desarrollo de Sistemas de Información para las distintas tecnologías que actualmente están conviviendo y los aspectos de gestión que aseguran que un Proyecto cumple sus objetivos en términos de calidad, coste y plazos.
- Su punto de partida es la versión anterior de MÉTRICA de la cual se han conservado la adaptabilidad, flexibilidad y sencillez, así como la estructura de actividades y tareas, si bien las fases y módulos de MÉTRICA versión 2.1 han dado paso a la división en Procesos, más adecuada a la entrada-transformación-salida que se produce en cada una de las divisiones del ciclo de vida de un proyecto. Para cada tarea se detallan los participantes que intervienen, los productos de entrada y de salida así como las técnicas y prácticas a emplear para su obtención.
- En la elaboración de MÉTRICA Versión 3 se han tenido en cuenta los métodos de desarrollo más extendidos, así como los últimos estándares de ingeniería del software y calidad, además de referencias específicas en cuanto a seguridad y gestión de proyectos. También se ha tenido en cuenta la experiencia de los usuarios de las versiones anteriores para solventar los problemas o deficiencias detectados.

Estructura

En una única estructura la metodología MÉTRICA Versión 3 cubre distintos tipos de desarrollo: estructurado y orientado a objetos, facilitando a través de interfaces la realización de los procesos de apoyo u organizativos: Gestión de Proyectos, Gestión de Configuración, Aseguramiento de Calidad y Seguridad.

La automatización de las actividades propuestas en la estructura de MÉTRICA Versión 3 es posible ya que sus técnicas están soportadas por una amplia variedad de herramientas de ayuda al desarrollo.

Cada Proceso detalla las Actividades y Tareas a realizar.

Para cada tarea se indican:

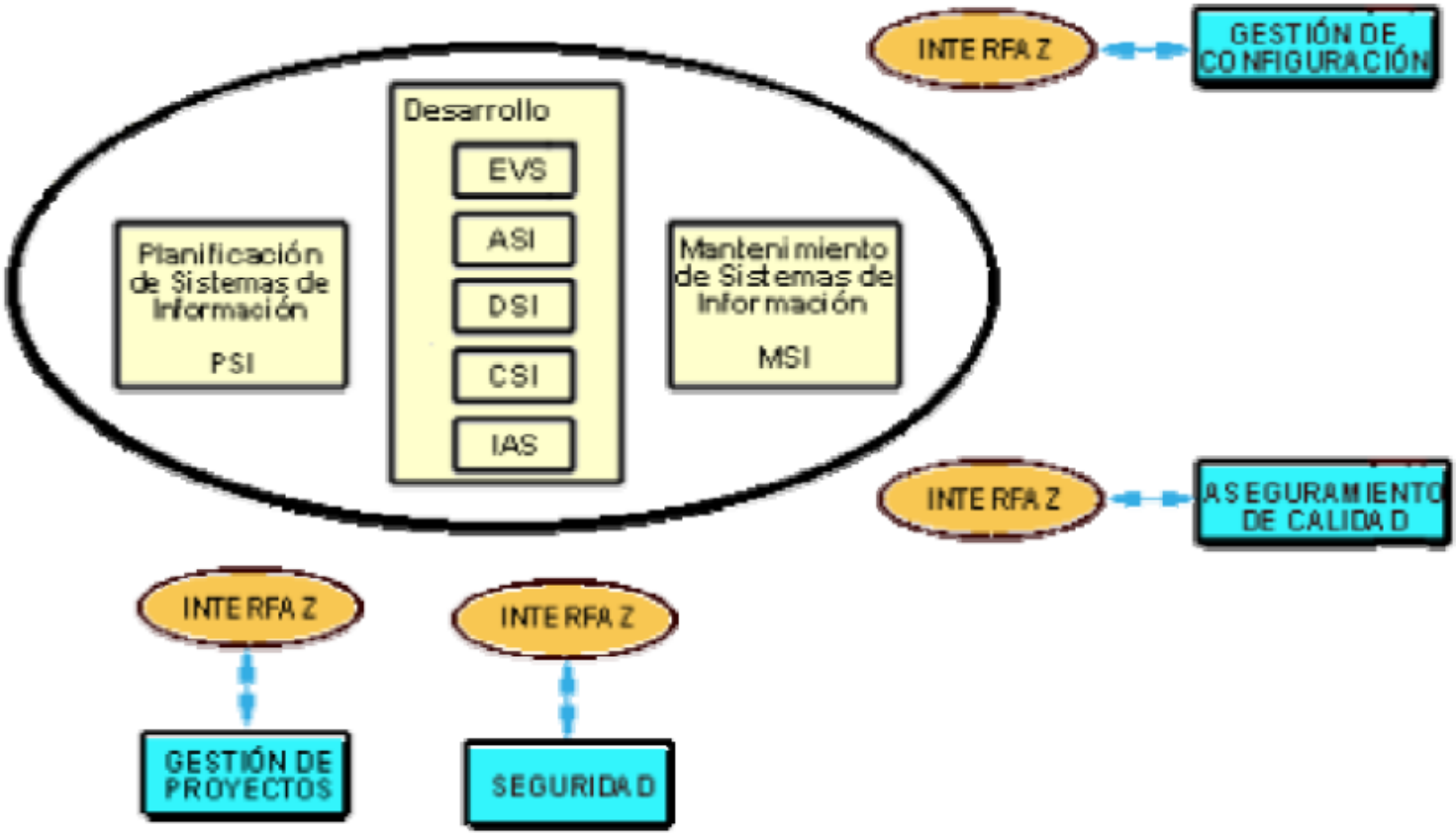
- Las técnicas y prácticas a utilizar.
- Los responsables de realizarla.
- Sus productos de entrada y salida.

La estructura de procesos es la siguiente:

- Planificación (PSI)
- Desarrollo
  - Estudio de Viabilidad (EVS)
  - Análisis (ASI)
  - Diseño (DSI)
  - Construcción (CSI)
  - Implantación y Aceptación (IAS)
- Mantenimiento (MSI)

Las interfaces son las siguientes:

- Gestión de la Configuración.
- Aseguramiento de la Calidad.
- Gestión de Proyectos.
- Seguridad.



Planificación de Sistemas de Información (PSI)

Descripción y Objetivos

El Plan de Sistemas de Información tiene como objetivo la obtención de un marco de referencia para el desarrollo de sistemas de información que responda a los objetivos estratégicos de la organización. Este marco de referencia consta de:

- Una descripción de la situación actual, que constituirá el punto de partida del Plan de Sistemas de Información. Dicha descripción incluirá un análisis técnico de puntos fuertes y riesgos, así como el análisis de servicio a los objetivos de la organización.
- Un conjunto de modelos que constituya la arquitectura de información.
- Una propuesta de proyectos a desarrollar en los próximos años, así como la prioridad de realización de cada proyecto.
- Una propuesta de calendario para la ejecución de dichos proyectos.
- La evaluación de los recursos necesarios para los proyectos a desarrollar en el próximo año, con el objetivo de tenerlos en cuenta en los presupuestos. Para el resto de proyectos, bastará con una estimación de alto nivel.
- Un plan de seguimiento y cumplimiento de todo lo propuesto mediante unos mecanismos de evaluación adecuados.

La perspectiva del plan debe ser estratégica y operativa, no tecnológica.

Es fundamental que la alta dirección de la organización tome parte activa en la decisión del Plan de Sistemas de Información con el fin de posibilitar su éxito. La dirección debe convencer a sus colaboradores más directos de la necesidad de realización del plan; de su

apoyo de forma constructiva, mentalizándose de que la ejecución del mismo requerirá la utilización de unos recursos de los cuales son responsables.

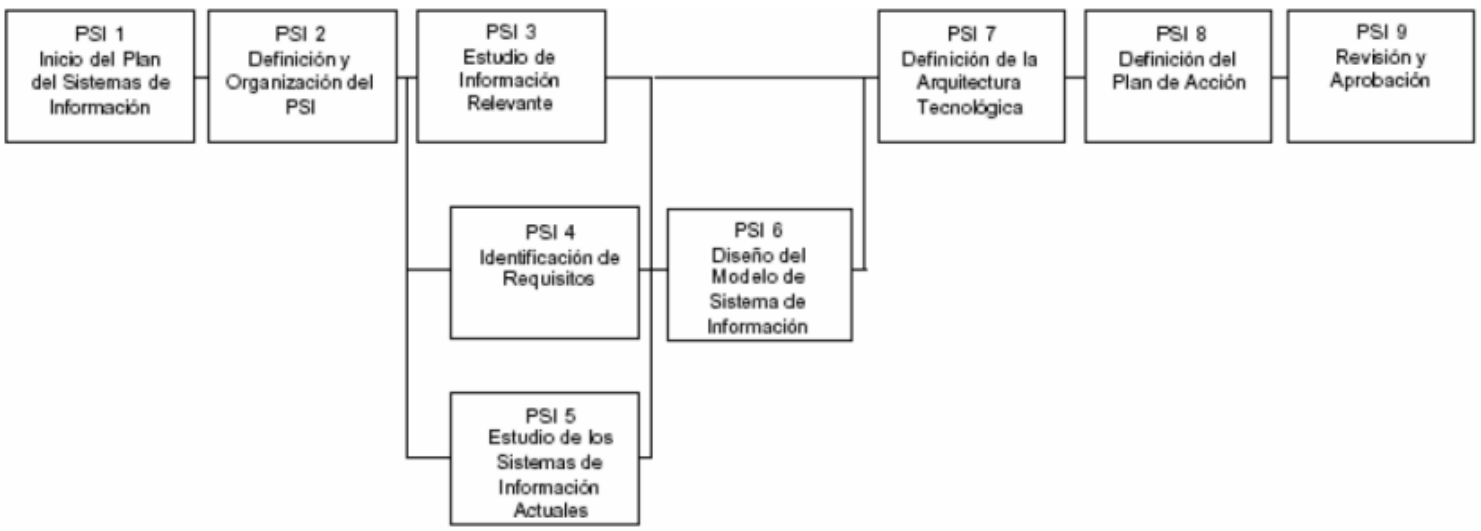
La presentación del Plan de Sistemas de Información y la constitución del equipo supone el arranque del proyecto y es fundamental que las más altas instancias de la organización estén implicadas en ambos, dando el apoyo necesario y aportando todo tipo de medios. Explicar el plan a las personas de la organización y a las unidades organizativas afectadas sobre las que recaerá el Plan, el apoyo de los altos directivos y la calificación de los recursos de las distintas unidades implicadas, serán factores críticos de éxito del Plan de Sistemas de Información.

El nivel de detalle con el que se hará el estudio de la situación actual dependerá de la existencia de documentación actual, de si hay personas que conozcan dicha documentación y de la predisposición a una sustitución total o parcial por sistemas de información nuevos. En cualquier caso, como paso previo para detectar aspectos importantes que puedan afectar a la organización, es necesario investigar sus puntos fuertes, áreas de mejora, riesgos y amenazas posibles y hacer un diagnóstico de los mismos.

Para la elaboración del Plan de Sistemas de Información se estudian las necesidades de información de los procesos de la organización afectados por el Plan, con el fin de definir los requisitos generales y obtener modelos conceptuales de información. Por otra parte se evalúan las opciones tecnológicas y se propone un entorno.

Tras analizar las prioridades relacionadas con las distintas variables que afectan a los sistemas de información, se elabora un calendario de proyectos con una planificación lo más detallada posible de los más inmediatos. Además, se propone una sistemática para mantener actualizado el Plan de Sistemas de Información para incluir en él todos los cambios necesarios, garantizando el cumplimiento adecuado del mismo.

A continuación se incluye un gráfico que representa la secuencia de actividades del proceso PSI.



Aunque los resultados de la actividad *Estudio de Información Relevante (PSI 3)* deberán tenerse en cuenta para la definición de requisitos que se efectúa en la actividad Identificación de Requisitos (PSI *4),* ambas podrán realizarse en paralelo, junto con el *Estudio de los Sistemas de Información Actuales (PSI 5)*.

Actividad PSI 1: Inicio del Plan de Sistemas de Información

El objetivo de esta actividad es determinar la necesidad del Plan de Sistemas de Información y llevar a cabo el arranque formal del mismo, con el apoyo del nivel más alto de la organización. Como resultado, se obtiene una descripción general del Plan de

Sistemas de Información que proporciona una definición inicial del mismo, identificando los objetivos estratégicos en los que apoya, así como el ámbito general de la organización al que afecta, lo que permite implicar a las direcciones de las áreas afectadas por el Plan de Sistemas de Información. Además, se identifican los factores críticos de éxito y los participantes en el Plan de Sistemas de Información, nombrando a los máximos responsables.

A continuación se incluye una table resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 1.1	Análisis de la necesidad del PSI	- Descripción general del PSI: o Aprobación de inicio del PSI	- Sesiones de trabajo	- Comité de Dirección
PSI 1.2	Identificación del alcance del PSI	- Descripción general del PSI: o Ámbito y objetivos del PSI o Objetivos estratégicos relacionados con el PSI o Factores críticos de éxito	- Factores críticos de éxito - Sesiones de trabajo	- Comité de Dirección
PSI 1.3	Determinación de responsables	- Descripción general del PSI: o Responsables del PSI	- Sesiones de trabajo	- Comité de Dirección

Actividad PSI 2: Definición y Organización del PSI

En esta actividad se detalla el alcance del plan, se organiza el equipo de personas que lo va a llevar a cabo y se elabora un calendario de ejecución. Todos los resultados o productos de esta actividad constituirán un marco de actuación del proyecto más detallado que en PSI 1 en cuanto a objetivos, procesos afectados, participantes, resultados y fechas de entrega.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 2.1	Especificación del Ámbito y Alcance	- Descripción general de procesos de la organización afectados - Catálogo de objetivos de PSI: o Objetivos generales o Objetivos específicos de cada proceso (si los hubiera)	- Catalogación	- Comité de Dirección - Directores de Usuarios - Jefe de Proyecto del PSI
PSI 2.2	Organización del PSI	- Catálogo de usuarios - Equipos de trabajo	- Catalogación - Sesiones de trabajo	- Directores de Usuarios - Jefe de Proyecto del PSI
PSI 2.3	Definición del Plan de Trabajo	- Plan de trabajo	- Planificación - Estimación	- Directores de Usuarios - Jefe de Proyecto del PSI
PSI 2.4	Comunicación del Plan de Trabajo	- Plan de trabajo: o Aceptación del Plan de Trabajo por parte de los implicados		- Comité de Dirección - Directores de Usuarios - Jefe de Proyecto del PSI

Actividad PSI 3: Estudio de la Información Relevante

El objetivo de esta actividad es recopilar y analizar todos los antecedentes generales que puedan afectar a los procesos y a las unidades organizativas implicadas en el Plan de Sistemas de Información, así como a los resultados del mismo. Pueden ser de especial interés los estudios realizados con anterioridad al Plan de Sistemas de Información, relativos a los sistemas de información de su ámbito, o bien a su entorno tecnológico, cuyas conclusiones deben ser conocidas por el equipo de trabajo del Plan de Sistemas de Información.

La información obtenida en esta actividad se tendrá en cuenta en la elaboración de los requisitos.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 3.1	Selección y Análisis de Antecedentes	- Valoración de antecedentes	- Sesiones de trabajo	- Consultores - Consultores Informáticos - Usuarios expertos
PSI 3.2	Valoración de Antecedentes	- Catálogo de requisitos - Requisitos generales - Catálogo de normas del PSI	- Catalogación	- Consultores - Consultores Informáticos

Actividad PSI 4: Identificación de Requisitos

El objetivo final de esta actividad va a ser la especificación de los requisitos de información de la organización, así como obtener un modelo de información que los complemente. Para conseguir este objetivo, se estudia el proceso o procesos de la organización incluidos en el ámbito del Plan de Sistemas de Información. Para ello es necesario llevar a cabo sesiones de trabajo con los usuarios, analizando cada proceso tal y como debería ser, y no según su situación actual, ya que ésta puede estar condicionada por los sistemas de información existentes.

Del mismo modo, se identifican los requisitos de información, y se elabora un modelo de información que represente las distintas entidades implicadas en el proceso, así como las relaciones entre ellas. Por último, se clasifican los requisitos identificados según su prioridad, con el objetivo de incorporarlos al catálogo de requisitos del Plan de Sistemas de Información.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 4.1	Estudio de los Procesos del PSI	- Modelo de procesos de la organización	- Modelado de Procesos de la Organización - Sesiones de trabajo	- Consultores - Usuarios expertos
PSI 4.2	Análisis de las Necesidades de la Información	- Necesidades de información - Modelo de información	- Modelo Entidad/Relación extendido - Diagrama de clases - Sesiones de trabajo	- Consultores - Usuarios expertos
PSI 4.3	Catalogación de Requisitos	- Catálogo de requisitos: - Requisitos de los procesos afectados por el PSI	- Catalogación	- Consultores - Usuarios expertos

Actividad PSI 5: Estudio de los Sistemas de Información Actuales

El objetivo de esta actividad es obtener una valoración de la situación actual al margen de los requisitos del catálogo, apoyándose en criterios relativos a facilidad de mantenimiento, documentación, flexibilidad, facilidad de uso, etc. En esta actividad se debe tener en cuenta la opinión de los usuarios, ya que aportarán elementos de valoración, como por

ejemplo, su nivel de satisfacción con cada sistema de información. Se seleccionan los sistemas de información actuales que son objeto del análisis y se lleva a cabo el estudio de los mismos con la profundidad y el detalle que se determine conveniente en función de los objetivos definidos para el Plan de Sistemas de Información. Este estudio permite, para cada sistema, determinar sus carencias y valorarlos.

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 5.1	Alcance y Objetivos del Estudio de los Sistemas de Información Actuales	- Catálogo de objetivos de PSI: - Objetivos del estudio de los Sistemas de Información actuales - Identificación de Sistemas de información actuales	- Catalogación - Sesiones de trabajo	- Consultores - Consultores Informáticos - Usuarios expertos
PSI 5.2	Análisis de los Sistemas de Información Actuales	- Descripción general de sistemas de información actuales	- Sesiones de trabajo - Diagrama de representación	- Consultores - Consultores Informáticos - Usuarios expertos - Equipo del Proyecto - Equipo de Soporte Técnico - Responsable de Mantenimiento
PSI 5.3	Valoración de los Sistemas de Información Actuales	- Valoración de la situación actual	- Sesiones de trabajo	- Consultores - Consultores Informáticos

Actividad PSI 6: Diseño del Modelo de Sistemas de Información

El objetivo de esta actividad es identificar y definir los sistemas de información que van a dar soporte a los procesos de la organización afectados por el Plan de Sistemas de Información. Para ello, en primer lugar, se analiza la cobertura que los sistemas de información actuales dan a los requisitos recogidos en el catálogo elaborado en las actividades Estudio de la Información Relevante (PSI 3) e Identificación de Requisitos (PSI 4). Esto permitirá efectuar un diagnóstico de la situación actual, a partir del cual se seleccionan los sistemas de información actuales considerados válidos, identificando las mejoras a realizar en los mismos. Por último, se definen los nuevos sistemas de información necesarios para cubrir los requisitos y funciones de los procesos no soportados por los sistemas actuales seleccionados. Teniendo en cuenta los resultados anteriores, se elabora el modelo de sistemas de información válido para dar soporte a los procesos de la organización incluidos en el ámbito del Plan de Sistemas de Información.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 6.1	Diagnóstico de la Situación Actual	- Diagnóstico de la situación actual: o Relación de sistemas de información que se conservan y mejoras necesarias	- Matricial	- Consultores
PSI 6.2	Definición del Modelo de Sistemas de Información	- Modelo de sistemas de información	- Matricial - Diagrama de representación - Sesiones de trabajo	- Consultores - Consultores Informáticos - Usuarios expertos

Actividad PSI 7: Definición de la Arquitectura Tecnológica

En esta actividad se propone una arquitectura tecnológica que dé soporte al modelo de información y de sistemas de información incluyendo, si es necesario, opciones. Para esta

actividad se tienen en cuenta especialmente los requisitos de carácter tecnológico, aunque es necesario considerar el catálogo completo de requisitos para entender las necesidades de los procesos y proponer los entornos tecnológicos que mejor se adapten a las mismas.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 7.1	Identificación de las necesidades de Infraestructura Tecnológica	- Alternativas de arquitectura tecnológica	- Diagrama de representación - Sesiones de trabajo	- Consultores Informáticos - Equipo de Soporte Técnico
PSI 7.2	Selección de la Arquitectura Tecnológica	- Arquitectura tecnológica	- Análisis Coste / Beneficio - Diagrama de representación - Impacto en la organización	- Consultores - Consultores Informáticos - Usuarios expertos - Equipo de Soporte Técnico

Actividad PSI 8: Definición del Plan de Acción

En el Plan de Acción, que se elabora en esta actividad, se definen los proyectos y acciones a llevar a cabo para la implantación de los modelos de información y de sistemas de información, determinados en las actividades Identificación de Requisitos (PSI 4) y Diseño del Modelo de Sistemas de Información (PSI 6), con la arquitectura tecnológica propuesta en la actividad Definición de la Arquitectura Tecnológica (PSI 7), el conjunto de estos tres modelos constituye la arquitectura de información. Dentro del Plan de Acción se incluye un calendario de proyectos, con posibles alternativas, y una estimación de recursos, cuyo detalle será mayor para los más inmediatos. Para la elaboración del calendario se tienen que analizar las distintas variables que afecten a la prioridad de cada proyecto y sistema de información. El orden definitivo de los proyectos y acciones debe pactarse con los usuarios, para llegar a una solución de compromiso que resulte la mejor posible para la organización. Por último, se propone un plan de mantenimiento para el control y seguimiento de la ejecución de los proyectos, así como para la actualización de los productos finales del Plan de Sistemas de Información.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 8.1	Definición de Proyectos a Realizar	- Plan de proyectos: o Definición de proyectos o Prioridad de proyectos o Calendario de proyectos y acciones	- Análisis Coste/ Beneficio - Impacto en la Organización - Sesiones de trabajo	- Consultores - Consultores Informáticos - Usuarios expertos
PSI 8.2	Elaboración del Plan de Mantenimiento del PSI	- Plan de mantenimiento del PSI		- Consultores - Consultores Informáticos

Actividad PSI 9: Revisión y Aprobación del PSI

Esta actividad tiene como objetivo contrastar con los responsables de la dirección del Plan de Sistemas de Información la arquitectura de información y el plan de acción elaborados anteriormente, para mejorar la propuesta si se considera necesario y por último, obtener su aprobación final.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
PSI 9.1	Convocatoria de la Presentación	- Plan de presentación	- Presentación	- Jefe de Proyecto del PSI - Consultores - Consultores Informáticos
PSI 9.2	Evaluación y Mejora de la Propuesta	- Resultado de las sesiones de trabajo - Presentación - Catálogo de requisitos del PSI - Arquitectura de información: o Modelo de información o Modelo de sistemas de información o Arquitectura tecnológica - Plan de acción: o Plan de proyectos o Plan de mantenimiento del PSI	- Sesiones de trabajo - Presentación	- Comité de Dirección - Jefe de Proyecto del PSI - Consultores - Consultores Informáticos - Usuarios expertos
PSI 9.3	Aprobación del PSI	- Aprobación formal del PSI - Plan de comunicación del PSI	- Sesiones de trabajo	- Comité de Dirección - Jefe de Proyecto del PSI

Estudio de Viabilidad del Sistema (EVS)

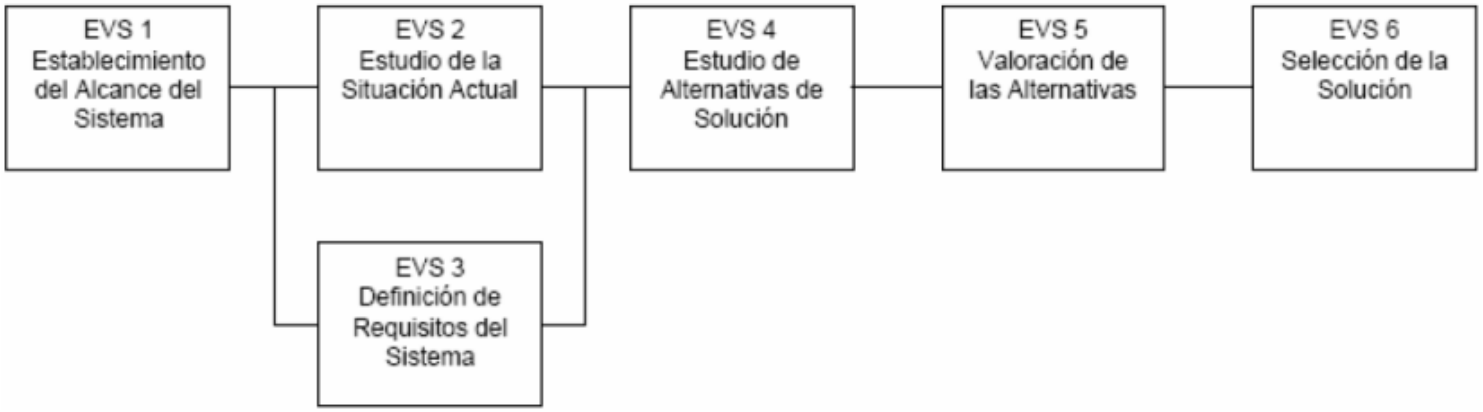
Descripción y Objetivos

Mientras que el Plan de Sistemas de Información tiene como objetivo proporcionar un marco estratégico que sirva de referencia para los Sistemas de Información de un ámbito concreto de una organización, el objetivo del Estudio de Viabilidad del Sistema es el análisis de un conjunto concreto de necesidades para proponer una solución a corto plazo, que tenga en cuenta restricciones económica, técnicas, legales y operativas. Las solución obtenida como resultado del estudio puede ser la definición de uno o varios proyectos que afecten a uno o varios sistemas de información y a existentes o nuevos. Para ello, se identifican los requisitos que se ha de satisfacer y se estudia, si procede, la situación actual.

A partir del estado inicial, la situación actual y los requisitos planteados, se estudian las alternativas de solución. Dichas alternativas pueden incluir soluciones que impliquen desarrollos a medida, soluciones basadas en la adquisición de productos software del mercado o soluciones mixtas. Se describe cada una de las alternativas, indicando los requisitos que cubre.

Una vez descritas cada una de las alternativas planteadas, se valora su impacto en la organización, la inversión a realizar en cada caso y los riesgos asociados. Esta información se analiza con el fin de evaluar las distintas alternativas y seleccionar la más adecuada, definiendo y estableciendo su planificación. Si en la organización se ha realizado con anterioridad un Plan de Sistemas de Información que afecte al sistema objeto de este estudio, se dispondrá de un conjunto de productos que proporcionarán información a tener en cuenta en todo el proceso.

Las actividades que engloba este proceso se recogen en la siguiente figura, en la que se indican las actividades que pueden ejecutarse en paralelo y las que precisan para su realización resultados originados en actividades anteriores.



Actividad EVS 1: Establecimiento del Alcance del Sistema

En esta actividad se estudia el alcance de la necesidad planteada por el cliente o usuario, o como consecuencia de la realización de un PSI, realizando una descripción general de la misma. Se determinarán los objetivos, se inicia el estudio de los requisitos y se identifican las unidades organizativas afectadas estableciendo su estructura.

Se analizan las posibles restricciones, tanto generales como específica, que puedan condicionar el estudio y la planificación de las alternativas de solución que se propongan. Si la justificación económica es obvia, el riesgo técnico bajo, se esperan pocos problemas legales y no existe ninguna alternativa razonable, no es necesario profundizar en el estudio de viabilidad del sistema, analizando posibles alternativas y realizando una valoración y evaluación de las mismas, sino que éste se orientará a la especificación de requisitos, descripción del nuevo sistema y planificación. Se detalla la composición del equipo de trabajo necesario para este proceso y su planificación. Finalmente, con el fin de facilitar la implicación activa de los usuarios en la definición del sistema, se identifican sus perfiles, dejando claras sus tareas y responsabilidades.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
EVS 1.1	Estudio de la Solicitud	- Descripción General del Sistema - Catálogo Objetivos EVS - Catálogo de Requisitos	- Catalogación - Sesiones de trabajo	- Comité de Dirección - Jefe de Proyecto - Analistas
EVS 1.2	Identificación del Alcance del Sistema	- Descripción General del Sistema: o Contexto del Sistema o Estructura Organizativa - Catálogo de Requisitos - Catálogo de Usuarios	- Diagrama de Flujo de Datos - Diagrama de Descomposición Funcional - Catalogación - Sesiones de trabajo	- Comité de Dirección - Jefe de Proyecto - Analistas
EVS 1.3	Especificación del Alcance del EVS	- Catálogo de Objetivos del EVS - Catálogo de Usuarios - Plan de Trabajo	- Catalogación - Sesiones de trabajo	- Comité de Dirección - Jefe de Proyecto - Analistas

Actividad EVS 2: Estudio de la Situación Actual

La situación actual es el estado en el que se encuentran los sistemas de información existentes en el momento en el que se inicia su estudio. Teniendo en cuenta el objetivo del estudio de la situación actual, se realiza una valoración de la información existente acerca de los sistemas de información afectados. En función de dicha valoración, se especifica el nivel de detalle con que se debe llevar a cabo el estudio. Si es necesario, se constituye un equipo de trabajo específico para su realización y se identifican los usuarios participantes en el mismo.

Si se decide documentar la situación actual, normalmente es conveniente dividir el sistema actual en subsistemas. Si es posible se describirá cada uno de los subsistemas, valorando qué información puede ser relevante para la descripción. Como resultado de esta actividad se genera un diagnóstico, estimando la eficiencia de los sistemas de información existentes e identificando los posibles problemas y las mejoras.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
EVS 2.1	Valoración del Estudio de la Situación Actual	- Descripción de la Situación Actual: o Contexto del Sistema Actual o Descripción de los Sistemas de Información Actuales	- Diagrama de Flujo de Datos - Diagrama de Representación - Sesiones de Trabajo	- Jefe de Proyecto - Analistas - Directores de Usuarios
EVS 2.2	Identificación de los Usuarios Participantes en el Estudio de la Situación Actual	- Catálogo Usuarios	- Sesiones de Trabajo - Catalogación	- Jefe de Proyecto - Directores de Usuarios
EVS 2.3	Descripción de los Sistemas de Información Existentes	- Descripción de la Situación Actual: o Descripción Lógica del Sistema Actual o Modelo Físico del Sistema Actual (opcional) o Matriz Localización Módulos y Datos	- Modelo Entidad /Relación Extendido - Diagrama de Flujo de Datos - Diagrama de Clases - Diagrama de Interacción de Objetos - Matricial - Diagrama de Representación - Sesiones de Trabajo	- Analistas - Usuarios expertos - Equipo de Soporte Técnico
EVS 2.4	Realización del Diagnóstico de la Situación Actual	- Descripción de la Situación Actual: o Diagnóstico de la Situación Actual		- Analistas - Responsable de Mantenimiento

Actividad EVS 3: Definición de Requisitos del Sistema

Esta actividad incluye la determinación de los requisitos generales, mediante una serie de sesiones de trabajo con los usuarios participantes, que hay que planificar y realizar. Una vez finalizadas, se analiza la información obtenida definiendo los requisitos y sus prioridades, que se añaden al catálogo de requisitos que servirá para el estudio y valoración de las distintas alternativas de solución que se propongan.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
EVS 3.1	Identificación de las Directrices Técnicas y de Gestión	- Catálogo de Normas	- Catalogación	- Jefe de Proyecto - Analistas - Usuarios Expertos
EVS 3.2	Identificación de Requisitos	- Identificación de Requisitos	- Sesiones de Trabajo	- Jefe de Proyecto - Analistas - Usuarios Expertos
EVS 3.3	Catalogación de Requisitos	- Catálogo de Requisitos	- Catalogación	- Jefe de Proyecto - Analistas - Usuarios Expertos

Actividad EVS 4: Estudio de Alternativas de Solución

Este estudio se centra en proponer diversas alternativas que respondan satisfactoriamente a los requisitos planteados, considerando también los resultado obtenidos en el Estudio de la Situación Actual (EVS 2), en el caso de que se haya realizado. Teniendo en cuenta el ámbito y funcionalidad que debe cubrir el sistema, puede ser conveniente realizar, previamente a la definición de cada alternativa, una descomposición del sistema en subsistemas. En la descripción de las distintas alternativas de solución propuestas, se debe especificar si alguna de ellas está basada, total o parcialmente, en un producto existente en el mercado. Si la alternativa incluye un desarrollo a medida, se debe incorporar en la descripción de la misma un modelo abstracto de datos y un modelo de procesos, y en orientación a objetos, un modelo de negocio y un modelo de dominio.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
EVS 4.1	Preselección de Alternativas de Solución	- Descomposición Inicial del Sistema en Subsistemas (opcional) - Alternativas de Solución a Estudiar	- Diagrama de Representación	- Jefe de Proyecto - Analistas - Técnicos de sistemas
EVS 4.2	Descripción de las Alternativas de Solución	- Catálogo de Requisitos - Alternativas de solución a estudiar: o Catálogo de Requisitos (cobertura) o Modelo de Descomposición en Subsistemas o Matriz Procesos / Localización Geográfica o Matriz Datos / Localización Geográfica o Entorno Tecnológico y Comunicaciones o Estrategia de Implantación Global del Sistema Si la alternativa requiere desarrollo: o Modelo Abstracto de Datos / Modelo de Procesos (En caso de ***Estructurado***) o Modelo de Negocio / Modelo de Dominio (En caso de ***Orientación a Objetos***) Si la alternativa incluye producto software estándar: o Descripción del Producto o Previsión de Evolución del Producto o Costes Ocasionados por Producto o Estándares del Producto o Descripción de Adaptación (si es necesaria)	- Matricial - Modelo Entidad/ Relación extendido - Diagrama de Flujo de Datos - Casos de Uso - Diagrama de Clases - Catalogación - Diagrama de Representación	- Jefe de Proyecto - Analistas - Usuarios Expertos - Técnicos de sistemas - Responsables de Seguridad - Especialistas en Comunicaciones

Actividad EVS 5: Valoración de las Alternativas

Una vez descritas las alternativas se realiza una valoración de las mismas, considerando el impacto en la organización, tanto desde el punto de vista tecnológico y organizativo como de operación, y los posibles beneficios que se esperan contrastados con los costes asociados. Se realiza también un análisis de los riesgos, decidiendo cómo enfocar el plan

de acción para minimizar los mismos y cuantificando los recursos y plazos precisos para planificar cada alternativa.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
EVS 5.1	Estudio de la Inversión	- Valoración de Alternativas: o Impacto en la Organización de Alternativas o Coste / Beneficio de Alternativas	- Análisis Coste / Beneficio	- Jefe de Proyecto - Analistas -
EVS 5.2	Estudio de los Riesgos	- Valoración de Alternativas: o Valoración de Riesgos	- Impacto en la Organización	- Jefe de Proyecto - Analistas
EVS 5.3	Planificación de Alternativas	- Plan de Trabajo de Cada Alternativa: o Enfoque del Plan de Trabajo de Cada Alternativa o Planificación de Cada Alternativa	- Planificación	- Jefe de Proyecto - Analistas

Actividad EVS 6: Selección de la Solución

Antes de finalizar el Estudio de Viabilidad del Sistema, se convoca al Comité de Dirección para la presentación de las distintas alternativas de solución, resultantes de la actividad anterior. En dicha presentación, se debaten las ventajas de cada una de ellas, incorporando las modificaciones que se consideren oportunas, con el fin de seleccionar la más adecuada. Finalmente, se aprueba la solución o se determina su inviabilidad.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
EVS 6.1	Convocatoria de Presentación	- Plan de Presentación de Alternativas	- Presentación	- Jefe de Proyecto
EVS 6.2	Evaluación de Alternativas y Selección	- Plan de Presentación de Alternativas - Catálogo de Requisitos - Solución Propuesta: o Descripción de la Solución o Contexto del Sistema (con la definición de las interfaces) o Impacto en Organización de la Solución o Coste / Beneficio de la Solución o Valoración de Riesgos de la Solución o Enfoque del Plan de Trabajo de la Solución o Planificación de la Solución	- Presentación - Sesiones de Trabajo	- Comité de Dirección - Jefe de Proyecto - Analistas
EVS 6.3	Aprobación de la Solución	- Aprobación de la Solución		- Comité de Dirección - Jefe de Proyecto

Análisis del Sistema de Información (ASI)

Descripción y Objetivos

El objetivo de este proceso es la obtención de una especificación detallada del sistema de información que satisfaga las necesidades de información de los usuarios y sirva de base para el posterior diseño del sistema. Al ser MÉTRICA Versión 3 una metodología que cubre tanto desarrollos estructurados como orientados a objetos, las actividades de ambas aproximaciones están integradas en una estructura común.

En la primera actividad, Definición del Sistema (ASI 1), se lleva a cabo la descripción inicial del sistema de información, a partir de los productos generados en el proceso Estudio de Viabilidad del Sistema (EVS). Se delimita el alcance del sistema, se genera un catálogo de requisitos generales y se describe el sistema mediante unos modelos iniciales de alto nivel. También se identifican los usuarios que participan en el proceso de análisis, determinando sus perfiles, responsabilidades y dedicaciones necesarias. Así mismo se elabora el plan de trabajo a seguir.

La definición de requisitos del nuevo sistema se realiza principalmente en la actividad. Establecimiento de Requisitos (ASI 2). El objetivo de esta actividad es elaborar un catálogo de requisitos detallado, que permita describir con precisión el sistema de información, y que además sirva de base para comprobar que es completa la especificación de los modelos obtenidos en las actividades Identificación de Subsistemas de Análisis (ASI 3), Análisis de Casos de Uso (ASI 4), Análisis de Clases (ASI 5), Elaboración del Modelo de Datos (ASI 6), Elaboración del Modelo de Procesos (ASI 7) y Definición de Interfaces de Usuario (ASI 8).

Hay que hacer constar que estas actividades pueden provocar la actualización del catálogo, aunque no se refleja como producto de salida en las tareas de dichas actividades, ya que el objetivo de las mismas no es crear el catálogo sino definir modelos que soporten los requisitos.

Para la obtención de requisitos en la actividad Establecimiento de Requisitos (ASI 2) se toman como punto de partida el catálogo de requisitos y los modelos elaborados en la actividad Definición del Sistema (ASI 1), completándolos mediante sesiones de trabajo con los usuarios. Estas sesiones de trabajo tienen como objetivo reunir la información necesaria para obtener la especificación detallada del nuevo sistema. Las técnicas que ayudan a la recopilación de esta información pueden variar en función de las características del proyecto y los tipos de usuario a entrevistar. Entre ellas podemos citar la reuniones, entrevistas, *Join Application Design (JAD)*, etc. Durante estas sesiones de trabajo se propone utilizar la especificación de los casos de uso como ayuda y guía en el establecimiento de requisitos. Esta técnica facilita la comunicación con los usuarios y en el análisis orientado a objetos constituye la base de la especificación. A continuación se identifican las facilidades que ha de proporcionar el sistema, y las restricciones a que está sometido en cuanto a rendimiento, frecuencia de tratamiento, seguridad y control de accesos, etc. Toda esta información se incorpora al catálogo de requisitos.

En la actividad Identificación de Subsistemas de Análisis (ASI 3), se estructura el sistema de información en subsistemas de análisis, para facilitar la especificación de los distintos modelos y la traza de requisitos. En paralelo, se generan los distintos modelos que sirven de base para el diseño. En el caso de análisis estructurado, se procede a la elaboración y descripción detallada del modelo de datos y de procesos, y en el caso de un análisis orientado a objetos, se elaboran el modelo de clases y el de interacción de objetos, mediante el análisis de los casos de uso. Se especifican, asimismo, todas las interfaces entre el sistema y el usuario, tales como formatos de pantallas, diálogos, formatos de informes y formularios de entrada.

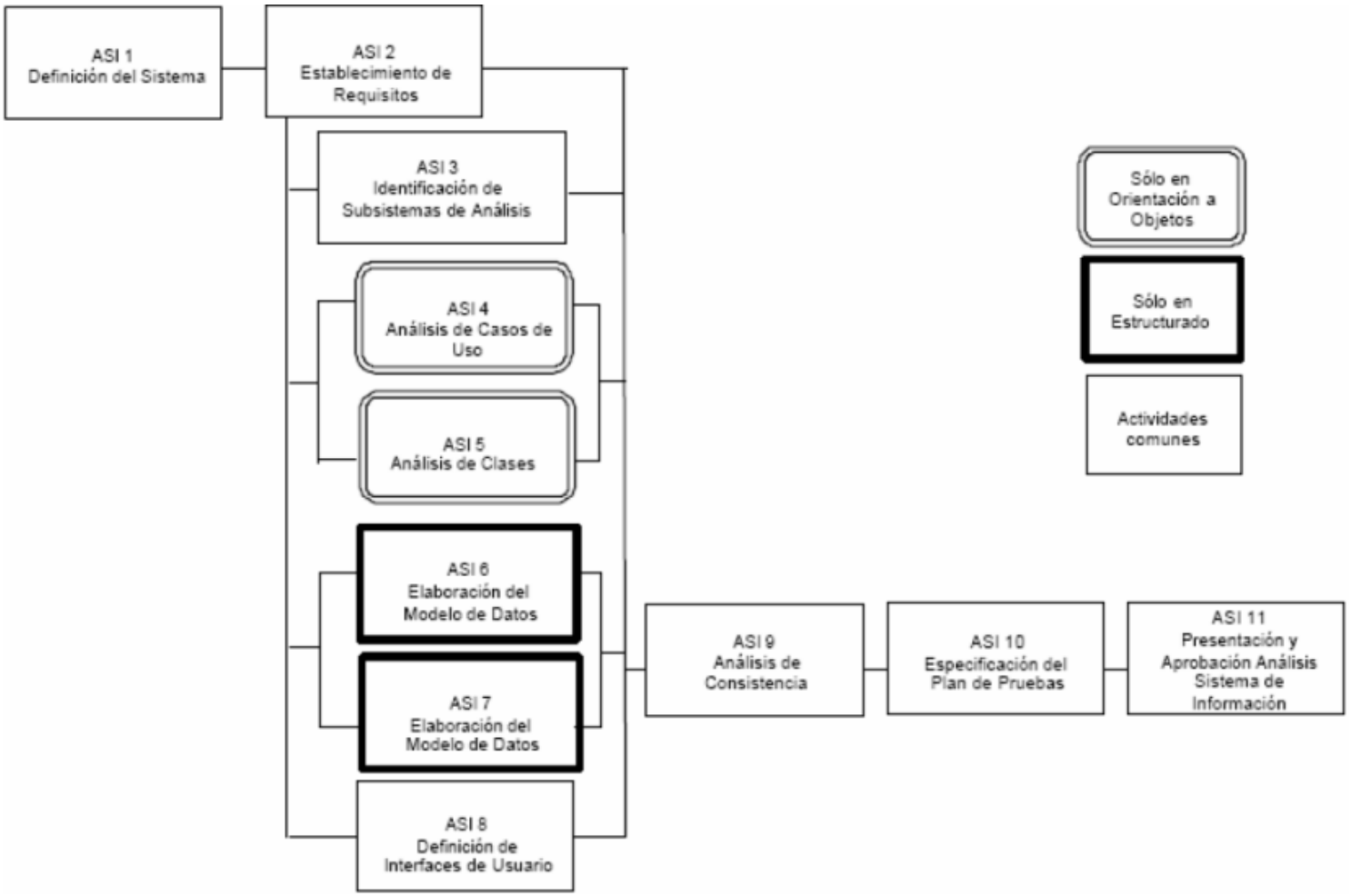
En la actividad Análisis de Consistencia y Especificación de Requisitos (ASI 9), se realiza la verificación y validación de los modelos, con el fin de asegurar que son:

- Completos, puesto que cada modelo obtenido contiene toda la información necesaria recogida en el catálogo de requisitos.

- Consistentes, ya que cada modelo es coherente con el resto de los modelos.
- Correctos, dado que cada modelo sigue unos criterios de calidad predeterminados en relación a la técnica utilizada, calidad de diagramas, elección de nombre, normas de calidad, etc.

En la actividad Especificación del Plan de Pruebas (ASI 10), se establece el marco general del plan de pruebas, iniciándose su especificación, que se completará en el proceso Diseño del Sistema de Información (DSI).

La participación activa de los usuarios es una condición imprescindible para el análisis del sistema de información, ya que dicha participación constituye una garantía de que los requisitos identificados son comprendidos e incorporados al sistema y, por tanto, de que éste será aceptado. Para facilitar la colaboración de los usuarios, se pueden utilizar técnicas interactivas, como diseño de diálogos y prototipos, que permiten al usuario familiarizarse con el nuevo sistema y colaborar en la construcción y perfeccionamiento del mismo. En el siguiente gráfico se muestra la relación de actividades del proceso Análisis del Sistema de Información, tanto para desarrollos estructurados como para desarrollos orientados a objetos, distinguiendo las que se pueden realizar en paralelo de aquellas que han de realizarse secuencialmente.



Actividad ASI 1: Definición del Sistema

Esta actividad tiene como objetivo efectuar una descripción del sistema, delimitando su alcance, estableciendo las interfaces con otros sistemas e identificando a los usuarios representativos. Las tareas de esta actividad se pueden haber desarrollado ya en parte en el proceso de Estudio de Viabilidad del Sistema (EVS), de modo que se parte de los productos obtenidos en dicho proceso para proceder a su adecuación como punto de partida para definir el sistema de información.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 1.1	Determinación del Alcance del Sistema	- Catálogo de Requisitos - Glosario ***Estructurado***: - Contexto del Sistema - Modelo Conceptual de Datos ***Orientación a Objetos***: - Modelo de Negocio - Modelo de Dominio	- Sesiones de Trabajo - Catalogación - Diagrama de Flujo de Datos - Modelo Entidad / Relación Extendido - Casos de Uso - Diagrama de Clases	- Jefe de Proyecto - Analistas - Directores de los Usuarios
ASI 1.2	Identificación del Entorno Tecnológico	- Catálogo de Requisitos - Descripción General del Entorno Tecnológico del Sistema	- Sesiones de Trabajo - Catalogación - Diagramas de Representación	- Jefe de Proyecto - Analistas - Directores de los Usuarios - Equipo de Soporte Técnico
ASI 1.3	Especificación de Estándares y Normas	- Catálogo de Normas	- Sesiones de Trabajo - Catalogación	- Jefe de Proyecto - Analistas - Directores de los Usuarios - Equipo de Soporte Técnico
ASI 1.4	Identificación de Usuarios Participantes y Finales	- Catálogo de Usuarios - Planificación	- Sesiones de Trabajo - Catalogación	- Jefe de Proyecto - Analistas - Directores de los Usuarios

Actividad ASI 2: Establecimiento de Requisitos

En esta actividad se lleva a cabo la definición, análisis y validación de los requisitos a partir de la información facilitada por el usuario, completándose el catálogo de requisitos obtenido en la actividad Definición del Sistema (ASI 1). El objetivo de esta actividad es obtener un catálogo detallado de los requisitos, a partir del cual se pueda comprobar que los productos generados en las actividades de modelización se ajustan a los requisitos de usuario.

Esta actividad se descompone en un conjunto de tareas que, si bien tienen un orden, exige continuas realimentaciones y solapamientos, entre sí y con otras tareas realizadas en paralelo. No es necesaria la finalización de una tarea para el comienzo de la siguiente. Lo que se tiene en un momento determinado es un catálogo de requisitos especificado en función de la progresión del proceso de análisis.

Se propone como técnica de obtención de requisitos la especificación de los casos de uso de la orientación a objetos, siendo opcional en el caso estructurado. Dicha técnica ofrece un diagrama simple y una guía de especificación en las sesiones de trabajo con el usuario.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 2.1	Obtención de Requisitos	- Catálogo de Requisitos - Modelo de Casos de Uso	- Sesiones de Trabajo - Catalogación - Casos de Uso	- Usuarios Expertos - Analistas
ASI 2.2	Especificación de Casos de Uso	- Catálogo de Requisitos - Modelo de Casos de Uso - Especificación de Casos de Uso	- Sesiones de Trabajo - Catalogación - Casos de Uso	- Usuarios Expertos - Analistas
ASI 2.3	Análisis de Requisitos	- Catálogo de Requisitos - Modelo de Casos de Uso - Especificación de Casos de Uso	- Sesiones de Trabajo - Catalogación - Casos de Uso	- Usuarios Expertos - Analistas
ASI 2.4	Validación de Requisitos	- Catálogo de Requisitos - Modelo de Casos de Uso - Especificación de Casos de Uso	- Sesiones de Trabajo - Catalogación - Casos de Uso	- Usuarios Expertos - Analistas

Actividad DSI 3: Identificación de Subsistemas de Análisis

El objetivo de esta actividad, común tanto para análisis estructurado como para análisis orientado a objetos, es facilitar el análisis del sistema de información llevando a cabo la descomposición del sistema en subsistemas. Se realiza en paralelo con el resto de las actividades de generación de modelos del análisis. Por tanto, se asume la necesidad de una realimentación y ajuste continuo con respecto a la definición de los subsistemas, sus dependencias y sus interfaces.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 3.1	Determinación de Subsistemas de Análisis	***Estructurado***: - Modelo de Procesos ***Orientación a Objetos***: - Descripción de Subsistemas de Análisis - Descripción de Interfaces entre Subsistemas	- Diagrama de Flujo de Datos - Diagrama de Paquetes (Subsistemas)	- Jefe de Proyecto - Analistas
ASI 3.2	Integración de Subsistemas de Análisis	- Desarrollo y Aceptación ***Estructurado***: - Modelo de Procesos ***Orientación a Objetos***: - Descripción de Subsistemas de Análisis - Descripción de Interfaces entre Subsistemas	- Diagrama de Flujo de Datos - Diagrama de Paquetes (Subsistemas)	- Jefe de Proyecto - Analistas

Actividad ASI 4: Análisis de los Casos de Uso

El objetivo de esta actividad, que sólo se realiza en el caso de **Análisis Orientado a Objetos**, es identificar las clases cuyos objetos son necesarios para realizar un caso de uso y describir su comportamiento mediante la interacción de dichos objetos. Esta actividad se lleva a cabo para cada uno de los casos de uso contenidos en un subsistema de los definidos en la actividad Identificación de Subsistemas de Análisis (ASI 3).

Las tareas de esta actividad no se realizan de forma secuencial sino en paralelo, con continuas realimentaciones entre ellas y con las realizadas en las actividades Establecimiento de Requisitos (ASI 2), Identificación de Subsistemas de Análisis (ASI 3), Análisis de Clases (ASI 5) y Definición de Interfaces de Usuario (ASI 8).

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 4.1	Identificación de Clases Asociadas a un Caso de Uso	- Modelo de Clases de Análisis	- Diagrama de Clases	- Analistas
ASI 4.2	Descripción de la Interacción de Objetos	- Análisis de la Realización de los Casos de Uso	- Diagrama de Interacción de Objetos (secuencia o colaboración)	- Analistas

Actividad ASI 5: Análisis de Clases

El objetivo de esta actividad que sólo se realiza en el caso de **Análisis Orientado a Objetos** es describir cada una de las clases que ha surgido, identificando las responsabilidades que tienen asociadas, sus atributos, y las relaciones entre ellas. Para esto, se debe tener en cuenta la normativa establecida en la tarea Especificación de Estándares y Normas (ASI 1.3), de forma que el modelo de clases cumpla estos criterios, con el fin de evitar posibles inconsistencias en el diseño. Teniendo en cuenta las clases identificadas en la actividad Análisis de los Casos de Uso (ASI 4), se elabora el modelo de clases para cada subsistema. A medida que avanza el análisis, dicho modelo se va completando con las clases que vayan apareciendo, tanto del estudio de los casos de uso, como de la interfaz de usuario necesaria para el sistema de información.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 5.1	Identificación de Responsabilidades y Atributos	- Modelo de Clases de Análisis - Comportamiento de Clases de Análisis	- Diagrama de Clases - Diagrama de Transición de Estados	- Analistas
ASI 5.2	Identificación de Asociaciones y Agregaciones	- Modelo de Clases de Análisis	- Diagrama de Clases	- Analistas
ASI 5.3	Identificación de Generalizaciones	- Modelo de Clases de Análisis	- Diagrama de Clases	- Analistas

Actividad ASI 6: Elaboración del Modelo de Datos

El objetivo de esta actividad que se lleva a cabo únicamente en el caso de **Análisis Estructurado** es identificar las necesidades de información de cada uno de los procesos que conforman el sistema de información, con el fin de obtener un modelo de datos que contemple todas las entidades, relaciones, atributos y reglas de negocio necesarias para dar respuesta a dichas necesidades.

El modelo de datos se elabora siguiendo un enfoque descendente (*top-down*). A partir del modelo conceptual de datos, obtenido en la tarea Determinación del Alcance del Sistema (ASI 1.1), se incorporan a dicho modelo todas las entidades que vayan apareciendo, como resultado de las funcionalidades que se deban cubrir y de las necesidades de información del usuario. Es necesario tener en cuenta el catálogo de requisitos y el modelo de procesos, productos que se están generando en paralelo en las actividades Establecimiento de Requisitos (ASI 2), Identificación de Subsistemas de Análisis (ASI 3) y Elaboración del Modelo de Procesos (ASI 7). Una vez construido el modelo conceptual y definidas sus entidades, se resuelven las relaciones complejas y se completa la información de entidades, relaciones, atributos y ocurrencias de las entidades, generando el modelo lógico de datos. Como última tarea en la definición del modelo, se asegura la normalización hasta la tercera forma normal para obtener el modelo lógico de datos normalizado. Finalmente, si procede, se describen las necesidades de migración y carga inicial de los datos.

Esta actividad se realiza en paralelo, y con continuas realimentaciones, con otras tareas realizadas en las actividades Establecimiento de Requisitos (ASI 2), Identificación de Subsistemas de Análisis (ASI 3), Elaboración del Modelo de Procesos (ASI 7) y Definición de Interfaces de Usuario (ASI 8).

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 6.1	Elaboración del Modelo Conceptual de Datos	- Modelo Conceptual de Datos	- Modelo Entidad / Relación Extendido	- Analistas
ASI 6.2	Elaboración del Modelo Lógico de Datos	- Modelo Lógico de Datos	- Modelo Entidad / Relación Extendido	- Analistas
ASI 6.3	Normalización del Modelo Lógico de Datos	- Modelo Lógico de Datos Normalizado	- Normalización	- Analistas
ASI 6.4	Especificación de Necesidades de Migración de Datos y Carga Inicial	- Plan de Migración y Carga Inicial de Datos	- Sesiones de Trabajo	- Usuarios Expertos - Analistas - Equipo de soporte Técnico

Actividad ASI 7: Elaboración del Modelo de Procesos

El objetivo de esta actividad, que se lleva a cabo únicamente en el caso de **Análisis Estructurado**, es analizar las necesidades del usuario para establecer el conjunto de procesos que conforma el sistema de información. Para ello, se realiza una descomposición de dichos procesos siguiendo un enfoque descendente (*top-down*), en varios niveles de abstracción, donde cada nivel proporciona una visión más detallada del proceso definido en el nivel anterior. Con el fin de facilitar el desarrollo posterior, se debe llegar a un nivel de descomposición en el que los procesos obtenidos sean claros y sencillos, es decir, buscar un punto de equilibrio en el que dichos procesos tengan significado por sí mismos dentro del sistema global y a su vez la máxima independencia y simplicidad.

Esta actividad se lleva a cabo para cada uno de los subsistemas identificados en la actividad Identificación de Subsistemas de Análisis (ASI 3). Las tareas de esta actividad se realizan en paralelo y con continuas realimentaciones con otras tareas ejecutadas en las actividades Establecimiento de Requisitos (ASI 2), Elaboración del Modelo de Datos (ASI 6) y Definición de Interfaces de Usuario (ASI 8).

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 7.1	Obtención del Modelo de Procesos del Sistema	- Modelo de Procesos - Matriz de Procesos / Localización Geográfica (ampliada)	- Diagrama de Flujo de Datos - Matricial	- Analistas
ASI 7.2	Especificación de Interfaces con otros Sistemas	- Descripción de Interfaz con otros Sistemas	-	- Analistas

Actividad ASI 8: Definición de Interfaces de usuario

En esta actividad se especifican las interfaces entre el sistema y el usuario: formatos de pantallas, diálogos, e informes, principalmente. El objetivo es realizar un análisis de los

procesos del sistema de información en los que se requiere una interacción del usuario, con el fin de crear una interfaz que satisfaga todos los requisitos establecidos, teniendo en cuenta los diferentes perfiles a quiénes va dirigido. Al comienzo de este análisis es necesario seleccionar el entorno en el que es operativa la interfaz, considerando estándares internacionales y de la instalación, y establecer las directrices aplicables en los procesos de diseño y construcción. El propósito es construir una interfaz de usuario acorde a sus necesidades, flexible, coherente, eficiente y sencilla de utilizar, teniendo en cuenta la facilidad de cambio a otras plataformas, si fuera necesario. Se identifican los distintos grupos de usuarios de acuerdo con las funciones que realizan, conocimientos y habilidades que poseen, y características del entorno en el que trabajan. La identificación de los diferentes perfiles permite conocer mejor las necesidades y particularidades de cada uno de ellos.

Asimismo, se determina la naturaleza de los procesos que se llevan a cabo (en lotes o en línea). Para cada proceso en línea se especifica qué tipo de información requiere el usuario para completar su ejecución realizando, para ello, una descomposición en diálogos que refleje la secuencia de la interfaz de pantalla tipo carácter o pantalla gráfica. Finalmente, se define el formato y contenido de cada una de las interfaces de pantalla especificando su comportamiento dinámico. Se propone un flujo de trabajo muy similar para desarrollos estructurados y orientados a objetos, coincidiendo en la mayoría de las tareas, si bien es cierto que en orientación a objetos, al identificar y describir cada escenario en la especificación de los casos de uso, se hace un avance muy significativo en la toma de datos para la posterior definición de la interfaz de usuario. Como resultado de esta actividad se genera la especificación de interfaz de usuario, como producto que engloba los siguientes elementos:

- Principios generales de la interfaz.
- Catálogo de perfiles de usuario.
- Descomposición funcional en diálogos.
- Catálogo de controles y elementos de diseño de interfaz de pantalla.
- Formatos individuales de interfaz de pantalla.
- Modelo de navegación de interfaz de pantalla.
- Formatos de impresión.
- Prototipo de interfaz interactiva.
- Prototipo de interfaz de impresión.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 8.1	Especificación de Principios Generales de la Interfaz	- Especificación de Interfaz de Usuario: ○ Principios Generales de la Interfaz	- Sesiones de Trabajo	- Usuarios Expertos - Analistas

ASI 8.2	Identificación de Perfiles y Diálogos (Solo para ***Análisis Estructurado***)	- Especificación de Interfaz de Usuario:
		  ○ Catálogo de Perfiles de Usuario
		  ○ Descomposición Funcional en Diálogos
			- Diagrama de Descomposición Funcional
			- Sesiones de Trabajo
			- Catalogación
			- Diagrama de Representación
				- Usuarios Expertos
				- Analistas

ASI 8.3	Especificación de Formatos Individuales de la Interfaz de Pantalla	- Especificación de Interfaz de Usuario:
		  ○ Formatos Individuales de Interfaz de Pantalla
		  ○ Catálogo de Controles y Elementos de Diseño de Interfaz de Pantalla
			- Prototipado
			- Catalogación
			- Sesiones de Trabajo
			- Casos de Uso
				- Usuarios Expertos
				- Analistas

ASI 8.4	Especificación del Comportamiento Dinámico de la Interfaz	- Especificación de Interfaz de Usuario:
		  ○ Modelo de Navegación de Interfaz de Pantalla
		  ○ Prototipo de Interfaz Interactiva
			- Diagrama de Transición de Estados
			- Prototipado
			- Sesiones de Trabajo
			- Matricial
			- Diagrama de Interacción de Objetos
				- Usuarios Expertos
				- Analistas

ASI 8.5	Especificación de Formatos de Impresión	- Especificación de Interfaz de Usuario:
		  ○ Formatos de Impresión
		  ○ Prototipo de Interfaz de Impresión
			- Prototipado
			- Sesiones de Trabajo
				- Usuarios Expertos
				- Analistas

Actividad ASI 9: Análisis de Consistencia y Especificación de Requisitos

El objetivo de esta actividad es garantizar la calidad de los distintos modelos generados en el proceso de Análisis del Sistema de Información, y asegurar que los usuarios y los Analistas tienen el mismo concepto del sistema.

Para cumplir dicho objetivo, se llevan a cabo las siguientes acciones:

- Verificación de la calidad técnica de cada modelo.
- Aseguramiento de la coherencia entre los distintos modelos.
- Validación del cumplimiento de los requisitos.

Esta actividad requiere una herramienta de apoyo para realizar el análisis de consistencia. También se elabora en esta actividad la Especificación de Requisitos Software (ERS), como producto para la aprobación formal, por parte del usuario, de las especificaciones del sistema. La Especificación de Requisitos Software se convierte en la línea base para los procesos posteriores del desarrollo del software, de modo que cualquier petición de cambio en los requisitos que pueda surgir posteriormente, debe ser evaluada y aprobada.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 9.1	Verificación de los Modelos	- Especificación de Interfaz de Usuario ***Estructurado***: - Modelo Lógico de Datos Normalizado - Modelo de Procesos ***Orientación a Objetos***: - Modelo de Casos de Uso - Especificación de Casos de Uso - Descripción de Subsistemas de Análisis - Descripción de Interfaces entre Subsistemas - Modelo Clases de Análisis - Comportamiento de Clases de Análisis - Análisis de la Realización de los Casos de Uso	-	- Analistas - Equipo de Arquitectura
ASI 9.2	Análisis de Consistencia entre Modelos	- Resultado de Análisis de Consistencia - Especificación de Interfaz de Usuario ***Estructurado***: - Modelo Lógico de Datos Normalizado - Modelo de Procesos ***Orientación a Objetos***: - Modelo de Casos de Uso - Especificación de Casos de Uso - Descripción de Subsistemas de Análisis - Descripción de Interfaces entre Subsistemas - Modelo de Clases de Análisis - Comportamiento de Clases de Análisis - Análisis de la Realización de los Casos de Uso	- Matricial - Cálculo de Accesos Lógicos - Caminos de Accesos Lógicos en Consultas	- Analistas - Equipo de Arquitectura
ASI 9.3	Validación de los Modelos	- Especificación de Interfaz de Usuario ***Estructurado***: - Modelo Lógico de Datos Normalizado - Modelo de Procesos ***Orientación a Objetos***: - Modelo de Casos de Uso - Especificación de Casos de Uso - Descripción de Subsistemas de Análisis - Descripción de Interfaces entre Subsistemas - Modelo de Clases de Análisis - Comportamiento de Clases de Análisis - Análisis de la Realización de los Casos de Uso	- Prototipado	- Analistas - Usuarios Expertos

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 9.4	Elaboración de la Especificación de Requisitos Software (ERS)	- Especificación de Requisitos Software (ERS)		- Analistas

Actividad ASI 10: Especificación del Plan de Pruebas

En esta actividad se inicia la definición del plan de pruebas, el cual sirve como guía para la realización de las pruebas, y permite verificar que el sistema de información cumple las necesidades establecidas por el usuario, con las debidas garantías de calidad. El plan de pruebas es un producto formal que define los objetivos de la prueba de un sistema, establece y coordina una estrategia de trabajo, y provee del marco adecuado para elaborar una planificación paso a paso de las actividades de prueba. El plan se inicia en el proceso Análisis del Sistema de Información (ASI), definiendo el marco general, y estableciendo los requisitos de prueba de aceptación, relacionados directamente con la especificación de requisitos.

Dicho plan se va completando y detallando a medida que se avanza en los restantes procesos del ciclo de vida del software, Diseño del Sistema de Información (DSI), Construcción del Sistema de Información (CSI) e Implantación y Aceptación del Sistema (IAS). Se plantean los siguientes niveles de prueba:

- Pruebas unitarias.
- Pruebas de integración.
- Pruebas del sistema.
- Pruebas de implantación.
- Pruebas de aceptación.

En esta actividad también se avanza en la definición de las pruebas de aceptación del sistema. Con la información disponible, es posible establecer los criterios de aceptación de las pruebas incluidas en dicho nivel, al poseer la información sobre los requisitos que debe cumplir el sistema, recogidos en el catálogo de requisitos.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 10.1	Definición del Alcance de las Pruebas	- Plan de Pruebas	- Sesiones de Trabajo	- Jefe de Proyecto - Analistas - Equipo de Soporte Técnico - Usuarios Expertos
ASI 10.2	Definición de Requisitos del Entorno de Pruebas	- Plan de Pruebas	- Sesiones de Trabajo	- Jefe de Proyecto - Analistas - Equipo de Soporte Técnico - Usuarios Expertos
ASI 10.3	Definición de las Pruebas de Aceptación del Sistema	- Plan de Pruebas	- Sesiones de Trabajo	- Jefe de Proyecto - Analistas - Equipo de Soporte Técnico - Usuarios Expertos

Actividad ASI 11: Aprobación del Análisis del Sistema de Información

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
ASI 11.1	Presentación y Aprobación del Análisis del Sistema de Información	- Aprobación del Análisis del Sistema de Información	- Presentación	- Comité de Dirección - Jefe de Proyecto

Diseño del Sistema de Información (DSI)

Descripción y Objetivos

El objetivo del proceso de Diseño del Sistema de Información (DSI) es la definición de la arquitectura del sistema y del entorno tecnológico que le va a dar soporte, junto con la especificación detallada de los componentes del sistema de información. A partir de dicha información, se generan todas las especificaciones de construcción relativas al propio sistema, así como la descripción técnica del plan de pruebas, la definición de los requisitos de implantación y el diseño de los procedimientos de migración y carga inicial, éstos últimos cuando proceda.

Al ser MÉTRICA Versión 3 una metodología que cubre tanto desarrollos estructurados como orientados a objetos, las actividades de ambas aproximaciones están integradas en una estructura común. Las actividades de este proceso se agrupan en dos grandes bloques:

- En un primer bloque de actividades, que se llevan a cabo en paralelo, se obtiene el diseño de detalle del sistema de información. La realización de estas actividades exige una continua realimentación. En general, el orden real de ejecución de las mismas depende de las particularidades del sistema de información y, por lo tanto, de generación de sus productos.
  En la actividad Definición de la Arquitectura del Sistema (DSI 1), se establece el particionamiento físico del sistema de información, así como su organización en subsistemas de diseño, la especificación del entorno tecnológico, y sus requisitos de operación, administración, seguridad y control de acceso. Se completan los catálogos de requisitos y normas, en función de la definición del entorno tecnológico, con aquellos aspectos relativos al diseño y construcción que sea necesario contemplar. Asimismo, se crea un catálogo de excepciones del sistema, en el que se registran las situaciones de funcionamiento secundario o anómalo que se estime oportuno considerar y, por lo tanto, diseñar y probar. Este catálogo de excepciones se utiliza como referencia en la especificación técnica de las pruebas del sistema.
  El particionamiento físico del sistema de información permite organizar un diseño que contemple un sistema de información distribuido, como por ejemplo la arquitectura cliente/servidor, siendo aplicable a arquitecturas multinivel en general. Independientemente de la infraestructura tecnológica, dicho particionamiento representa los distintos niveles funcionales o físicos del sistema de información. La relación entre los elementos del diseño y particionamiento físico, y a su vez, entre el particionamiento físico y el entorno tecnológico, permite una especificación de la distribución de los elementos del sistema de información y, al mismo tiempo, un diseño orientado a la movilidad a otras plataformas o la reubicación de subsistemas.
  El sistema de información se estructura en subsitemas de diseño. Éstos a su vez se clasifican como de soporte o específicos, al responder a propósitos diferentes.
  - Los subsistemas de soporte contienen los elementos o servicios comunes al sistema y a la instalación, y generalmente están originados por la interacción con la infraestructura técnica o la reutilización de otros sistemas, con un nivel de complejidad técnica mayor.
  - Los subsistemas específicos contienen los elementos propios del sistema de información, generalmente con una continuidad de los subsistemas definidos en el proceso de Análisis del Sistema de Información (ASI).

También se especifica en detalle el entorno tecnológico del sistema de información, junto con su planificación de capacidades (capacity planning), y sus requisitos de operación, administración, seguridad y control de acceso.
El diseño detallado del sistema de información, siguiendo un enfoque estructurado, comprende un conjunto de actividades que se llevan a cabo en paralelo a la Definición de la Arquitectura del Sistema (DSI 1). El alcance de cada una de estas actividades se resume a continuación:

* Diseño de la Arquitectura de Soporte (DSI 2), que incluye el diseño detallado de los subsistemas de soporte, el establecimiento de las normas y requisitos propios del diseño y construcción, así como la identificación y definición de los mecanismos genéricos de diseño y construcción.
* Diseño de la Arquitectura de Módulos del Sistema (DSI 5), donde se realiza el diseño de detalle de los subsistemas específicos del sistema de información y la revisión de la interfaz de usuario.
* Diseño Físico de Datos (DSI 6), que incluye el diseño y optimización de las estructuras de datos del sistema, así como su localización en los nodos de la arquitectura propuesta.

En el caso de Diseño Orientado a Objetos, conviene señalar que el diseño de la persistencia de los objetos se lleva a cabo sobre bases de datos relacionales, y que el diseño detallado del sistema de información se realiza en paralelo con la actividad de Diseño de la Arquitectura de Soporte (DSI 2), y se corresponde con las siguientes actividades:

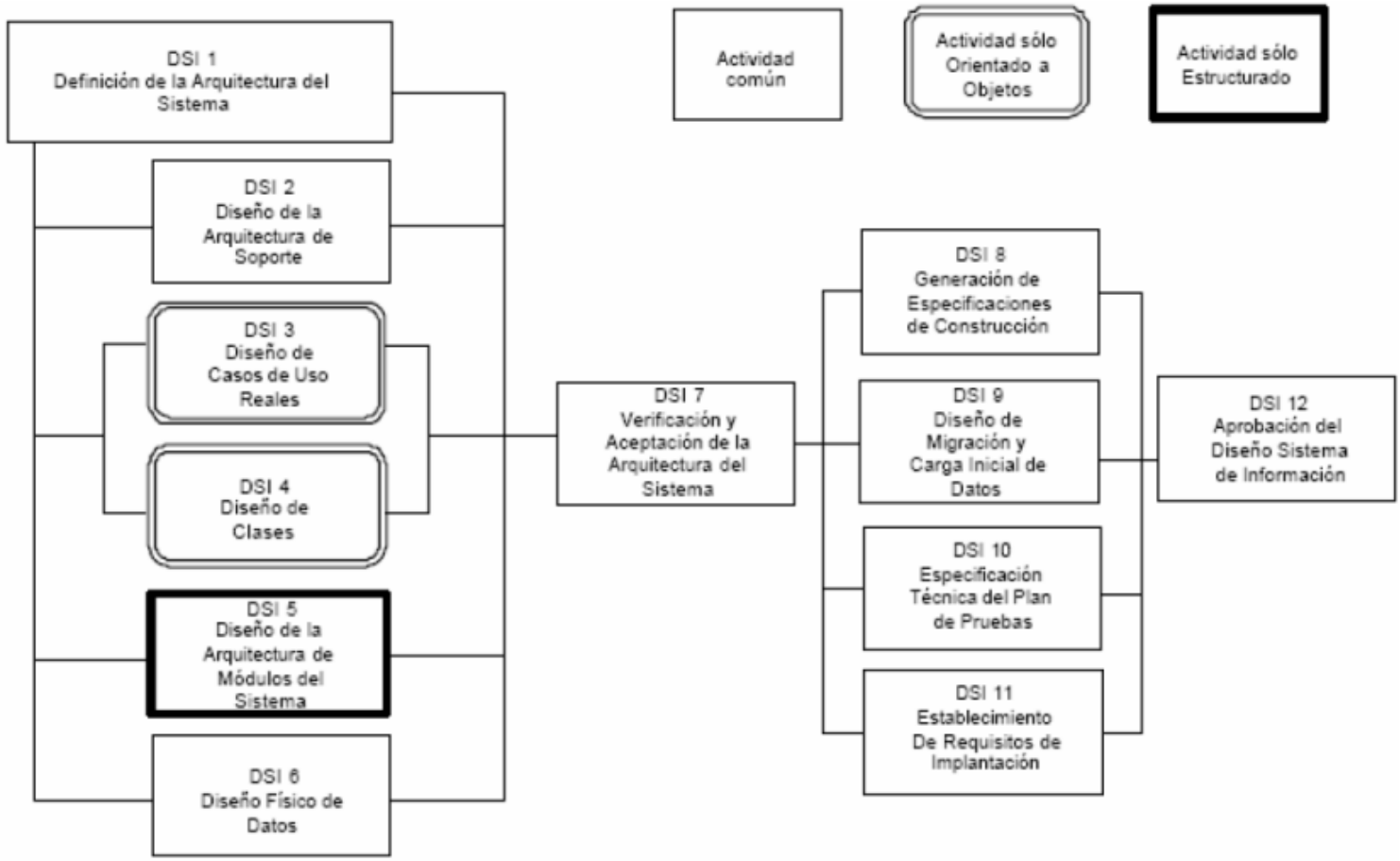
* Diseño de Casos de Uso Reales (DSI 3), con el diseño detallado del comportamiento del sistema de información para los casos de uso, el diseño de la interfaz de usuario y la validación de la división en subsistemas.
* Diseño de Clases (DSI 4), con el diseño detallado de cada una de las clases que forman parte del sistema, sus atributos, operaciones, relaciones y métodos, y la estructura jerárquica del mismo. En el caso de que sea necesario, se realiza la definición de un plan de migración y carga inicial de datos.

Una vez que se tiene el modelo de clases, se comienza el diseño físico en la actividad Diseño Físico de Datos (DSI 6), común con el enfoque estructurado.
Una vez finalizado el diseño de detalle, se realiza su revisión y validación en la actividad Verificación y Aceptación de la Arquitectura del Sistema (DSI 7), con el objeto de analizar la consistencia entre los distintos modelos y conseguir la aceptación del diseño por parte de los responsables de las áreas de Explotación y Sistemas.

- El segundo bloque de actividades complementa el diseño del sistema de información. En él se generan todas las especificaciones necesarias para la construcción del sistema de información:
    - Generación de Especificaciones de construcción (DSI 8), fijando las directrices para la construcción de los componentes del sistema, así como de las estructuras de datos.
    - Diseño de la Migración y Carga Inicial de Datos (DSI 9), en el que se definen los procedimientos de migración y sus componentes asociados, con las especificaciones de construcción oportunas.
    - Especificación Técnica del Plan de Pruebas (DSI 10), que incluye la definición y revisión del plan de pruebas, y el diseño de las verificaciones de los niveles de prueba establecidos. El catálogo de excepciones permite, de una forma muy ágil, establecer un conjunto de verificaciones relacionadas con el propio diseño o con la arquitectura del sistema.
    - Establecimiento de Requisitos de Implantación (DSI 11), que hace posible concretar las exigencias relacionados con la propia implantación del sistema, tales como formación de usuarios finales, infraestructura, etc.

Finalmente, en la actividad de Presentación y Aprobación del Diseño del Sistema de Información (DSI 12), se realiza una presentación formal y aprobación de los distintos productos del diseño.

En el siguiente gráfico se muestra la relación de actividades del proceso Diseño del Sistema de Información (DSI), tanto para Desarrollos Estructurados como para Desarrollos Orientados a Objetos.



Actividad DSI 1: Definición de la Arquitectura del Sistema

En esta actividad se define la arquitectura general del sistema de información, especificando las distintas particiones físicas del mismo, la descomposición lógica en subsistemas de diseño y la ubicación de cada subsistema en cada partición, así como la especificación detallada de la infraestructura tecnológica necesaria para dar soporte al sistema de información.

El particionamiento físico del sistema de información se especifica identificando los nodos y las comunicaciones entre los mismos, con cierta independencia de la infraestructura tecnológica que da soporte a cada nodo.

Con el fin de organizar y facilitar el diseño, se realiza una división del sistema de información en subsistemas de diseño, como partes lógicas coherentes y con interfaces claramente definidas. Se establece una distinción entre subsistemas específicos del sistema de información (en adelante, subsistemas específicos) y subsistemas de soporte, con la finalidad de independizar, en la medida de los posible, las funcionalidades a cubrir por el sistema de información de la infraestructura que le da soporte.

En la mayoría de los casos, los subsistemas específicos provienen directamente de las especificaciones de análisis y de los subsistemas de análisis, mientras que los subsistemas de soporte provienen de la necesidad de interacción del sistema de información con la infraestructura y con el resto de los sistemas, así como de la reutilización de módulos o subsistemas ya existentes en la instalación.

Debido a que la definición de los subsistemas de soporte puede exigir la participación de distintos perfiles técnicos, se propone el diseño de ambos tipos de subsistemas en actividades distintas, aunque en paralelo.

Una vez identificados y definidos los distintos subsistemas de diseño, se determina su ubicación óptima de acuerdo a la arquitectura propuesta. La asignación de dichos subsistemas a cada nodo permite disponer, en función de la carga de proceso y comunicación existente entre los nodos, de la información necesaria para realizar una estimación de las necesidades de infraestructura tecnológica que da soporte al sistema de información. Este factor es especialmente crítico en arquitecturas multinivel o cliente/ servidor, donde las comunicaciones son determinantes en el rendimiento final del sistema. Se propone crear un catálogo de excepciones en el que se especifiquen las situaciones anómalas o secundarias en el funcionamiento y ejecución del sistema de información, y que se irá completando a medida que se avance en el diseño detallado de los subsistemas.

En esta actividad también se establecen los requisitos, normas y estándares originados como consecuencia de la adopción de una determinada solución de arquitectura o infraestructura, que serán aplicables tanto en este proceso como en la Construcción del Sistema de Información (CSI). Se detallan a su vez, de acuerdo a las particularidades de la arquitectura del sistema propuesta, los requisitos de operación, seguridad y control, especificando los procedimientos necesarios para su cumplimiento.

Como resultado de esta actividad, se actualizan los catálogos de requisitos y normas, y se generan los siguientes productos:

- Diseño de la Arquitectura del Sistema, como producto que engloba el particionamiento físico del sistema de información y la descripción de subsistemas de diseño.
- Entorno Tecnológico del Sistema, que a su vez comprende la especificación del entorno tecnológico, las restricciones técnicas y la planificación de capacidades.
- Catálogo de Excepciones.
- Procedimientos de Operación y Administración del Sistema.
- Procedimientos de Seguridad y Control de Acceso.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 1.1	Definición de Niveles de Arquitectura	- Diseño de la Arquitectura del Sistema o Particionamiento Físico del Sistema de Información	- Diagrama de Representación - Diagrama de Despliegue	- Equipo de Arquitectura - Equipo de Soporte Técnico - Equipo de Seguridad
DSI 1.2	Identificación de Requisitos de Diseño y Construcción	- Catálogo de Requisitos	- Sesiones de Trabajo - Catalogación	- Equipo de Arquitectura - Equipo de Soporte Técnico
DSI 1.3	Especificación de Excepciones	- Catálogo de Excepciones	- Sesiones de Trabajo - Catalogación	- Equipo de Arquitectura - Equipo de Soporte Técnico
DSI 1.4	Especificación de Estándares y Normas de Diseño y Construcción	- Catálogo de Normas	- Sesiones de Trabajo - Catalogación	- Equipo de Arquitectura - Equipo de Soporte Técnico
DSI 1.5	Identificación de Subsistemas de Diseño	- Diseño de la Arquitectura del Sistema o Descripción de Subsistemas de Diseño	- Matricial - Diagrama de Estructura - Diagrama de Interacción de Objetos - Diagrama de Paquetes - Diagrama de Despliegue	- Equipo de Arquitectura - Equipo de Soporte Técnico - Equipo de Seguridad
DSI 1.6	Especificación del Entorno Tecnológico	- Entorno Tecnológico del Sistema: o Especificación del Entorno Tecnológico o Restricciones Técnicas o Estimación de Planificación de Capacidades	- Sesiones de Trabajo - Diagrama de Representación	- Equipo de Arquitectura - Equipo de Soporte Técnico
DSI 1.7	Especificación de Requisitos de Operación y Seguridad	- Procedimientos de Seguridad y Control de Acceso - Procedimientos de Operación y Administración del Sistema		- Equipo de Seguridad - Equipo de Arquitectura - Equipo de Soporte Técnico

Actividad DSI 2: Diseño de la Arquitectura de Soporte

En esta actividad se lleva a cabo la especificación de la arquitectura de soporte, que comprende el diseño de los subsistemas de soporte identificados en la actividad de Definición de la Arquitectura del Sistema (DSI 1), y la determinación de los mecanismos genéricos de diseño. Estos últimos sirven de guía en la utilización de diferentes estilos de diseño, tanto en el ámbito global del sistema de información, como en el diseño de detalle.

El diseño de los subsistemas de soporte, conceptualmente, es similar al diseño de los subsistemas específicos, aunque debe cumplir con unos objetivos claros de reutilización. De esta manera, se consigue simplificar y abstraer el diseño de los subsistemas específicos de la complejidad del entorno tecnológico, dotando al sistema de información de una mayor independencia de la infraestructura que le da soporte. Con este fin, se aconseja la consulta de los datos de otros proyectos existentes, disponible en el Histórico de Proyectos. Si esto no fuera suficiente, se puede contar en esta actividad con la participación de perfiles técnicos, con una visión global de la instalación.

Esta actividad se realiza en paralelo al diseño detallado, debido a que existe una constante realimentación, tanto en la especificación de los subsistemas con sus interfaces y dependencias, como en la aplicación de esqueletos o patrones en el diseño.

Los productos resultantes de esta actividad son:

- Diseño Detallado de los Subsistemas de Soporte.
- Mecanismos Genéricos de Diseño y Construcción.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 2.1	Diseño de Subsistemas de Soporte	- Diseño Detallado de los Subsistemas de Soporte	- Diagrama de Estructura - Diagrama de Interacción de Objetos - Diagrama de Clases	- Equipo de Arquitectura
DSI 2.2	Identificación de Mecanismos Genéricos de Diseño	- Mecanismos Genéricos de Diseño y Construcción	- Sesiones de Trabajo - Diagrama de Interacción de Objetos - Diagrama de Clases	- Equipo de Arquitectura

Actividad DSI 3: Diseño de Casos de Uso Reales

Esta actividad, que se realiza solo en el caso de **Diseño Orientado a Objetos**, tiene como propósito especificar el comportamiento del sistema de información para un caso de uso, mediante objetos o subsistemas de diseño que interactúan, y determinar las operaciones de las clases e interfaces de los distintos subsistemas de diseño.

Para ello, una vez identificadas las clases participantes dentro de un caso de uso, es necesario completar los escenarios que se recogen del análisis, incluyendo las clases de diseño que correspondan y teniendo en cuenta las restricciones del entorno tecnológico, esto es, detalles relacionados con la implementación del sistema. Es necesario realizar los comportamientos de excepción para dichos escenarios. Algunos de ellos pueden haber sido identificados en el proceso de análisis, aunque no se resuelven hasta este momento. Dichas excepciones se añadirán al catálogo de excepciones para facilitar las pruebas.

Algunos de los escenarios detallados requerirán una nueva interfaz de usuario. Por este motivo es necesario diseñar el formato de cada una de las pantallas o impresos identificados.

Es importante validar que los subsistemas definidos en la tarea Identificación de Subsistemas de Diseño (DSI 1.5) tienen la mínima interfaz con otros subsistemas. Por este motivo, se elaboran los escenarios al nivel de subsistemas y, de esta forma, se delimitan las interfaces necesarias para cada uno de ellos, teniendo en cuenta toda la funcionalidad del sistema que recogen los casos de uso. Además, durante esta actividad pueden surgir requisitos de implementación, que se recogen en el catálogo de requisitos.

Las tareas de esta actividad se realizan en paralelo con las de Diseño de Clases (DSI 4).

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 3.1	Identificación de Clases Asociadas a un Caso de Uso	- Diseño de la Realización de los Casos de Uso ○ Especificación Detallada	- Diagrama de Interacción de Objetos	- Equipo del Proyecto
DSI 3.2	Diseño de la Realización de los Casos de Uso	- Diseño de la Realización de los Casos de Uso ○ Especificación Detallada	- Diagrama de Interacción de Objetos	- Equipo del Proyecto

DSI 3.3	Revisión de la Interfaz de Usuario	- Diseño de Interfaz de Usuario:
		  ○ Formatos Individuales de Interfaz de Pantalla Gráfica
		  ○ Catálogo de Controles y Elementos de Diseño de Interfaz de Pantalla Gráfica
		  ○ Modelo de Navegación de Interfaz de Pantalla Gráfica
		  ○ Formatos de Impresión
		  ○ Prototipo de Interfaz de Pantalla Gráfica
			- Catalogación
			- Diagrama de Transición de Estados
			- Diagrama de Interacción de Objetos
			- Prototipado
				- Equipo del Proyecto
				- Usuarios Expertos

DSI 3.4	Revisión de Subsistemas de Diseño e Interfaces	- Diseño de la Realización de los Casos de Uso ○ Definición a Nivel de Subsistemas e Interfaz	- Diagrama de Interacción de Objetos	- Equipo del Proyecto - Equipo de Arquitectura

Actividad DSI 4: Diseño de Clases

El propósito de esta actividad, que se realiza sólo en el caso de **Diseño Orientado a Objetos**, es transformar el modelo de clases lógico, que proviene del análisis, en un modelo de clases de diseño. Dicho modelo recoge la especificación detallada de cada una de las clases, es decir, sus atributos, operaciones, métodos, y el diseño preciso de las relaciones establecidas entre ellas, bien sean de agregación, asociación o jerarquía. Para llevar a cabo todos estos puntos, se tienen en cuenta las decisiones tomadas sobre el entorno tecnológico y el entorno de desarrollo elegido para la implementación.

Se identifican las clases de diseño, que denominamos clases adicionales, en función del estudio de los escenarios de los casos de uso, que se está realizando en paralelo en la actividad Diseño de Casos de Uso Reales (DSI 3), y aplicando los mecanismos genéricos de diseño que se consideren convenientes por el tipo de especificaciones tecnológicas y de desarrollo. Entre ellas se encuentran clases abstractas, que integran características comunes con el objetivo de especializarlas en clases derivadas. Se diseñan las clases de interfaz de usuario, que provienen del análisis. Como consecuencia del estudio de los escenarios secundarios que se está realizando, pueden aparecer nuevas clases de interfaz.

También hay que considerar que, por el diseño de las asociaciones y agregaciones, pueden aparecer nuevas clases, o desaparecer incluyendo sus atributos y métodos en otras, si se considera conveniente por temas de optimización.

La jerarquía entre las clases se va estableciendo a lo largo de esta actividad, a medida que se van identificando comportamientos comunes en las clases, aunque haya una tarea propia de diseño de la jerarquía. Otro de los objetivos del diseño de las clases es identificar para cada clase, los atributos, las operaciones que cubren las responsabilidades que se identificaron en el análisis, y la especificación de los métodos que implementan esas operaciones, analizando los escenarios del Diseño de Casos de Uso

Reales (DSI 3). Se determina la visibilidad de los atributos y operaciones de cada clase, con respecto a las otras clases del modelo.

Una vez que se ha elaborado el modelo de clases, se define la estructura física de los datos correspondiente a ese modelo, en la actividad Diseño Físico de Datos (DSI 6).

Además, en los casos en que sea necesaria una migración de datos de otros sistemas o una carga inicial de información, se realizará su especificación a partir del modelo de clases y las estructuras de datos de los sistemas origen.

Como resultado de todo lo anterior se actualiza el modelo de clases del análisis, una vez recogidas las decisiones de diseño.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 4.1	Identificación de Clases Adicionales	- Modelo de Clases de Diseño	- Diagrama de Clases	- Equipo del Proyecto
DSI 4.2	Diseño de Asociaciones y Agregaciones	- Modelo de Clases de Diseño	- Diagrama de Clases	- Equipo del Proyecto
DSI 4.3	Identificación de Atributos de las Clases	- Modelo de Clases de Diseño	- Diagrama de Clases	- Equipo del Proyecto
DSI 4.4	Identificación de Operaciones de las Clases	- Modelo de Clases de Diseño - Comportamiento de Clases de Diseño	- Diagrama de Clases - Diagrama de Transición de Estados	- Equipo del Proyecto
DSI 4.5	Diseño de la Jerarquía	- Modelo de Clases de Diseño	- Diagrama de Clases	- Equipo del Proyecto
DSI 4.6	Descripción de Métodos de las Operaciones	- Modelo de Clases de Diseño	- Diagrama de Clases	- Equipo del Proyecto
DSI 4.7	Especificación de Necesidades de Migración y Carga Inicial de Datos	- Plan de Migración y Carga Inicial de Datos	- Sesiones de Trabajo	- Analistas - Usuarios Expertos

Actividad DSI 5: Diseño de la Arquitectura de Módulos del Sistema

El objetivo de esta actividad, que sólo se realiza en el caso de **Diseño Estructurado**, es definir los módulos del sistema de información, y la manera en que van a interactuar unos con otros, intentando que cada módulo trate total o parcialmente un proceso específico y tenga una interfaz sencilla.

Para cada uno de los subsistemas específicos, identificados en la tarea Identificación de los Subsistemas de Diseño (DSI 1.5), se diseña la estructura modular de los procesos que lo integran, tomando como punto de partida los modelos obtenidos en la tarea Validación de los Modelos (ASI 9.3) del proceso de Análisis del Sistema de Información (ASI) y el catálogo de requisitos. Dicha estructura se irá completando con los módulos que vayan apareciendo como consecuencia del diseño de la interfaz de usuario, así como de la optimización del diseño físico de datos.

Durante el diseño de los módulos, se pueden identificar características o comportamientos comunes relacionados con accesos a las bases de datos o ficheros, lógica de tratamiento, llamadas a otros módulos, gestión de errores, etc. que determinen la necesidad de realizar su implementación como subsistemas de soporte.

Además, se analizan los comportamientos de excepción asociados a los diferentes módulos y a las interfaces entre los mismos, intentando independizar en la medida de lo posible aquéllos que presenten un tratamiento común. Dichas excepciones se incorporan al catálogo de excepciones.

En esta actividad, se consideran los estándares y normas establecidas para el diseño, aplicando, cuando proceda, los mecanismos genéricos de diseño identificados en la tarea Identificación de Mecanismos Genéricos de Diseño (DSI 2.2).

Las tareas de esta actividad no se realizan de forma secuencial, sino en paralelo, con continuas realimentaciones entre ellas y con las realizadas en las actividades Definición de la Arquitectura del Sistema (DSI 1), Diseño de la Arquitectura de Soporte (DSI 2) y Diseño Físico de Datos (DSI 6).

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 5.1	Diseño de Módulos del Sistema	- Diseño de la Arquitectura Modular del Sistema	- Diagrama de Estructura	- Equipo de Arquitectura - Equipo del Proyecto
DSI 5.2	Diseño de Comunicaciones entre Módulos	- Diseño de la Arquitectura Modular del Sistema	- Diagrama de Estructura	- Equipo de Arquitectura - Equipo del Proyecto - Equipo de Seguridad
DSI 5.3	Revisión de la Interfaz de Usuario	- Diseño de Interfaz de Usuario: o Descomposición Funcional en Diálogos o Formatos Individuales de Interfaz de Pantalla o Catálogo de Controles y Elementos de Diseño de Interfaz de Pantalla	- Diagrama de Descomposición Funcional - Diagrama de Transición de Estados - Matricial - Catalogación - Prototipado	- Equipo del Proyecto - Usuarios Expertos
		o Modelo de Navegación de Interfaz de Pantalla o Formatos de Impresión o Prototipo de Interfaz de Pantalla o Prototipo de Interfaz de Impresión		

Actividad DSI 6: Diseño Físico de Datos

En esta actividad se define la estructura física de datos que utilizará el sistema, a partir del modelo lógico de datos normalizado o modelo de clases, de manera que teniendo presentes las características específicas del sistema de gestión de datos concreto a utilizar, los requisitos establecidos para el sistema de información, y las particularidades del entorno tecnológico, se consiga una mayor eficiencia en el tratamiento de los datos.

También se analizan los caminos de acceso a los datos utilizados por cada módulo/clase del sistema en consultas y actualizaciones, con el fin de mejorar los tiempos de respuesta y optimizar los recursos de máquina.

Las tareas de esta actividad se realizan de forma iterativa y en paralelo con las realizadas en las actividades Definición de la Arquitectura del Sistema (DSI 1), donde se especifican los detalles de arquitectura e infraestructura y la planificación de capacidades, Diseño de la Arquitectura de Soporte (DSI 2), dónde se determinan y diseñan los servicios comunes que pueden estar relacionados con la gestión de datos (acceso a bases de datos, ficheros,

áreas temporales, sincronización de bases de datos, etc.), Diseño de Casos de Uso Reales y de Clases (DSI 3 y 4), para desarrollo orientado a objetos, y Diseño de la Arquitectura de Módulos del Sistema (DSI 5), para desarrollo estructurado, dónde se especifica la lógica de tratamiento y las interfaces utilizadas.

En el caso de diseño orientado a objetos, esta actividad también es necesaria. La obtención del modelo físico de datos se realiza aplicando una serie de reglas de transformación a cada elemento del modelo de clases que se está generando en la actividad Diseño de Clases (DSI 4).

Asimismo, en esta actividad hay que considerar los estándares y normas establecidos para el diseño aplicando, cuando proceda, los mecanismos genéricos de diseño identificados en la tarea Identificación de Mecanismos Genéricos de Diseño (DSI 2.2).

A continuación se incluye una table resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 6.1	Diseño del Modelo Físico de Datos	- Modelo Físico de Datos	- Reglas de Obtención del Modelo Físico a Partir del Lógico - Reglas de Transformación	- Equipo de Arquitectura - Equipo del Proyecto - Administradores de Bases de Datos
DSI 6.2	Especificación de los Caminos de Acceso a los Datos	- Especificación de los Caminos de Acceso a los Datos	- Cálculo de Accesos Físicos - Caminos de Acceso	- Equipo del Proyecto
DSI 6.3	Optimización del Modelo Físico de Datos	- Modelo Físico de Datos Optimizado	- Optimización	- Equipo de Arquitectura - Equipo del Proyecto - Administradores de Bases de Datos - Equipo de Seguridad
DSI 6.4	Especificación de la Distribución de Datos	- Esquemas Físicos de Datos - Asignación esquemas Físicos de Datos a Nodos	- Matricial	- Equipo de Arquitectura - Equipo de Soporte Técnico

Actividad DSI 7: Verificación y Aceptación de la Arquitectura del Sistema

El objetivo de esta actividad es garantizar la calidad de las especificaciones del diseño del sistema de información y la viabilidad del mismo, como paso previo a la generación de las especificaciones de construcción.

Para cumplir dicho objetivo, se llevan a cabo las siguientes acciones:

- Verificación de la calidad técnica de cada modelo o especificación.
- Aseguramiento de la coherencia entre los distintos modelos.
- Aceptación del diseño de la arquitectura por parte de Explotación y Sistemas.

Esta actividad es compleja, por lo que es aconsejable utilizar herramientas de apoyo para la realización de sus tareas.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 7.1	Verificación de las Especificaciones de Diseño	- Entorno Tecnológico del Sistema - Diseño de la Arquitectura del Sistema - Diseño Detallado de Subsistemas de Soporte - Modelo Físico de Datos Optimizado - Esquemas Físicos de Datos - Asignación de Esquemas Físicos de Datos a Nodos - Diseño de Interfaz de Usuario ***Estructurado:*** - Diseño de la Arquitectura Modular ***Orientación a Objetos:*** - Diseño de la Realización de los Casos de Uso - Modelo de Clases de Diseño - Comportamiento de Clases de Diseño		- Equipo de Arquitectura - Equipo del Proyecto
DSI 7.2	Análisis de Consistencia de las Especificaciones de Diseño	- Entorno Tecnológico del Sistema - Diseño de la Arquitectura del Sistema - Diseño Detallado de Subsistemas de Soporte - Modelo Físico de Datos Optimizado - Esquemas Físicos de Datos - Asignación de Esquemas Físicos de Datos a Nodos - Diseño de Interfaz de Usuario ***Estructurado:*** - Diseño de la Arquitectura Modular ***Orientación a Objetos:*** - Diseño de la Realización de los Casos de Uso - Modelo de Clases de Diseño - Comportamiento de Clases de Diseño	- Matricial	- Equipo de Arquitectura - Equipo del Proyecto
DSI 7.3	Aceptación de la Arquitectura del Sistema	- Aceptación Técnica del Diseño		- Jefe de Proyecto - Responsable de Operación - Responsable de Sistemas

Actividad DSI 8: Generación de Especificaciones de Construcción

En esta actividad se generan las especificaciones para la construcción del sistema de información, a partir del diseño detallado.

Estas especificaciones definen la construcción del sistema de información a partir de las unidades básicas de construcción (en adelante, componentes), entendiendo como tales unidades independientes y coherentes de construcción y ejecución, que se corresponden con un empaquetamiento físico de los elementos del diseño de detalle, como pueden ser módulos, clases o especificaciones de interfaz.

La división del sistema de información en subsistemas de diseño proporciona, por continuidad, una primera división en subsistemas de construcción, definiendo para cada uno de ellos los componentes que lo integran. Si se considera necesario, un subsistema de diseño se podrá dividir a su vez en sucesivos niveles para mayor claridad de las especificaciones de construcción.

Las dependencias entre subsistemas de diseño proporcionan información para establecer las dependencias entre los subsistemas de construcción y, por lo tanto, definir el orden o secuencia que se debe seguir en la construcción y en la realización de las pruebas.

También se generan las especificaciones necesarias para la creación de las estructuras de datos en los gestores de bases de datos o sistemas de ficheros.

El producto resultante de esta actividad es el conjunto de las especificaciones de construcción del sistema de información, que comprende:

- Especificación del entorno de construcción.
- Descripción de subsistemas de construcción y dependencias.
- Descripción de componentes.
- Plan de integración del sistema de información.
- Especificación detallada de componentes.
- Especificación de la estructura física de datos.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 8.1	Especificación del Entorno de Construcción	- Especificaciones de Construcción del Sistema de Información: o Especificación del Entorno de Construcción		- Equipo de Arquitectura - Equipo del Proyecto - Equipo de Soporte Técnico - Equipo de Sistemas - Equipo de Seguridad
DSI 8.2	Definición de Componentes y Subsistemas de Construcción	- Especificaciones de Construcción del Sistema de Información: o Descripción de Subsistemas de Construcción y Dependencias o Descripción de Componentes o Plan de Integración del Sistema de Información	- Diagrama de Estructura - Matricial - Diagrama de Componentes - Diagrama de Despliegue	- Equipo de Arquitectura - Equipo del Proyecto
DSI 8.3	Elaboración de Especificaciones de Construcción	- Especificaciones de Construcción del Sistema de Información: o Especificación Detallada de Componentes	- Diagrama de Componentes	- Equipo del Proyecto
DSI 8.4	Elaboración de Especificaciones del Modelo Físico de Datos	- Especificaciones de Construcción del Sistema de Información: o Especificación de la Estructura Física de Datos		- Equipo del Proyecto - Administradores de la Base de Datos

Actividad DSI 9: Diseño de la Migración y Carga Inicial de Datos

Esta actividad sólo se lleva a cabo cuando es necesaria una carga inicial de información, o una migración de datos de otros sistemas, cuyo alcance y estrategia a seguir se habrá establecido previamente.

Para ello, se toma como referencia el plan de migración y carga inicial de datos, que recoge las estructuras físicas de datos del sistema o sistemas origen implicadas en la conversión, la prioridad en las cargas y secuencia a seguir, las necesidades previas de depuración de la información, así como los requisitos necesarios para garantizar la correcta implementación de los procedimientos de migración sin comprometer el funcionamiento de los sistemas actuales.

A partir de dicho plan, y de acuerdo a la estructura física de los datos del nuevo sistema, obtenida en la actividad Diseño Físico de Datos (DSI 6), y a las características de la arquitectura y del entorno tecnológico propuesto en la actividad Definición de la Arquitectura del Sistema (DSI 1), se procede a definir y diseñar en detalle los procedimientos y procesos necesarios para realizar la migración.

Se completa el plan de pruebas específico establecido en el plan de migración y carga inicial, detallando las pruebas a realizar, los criterios de aceptación o rechazo de la prueba y los responsables de la organización, realización y evaluación de resultados.

Asimismo, se determinan las necesidades adicionales de infraestructura, tanto para la implementación de los procesos como para la realización de las pruebas.

Como resultado de esta actividad, se actualiza el plan de migración y carga inicial de datos con la información siguiente:

- Especificación del entorno de migración.
- Definición de procedimientos de migración.
- Diseño detallado de módulos.
- Especificación técnica de las pruebas.
- Planificación de la migración y carga inicial.

Es importante considerar que una carga inicial de información no tiene el mismo alcance y complejidad que una migración de datos, de modo que las tareas de esta actividad se deben llevar a cabo en mayor o menor medida en función de las características de los datos a cargar.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 9.1	Especificación del Entorno de Migración	- Plan de Migración y Carga Inicial de Datos: ○ Especificación del Entorno de Migración y Carga Inicial		- Equipo de Arquitectura - Equipo de Soporte Técnico
DSI 9.2	Diseño de Procedimientos de Migración y Carga Inicial	- Plan de Migración y Carga Inicial de Datos: ○ Definición de Procedimientos de Migración y Carga Inicial		- Equipo de Arquitectura - Equipo del Proyecto - Equipo de Seguridad

DSI 9.3	Diseño Detallado de Componentes de Migración y Carga Inicial	- Plan de Migración y Carga Inicial de Datos:
		  ○ Diseño Detallado de Módulos de Migración y Carga Inicial
		  ○ Especificación Técnica de las Pruebas de Migración y Carga Inicial
				- Equipo del Proyecto

DSI 9.4	Revisión de la Planificación de la Migración	- Plan de Migración y Carga Inicial de Datos: ○ Planificación de la Migración y Carga Inicial		- Jefe de Proyecto

Actividad DSI 10: Especificación Técnica del Plan de Pruebas

En esta actividad se realiza la especificación de detalle del plan de pruebas del sistema de información para cada uno de los niveles de prueba establecidos en el proceso Análisis del Sistema de Información:

- Pruebas unitarias.
- Pruebas de integración.
- Pruebas del sistema.
- Pruebas de implantación.
- Pruebas de aceptación.

Para ello se toma como referencia el plan de pruebas, que recoge los objetivos de la prueba de un sistema, establece y coordina una estrategia de trabajo, y provee del marco adecuado para planificar paso a paso las actividades de prueba. También puede ser una referencia el plan de integración del sistema de información propuesto, opcionalmente, en la tarea Definición de Componentes y Subsistemas de Construcción (DSI 8.2).

El catálogo de requisitos, el catálogo de excepciones y el diseño detallado del sistema de información, permiten la definición de las verificaciones que deben realizarse en cada nivel de prueba para comprobar que el sistema responde a los requisitos planteados. La asociación de las distintas verificaciones a componentes, grupos de componentes y subsistemas, o al sistema de información completo, determina las distintas verificaciones de cada nivel de prueba establecido.

Las pruebas unitarias comprenden las verificaciones asociadas a cada componente del sistema de información. Su realización tiene como objetivo verificar la funcionalidad y estructura de cada componente individual.

Las pruebas de integración comprenden verificaciones asociadas a grupos de componentes, generalmente reflejados en la definición de subsistemas de construcción o en el plan de integración del sistema de información. Tienen por objetivo verificar el correcto ensamblaje entre los distintos componentes.

Las pruebas del sistema, de implantación y de aceptación corresponden a verificaciones asociadas al sistema de información, y reflejan distintos propósitos en cada tipo de prueba:

- Las pruebas del sistema son pruebas de integración del sistema de información completo. Permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen.
- Las pruebas de implantación incluyen las verificaciones necesarias para asegurar que el sistema funcionará correctamente en el entorno de operación al responder satisfactoriamente a los requisitos de rendimiento, seguridad y operación, y coexistencia con el resto de los sistemas de la instalación, y conseguir la aceptación del sistema por parte del usuario de operación.
- Las pruebas de aceptación van dirigidas a validar que el sistema cumple los requisitos de funcionamiento esperado, recogidos en el catálogo de requisitos y en los criterios de aceptación del sistema de información, y conseguir la aceptación final del sistema por parte del usuario.

Las pruebas unitarias, de integración y del sistema se llevan a cabo en el proceso Construcción del Sistema de Información (CSI), mientras que las pruebas de implantación y aceptación se realizan en el proceso implantación y Aceptación del Sistemas (IAS).

Como resultado de esta actividad se actualiza el plan de pruebas con la información siguiente:

- Especificación del entorno de pruebas.
- Especificación técnica de niveles de prueba.
- Planificación de las pruebas.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 10.1	Especificación del Entorno de Pruebas	- Plan de Pruebas: ○ Especificación del Entorno de Pruebas		- Equipo de Arquitectura - Equipo de Soporte Técnico - Equipo del Proyecto - Equipo de Seguridad
DSI 10.2	Especificación Técnica de Niveles de Prueba	- Plan de Pruebas: ○ Especificación Técnica de Niveles de Prueba		- Jefe de Proyecto - Analistas - Usuarios Expertos
DSI 10.3	Revisión de la Planificación de Pruebas	- Plan de Pruebas: ○ Planificación de las Pruebas		- Jefe de Proyecto

Actividad DSI 11: Establecimiento de Requisitos de Implantación

En esta actividad se completa el catálogo de requisitos con aquéllos relacionados con la documentación que el usuario requiere para operar con el nuevo sistema, y los relativos a la propia implantación del sistema en el entorno de operación.

La incorporación de estos requisitos permite ir preparando, en los proceso de Construcción del Sistema de Información (CSI) e Implantación y Aceptación del Sistema (IAS), los medios y recursos necesarios para que los usuarios, tanto finales como de operación, sean capaces de utilizar el nuevo sistema de forma satisfactoria.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 11.1	Especificación de Requisitos de Documentación de Usuario	- Catálogo de Requisitos	- Catalogación - Sesiones de Trabajo	- Jefe de Proyecto - Analistas - Usuarios Expertos - Responsable de Operación - Responsable de Sistemas
DSI 11.2	Especificación de Requisitos de Implantación	- Catálogo de Requisitos	- Catalogación - Sesiones de Trabajo	- Jefe de Proyecto - Directores de Usuarios - Equipo de Soporte Técnico

Actividad DSI 12: Aprobación del Diseño del Sistema de Información

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
DSI 12.1	Presentación y Aprobación del Diseño del Sistema de Información	- Aprobación del Diseño del Sistema de Información	- Presentación	- Comité de Dirección - Jefe de Proyecto

Construcción del Sistema de Información

Descripción y Objetivos

En este proceso se genera el código de los componentes del Sistema de Información, se desarrollan todos los procedimientos de operación y seguridad y se elaboran todos los manuales de usuario final y de explotación con el objetivo de asegurar el correcto funcionamiento del Sistema para su posterior implantación.

Para conseguir dicho objetivo, en este proceso se realizan las pruebas unitarias, las pruebas de integración de los subsistemas y componentes y las pruebas del sistema, de acuerdo al plan de pruebas establecido.

Asimismo, se define la formación de usuario final y, si procede, se construyen los procedimientos de migración y carga inicial de datos.

Al ser MÉTRICA Versión 3 una metodología que cubre tanto desarrollos estructurados como orientados a objetos, las actividades de ambas aproximaciones están integradas en una estructura común.

El producto Especificaciones de Construcción del Sistema de Información, obtenido en la actividad de Generación de Especificaciones de Construcción (DSI 8), es la base para la construcción del sistema de información. En dicho producto se recoge la información relativa al entorno de construcción del sistema de información, la especificación detallada de los componentes y la descripción de la estructura física de datos, tanto bases de datos como sistemas de ficheros. Opcionalmente, incluye un plan de integración del sistema de información, en el que se especifica la secuencia y organización de la construcción de los distintos componentes.

En la actividad Preparación del Entorno de Generación y Construcción (CSI 1), se asegura la disponibilidad de la infraestructura necesaria para la generación del código de los componentes y procedimientos del sistema de información.

Una vez configurado el entorno de construcción, se realiza la codificación y las pruebas de los distintos componentes que conforman el sistema de información, en las actividades:

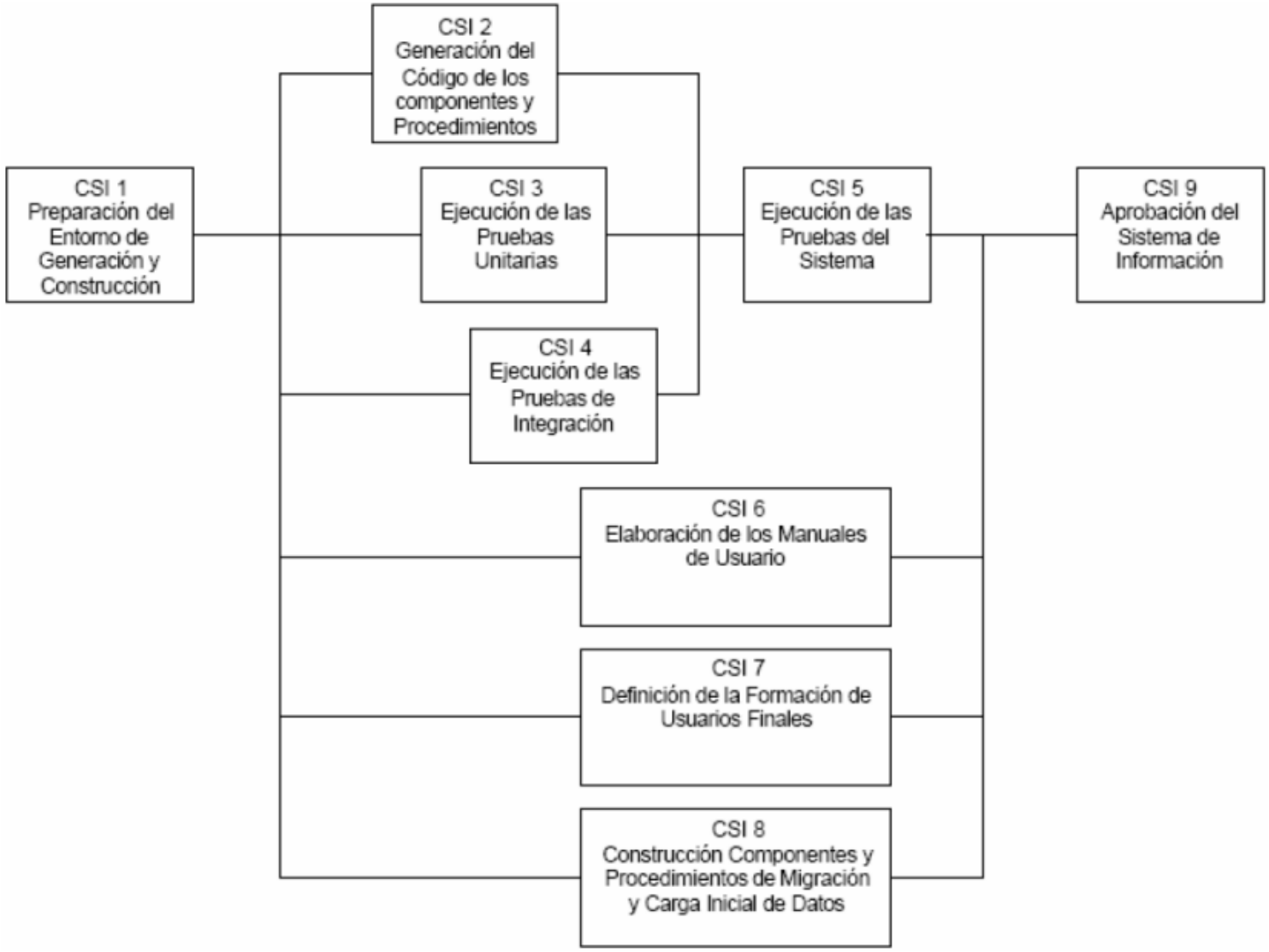
- Generación del Código de los Componentes y Procedimientos (CSI 2), que se hace según las especificaciones de construcción del sistema de información, y conforme al plan de integración del sistema de información.
- Ejecución de las Pruebas Unitarias (CSI 3), donde se llevan a cabo las verificaciones definidas en el plan de pruebas para cada uno de los componentes.
- Ejecución de las Pruebas de Integración (CSI 4), que incluye la ejecución de las verificaciones asociadas a los subsistemas y componentes, a partir de los componentes verificados individualmente, y la evaluación de los resultados.

Una vez construido el sistema de información y realizadas las verificaciones correspondientes, se lleva a cabo la integración final del sistema de información en la actividad Ejecución de las Pruebas del Sistema (CSI 5), comprobando tanto las interfaces entre subsistemas y sistemas externos como los requisitos, de acuerdo a las verificaciones establecidas en el plan de pruebas para el nivel de pruebas del sistema.

En la actividad Elaboración de los Manuales de Usuario (CSI 6), se genera la documentación de usuario final o explotación, conforme a los requisitos definidos en el proceso Diseño del Sistema de Información.

La formación necesaria para que los usuarios finales sean capaces de utilizar el sistema de forma satisfactoria se especifica en la actividad Definición de la Formación de Usuarios Finales (CSI 7).

Si se ha establecido la necesidad de realizar una migración de datos, la construcción y pruebas de los componentes y procedimientos relativos a dicha migración y a la carga inicial de datos se realiza en la actividad Construcción de los Componentes y Procedimientos de Migración y Carga Inicial de Datos (CSI 8).



Actividad CSI 1: Preparación del Entorno de Generación y Construcción

El objetivo de esta actividad es asegurar la disponibilidad de todos los medios y facilidades para que se pueda llevar a cabo la construcción del sistema de información. Entre estos medios, cabe destacar la preparación de los puestos de trabajo, equipos físicos y lógicos, gestores de bases de datos, bibliotecas de programas, herramientas de generación de código, bases de datos o ficheros de prueba, entre otros.

Las características del entorno de construcción y sus requisitos de operación y seguridad, así como las especificaciones de construcción de la estructura física de datos, se establecen en la actividad Generación de Especificaciones de Construcción (DSI 8), y constituyen el punto de partida para la realización de esta actividad.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 1.1	Implantación de la Base de Datos Física o Ficheros	- Base de Datos Física o Sistema de Ficheros		- Equipo del Proyecto - Administradores de Bases de Datos
CSI 1.2	Preparación del Entorno de Construcción	- Entorno de Construcción		- Equipo del Proyecto - Técnicos de Sistemas - Equipo de Operación - Administradores de Bases de Datos

Actividad CSI 2: Generación del Código de los Componentes y Procedimientos

El objetivo de esta actividad es la codificación de los componentes del sistema de información, a partir de las especificaciones de construcción obtenidas en el proceso Diseño del Sistema de Información (DSI), así como la construcción de los procedimientos de operación y seguridad establecidos para el mismo.

En paralelo a esta actividad, se desarrollan las actividades relacionadas con las pruebas unitarias y de integración del sistema de información. Esto permite una construcción incremental, en el caso de que así se haya especificado en el plan de pruebas y en el plan de integración del sistema de información.

A continuación se incluye una tabla resumen con las tareas de la presenta actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 2.1	Generación del Código de Componentes	- Producto Software: o Código Fuente de los Componentes		- Programadores
CSI 2.2	Generación del Código de los Procedimientos de Operación y Seguridad	- Producto Software: o Procedimientos de Operación y Administración del Sistema o Procedimientos de Seguridad y Control de Acceso		- Técnicos de Sistemas - Equipo de Operación - Administrador de la Base de Datos - Programadores

Actividad CSI 3: Ejecución de las Pruebas Unitarias

En esta actividad se realizan las pruebas unitarias de cada uno de los componentes del sistema de información, una vez codificados, con el objeto de comprobar que su estructura es correcta y que se ajustan a la funcionalidad establecida.

En el plan de pruebas se ha definido el entorno necesario para la realización de cada nivel de prueba, así como las verificaciones asociadas a las pruebas unitarias, la coordinación y secuencia a seguir en la ejecución de las mismas y los criterios de registro y aceptación de los resultados.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 3.1	Preparación del Entorno de Pruebas Unitarias	- Entorno de pruebas unitarias		- Técnicos de Sistemas - Programadores
CSI 3.2	Realización y evaluación de las Pruebas Unitarias	- Resultado de las pruebas unitarias	- Pruebas Unitarias	- Programadores

Actividad CSI 4: Ejecución de las Pruebas de Integración

El objetivo de las pruebas de integración es verificar si los componentes o subsistemas interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida, y se ajustan a los requisitos especificados en las verificaciones correspondientes.

La estrategia a seguir en las pruebas de integración se establece en el plan de pruebas, donde se habrá tenido en cuenta el plan de integración del sistema de información, siempre y cuando se haya especificado en la tarea Definición de Componentes y Subsistemas de Construcción (DSI 8.2).

Esta actividad se realiza en paralelo a las actividades Generación del Código de los Componentes y Procedimientos (CSI 2) y Ejecución de las Pruebas Unitarias (CSI 3). Sin embargo, es necesario que los componentes objeto de las pruebas de integración se hayan verificado de manera unitaria.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 4.1	Preparación del Entorno de las Pruebas de Integración	- Entorno de Pruebas de Integración		- Técnicos de Sistemas - Técnicos de Comunicaciones - Equipo de Arquitectura - Equipo del Proyecto
CSI 4.2	Realización de las Pruebas de Integración	- Resultado de las Pruebas de Integración	- Pruebas de integración	- Equipo del Proyecto
CSI 4.3	Evaluación del Resultado de las Pruebas de Integración	- Evaluación del Resultado de las Pruebas de Integración		- Analistas

Actividad CSI 5: Ejecución de las Pruebas del Sistema

El objetivo de las pruebas del sistema es comprobar la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.

En la realización de estas pruebas es importante comprobar la cobertura de los requisitos, dado que su incumplimiento puede comprometer la aceptación del sistema por el equipo de operación responsable de realizar las pruebas de implantación del sistema, que se llevarán a cabo en el proceso Implantación y Aceptación del Sistema.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 5.1	Preparación del Entorno de las Pruebas del Sistema	- Entorno de Pruebas del Sistema		- Técnicos de Sistemas - Técnicos de Comunicaciones - Equipo de Arquitectura - Equipo del Proyecto
CSI 5.2	Realización de las Pruebas del Sistema	- Resultado de las Pruebas del Sistema	- Pruebas del Sistema	- Equipo del Proyecto
CSI 5.3	Evaluación del Resultado de las Pruebas del Sistema	- Evaluación del Resultado de las Pruebas del Sistema		- Analistas - Jefe de Proyecto

Actividad CSI 6: Elaboración de los Manuales de Usuario

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 6.1	Elaboración de los Manuales de Usuario	- Producto Software: ○ Manuales de Usuario		- Equipo del Proyecto

Actividad CSI 7: Definición de la Formación de Usuarios Finales

En esta actividad se establecen las necesidades de formación del usuario final, con el objetivo de conseguir la explotación eficaz del nuevo sistema.

Para la definición de la formación hay que tener en cuenta las características funcionales y técnicas propias del sistema de información, así como los requisitos relacionados con la formación del usuario final, establecidos en la tarea Especificación de Requisitos de Implantación (DSI 11.2).

El producto resultante de esta actividad es la especificación de la formación de usuarios finales, que consta de los siguientes elementos:

- Esquema de formación.
- Materiales y entornos de formación.

En el proceso Implantación y Aceptación del Sistema (IAS), se unifican las especificaciones de formación de cada sistema de información implicado en la implantación y se elabora un único plan de formación que esté alineado con el plan de implantación del sistema.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 7.1	Definición del Esquema de Formación	- Especificación de la Formación a Usuarios Finales: o Esquema de Formación		- Analistas
CSI 7.2	Especificación de los Recursos y Entornos de Formación	- Especificación de la Formación a Usuarios Finales: o Materiales y Entornos de Formación		- Analistas - Equipo de Formación

Actividad CSI 8: Construcción de los Componentes y Procedimientos de Migración y Carga Inicial de Datos

El objetivo de esta actividad es la codificación y prueba de los componentes y procedimientos de migración y carga inicial de datos, a partir de las especificaciones recogidas en el plan de migración y carga inicial de datos obtenidos en el proceso Diseño del Sistema de Información.

Previamente a la generación del código, se prepara la infraestructura tecnológica necesaria para realizar la codificación y las pruebas de los distintos componentes y procedimientos asociados, de acuerdo a las características del entorno de migración especificado en el plan de migración y carga inicial de datos.

Finalmente, se llevan a cabo las verificaciones establecidas en la especificación técnica del plan de pruebas propio de la migración.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 8.1	Preparación del Entorno de Migración y Carga Inicial de Datos	- Entorno de Migración		- Equipo del Proyecto - Técnicos de Sistemas - Equipo de Operación - Equipo de Seguridad - Administradores de Bases de Datos
CSI 8.2	Generación del Código de los Componentes y Procedimientos de Migración y Carga Inicial de Datos	- Código Fuente de los Componentes de Migración y Carga Inicial de Datos - Procedimientos de Migración y Carga Inicial de Datos		- Programadores
CSI 8.3	Realización y Evaluación de las Pruebas de Migración y Carga Inicial de Datos	- Resultado de las pruebas de migración y carga inicial de datos - Evaluación del resultado de las pruebas de migración y carga inicial de datos	- Pruebas Unitarias - Pruebas de Integración	- Equipo del Proyecto

Actividad CSI 9: Aprobación del Sistema de Información

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
CSI 9.1	Presentación y Aprobación del Sistema de Información	- Sistema de Información: o Aprobación del Sistema de Información		- Comité de Seguimiento - Jefe de Proyecto

Implantación y Aceptación del Sistema

Descripción y Objetivos

Este proceso tiene como objetivo principal la entrega del sistema en su totalidad, y la realización de todas las actividades necesarias para el paso a producción del mismo.

En primer lugar, se revisa la estrategia de implantación que ya se determinó en el proceso Estudio de Viabilidad del Sistema (EVS). Se estudia su alcance y, en función de sus características, se define un plan de implantación y se especifica el equipo que lo va a llevar a cabo. Conviene señalar la participación del usuario de operación en las pruebas de implantación, del usuario final en las pruebas de aceptación, y del responsable de mantenimiento.

Las actividades previas al inicio de la producción incluyen la preparación de la infraestructura necesaria para configurar el entorno, la instalación de los componentes, la activación de los procedimientos manuales y automáticos asociados y, cuando proceda, la migración o carga inicial de datos. Para ello se toman como punto de partida los productos software probados, obtenidos en el proceso Construcción del Sistema de Información (CSI) y su documentación asociada. Se realizan las pruebas de implantación y de aceptación del sistema en su totalidad. Responden a los siguientes propósitos:

- Las pruebas de implantación cubren un rango muy amplio, que va desde la comprobación de cualquier detalle de diseño interno hasta aspectos tales como las comunicaciones. Se debe comprobar que el sistema puede gestionar los volúmenes de información requeridos, se ajusta a los tiempos de respuesta deseados y que los procedimientos de respaldo, seguridad e interfaces con otros sistemas funcionan correctamente. Se debe verificar también el comportamiento del sistema bajo las condiciones más extremas.
- Las pruebas de aceptación se realizan por y para los usuarios. Tienen como objetivo validar formalmente que el sistema se ajusta a sus necesidades.

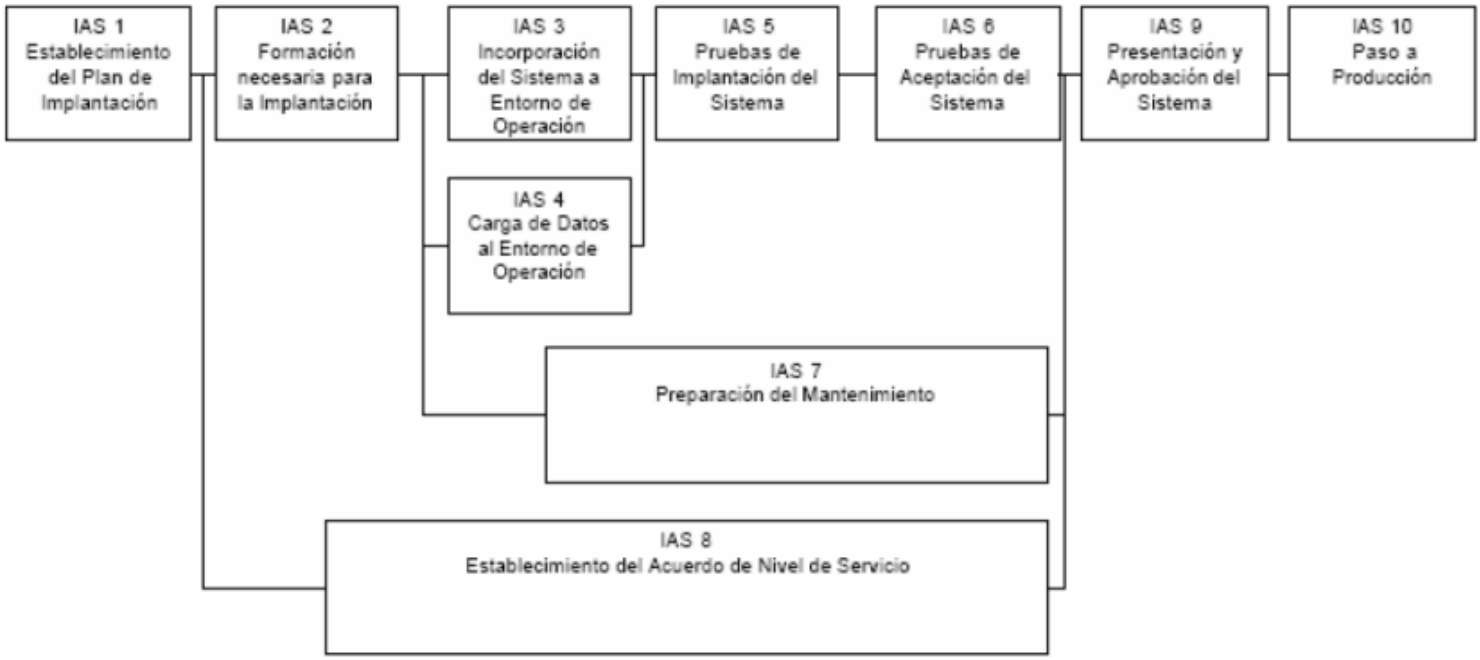
Asimismo, se llevan a cabo las tareas necesarias para la preparación del mantenimiento, siempre y cuando se haya decidido que éste va a efectuarse. En cualquier caso, es necesario que la persona que vaya a asumir el mantenimiento conozca el sistema, antes de su incorporación al entorno de producción.

Además hay que determinar los servicios (y niveles para cada uno) que requiere el sistema que se va a implantar, y el acuerdo que se adquiere una vez que se inicie la producción. Hay que distinguir entre servicios de gestión de operaciones (servicios por lotes, seguridad, comunicaciones, etc.) y servicios al cliente (servicio de atención a usuario, mantenimiento, etc.) que se deben negociar en cuanto a recursos, horarios, coste, etc. Se fija el nivel con el que se prestará el servicio como indicador de la calidad del mismo.

Conviene señalar que la implantación puede ser un proceso iterativo que se realiza de acuerdo al plan establecido para el comienzo de la producción del sistema en su entorno de operación. Para establecer este plan se tiene en cuenta:

- El cumplimiento de los requisitos de implantación definidos en la actividad Establecimiento de Requisitos (ASI 2) y especificados en la actividad Establecimiento de Requisitos de Implantación (DSI 11).
- La estrategia de transición del sistema antiguo al nuevo.

Finalmente, se realizan las acciones necesarias para el inicio de la producción. En el siguiente gráfico se muestra la relación de actividades de este proceso.



Actividad IAS 1: Establecimiento del Plan de Implantación

En esta actividad se revisa la estrategia de implantación para el sistema, establecida inicialmente en el proceso Estudio de Viabilidad del Sistema (EVS). Se identifican los distintos sistemas de información que forman parte del sistema objeto de la implantación. Para cada sistema se analizan las posibles dependencias con otros proyectos, que puedan condicionar el plan de implantación.

Una vez estudiado el alcance y los condicionantes de la implantación, se decide si ésta se puede llevar a cabo. Será preciso establecer, en su caso, la estrategia que se concretará de forma definitiva en el plan de implantación.

Se constituye el equipo de implantación, determinando los recursos humanos necesarios para la propia instalación del sistema, para las pruebas de implantación y aceptación, y

para la preparación del mantenimiento. Se identifican, para cada uno de ellos, sus perfiles y niveles de responsabilidad.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 1.1	Definición del Plan de Implantación	- Plan de implantación	- Sesiones de trabajo	- Jefe de Proyecto - Responsable de Implantación - Responsable de Operación - Responsable de Sistemas - Directores de los Usuarios
IAS 1.2	Especificación del Equipo de Implantación	- Equipo de implantación		- Jefe de Proyecto - Responsable de Implantación - Responsable de Operación - Responsable de Sistemas - Directores de los Usuarios

Actividad IAS 2: Formación Necesaria para la Implantación

En esta actividad se prepara y se imparte la formación al equipo que participará en la implantación y aceptación del sistema. Se realiza también el seguimiento de la formación de los usuarios finales, cuya impartición queda fuera del ámbito de MÉTRICA Versión 3. De esta forma, se asegura que la implantación se va a llevar a cabo correctamente.

Se determina la formación necesaria para el equipo de implantación, en función de los distintos perfiles y niveles de responsabilidad identificados en la actividad anterior. Para ello, se establece un plan de formación que incluye los esquemas de formación correspondientes, los recursos humanos y de infraestructura requeridos para llevarlo a cabo, así como una planificación que queda reflejada en el plan de formación.

La formación para que los usuarios finales sean capaces de utilizar el sistema de manera satisfactoria ha sido establecida, previamente, en la actividad Definición de la Formación de Usuarios Finales (CSI 7). En esta actividad, se analizan los esquemas de formación definidos según los diferentes perfiles, y se elabora un plan de formación que esté alineado con el plan de implantación.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 2.1	Preparación de la Formación del Equipo de Implantación	- Plan de Formación del Equipo de Implantación o Esquema de Formación o Materiales de Formación o Recursos de Formación o Planificación de la Formación		- Jefe de Proyecto - Responsable de Implantación - Directores de los Usuarios - Equipo de Formación
IAS 2.2	Formación del Equipo de Implantación	- Plan de Formación del Equipo de Implantación o Registro de Asistencia		- Equipo de Formación - Equipo de Implantación
IAS 2.3	Preparación de la Formación a Usuarios Finales	- Plan de Formación a Usuarios Finales o Esquema de Formación o Materiales de Formación o Planificación de la Formación		- Equipo de Operación - Jefe de Proyecto - Directores de los Usuarios
IAS 2.4	Seguimiento de la Formación a Usuarios Finales	- Plan de Formación a Usuarios Finales o Registro de Asistencia		- Jefe de Proyecto - Directores de los Usuarios

Actividad IAS 3: Incorporación del Sistema al Entorno de Operación

En esta actividad se realizan todas las tareas necesarias para la incorporación del sistema al entorno de operación en el que se van a llevar a cabo las pruebas de implantación y aceptación del sistema.

Mientras que las pruebas unitarias, de integración y del sistema se pueden ejecutar en un entorno distinto de aquel en el que finalmente se implantará, las pruebas de implantación y aceptación del sistema deben ejecutarse en el entorno real de operación. El propósito es comprobar que el sistema satisface todos los requisitos especificados por el usuario en las mismas condiciones que cuando se inicie la producción.

Por tanto, como paso previo a la realización de dichas pruebas y de acuerdo al plan de implantación establecido, se verifica que los recursos necesarios están disponibles para que se pueda realizar, adecuadamente, la instalación de todos los componentes que integran el sistema, así como la creación y puesta a punto de las bases de datos en el entorno de operación. Asimismo, se establecen los procedimientos de explotación y uso de las bases de datos de acuerdo a la normativa existente en dicho entorno.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 3.1	Preparación de la Instalación	- Incidencias de Preparación de Instalación		- Jefe de Proyecto - Responsable de Implantación - Equipo de Implantación
IAS 3.2	Realización de la Instalación	- Producto Software (instalado). - Código Fuente de los Componentes de Migración y Carga Inicial de Datos (instalado) - Procedimientos de Migración y Carga Inicial de Datos (instalado)		- Jefe de Proyecto - Equipo de Implantación

Actividad IAS 4: Carga de Datos al Entorno de Operación

Teniendo en cuenta que los sistemas de información que forman parte del sistema a implantar pueden mejorar, ampliar o sustituir a otros ya existentes en la organización, puede ser necesaria una carga inicial y/o una migración de datos cuyo alcance dependerá de las características y cobertura de cada sistema de información implicado. Por tanto, la necesidad de una migración de datos puede venir determinada desde el proceso Estudio de Viabilidad del Sistema (EVS), en la actividad Selección de la Solución (EVS 6). Allí se habrá establecido la estrategia a seguir en la sustitución, evaluando las opciones del enfoque de desarrollo e instalación más apropiados para llevarlo a cabo.

En cualquier caso, en la actividad Diseño de la Migración y Carga Inicial de Datos (DSI 9) se habrán definido y planificado los procesos y procedimientos necesarios para llevar a cabo la migración, realizándose su codificación en la actividad Construcción de los Componentes y Procedimientos de Migración y Carga Inicial de Datos (CSI 8).

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 4.1	Migración y Carga Inicial de Datos	- Bases de datos / ficheros cargados		- Equipo de Implantación - Equipo de Operación - Administradores de Bases de Datos - Usuarios Expertos

Actividad IAS 5: Pruebas de Implantación del Sistema

La finalidad de las pruebas de implantación es doble:

- Comprobar el funcionamiento correcto del mismo en el entorno de operación.
- Permitir que el usuario determine, desde el punto de vista de operación, la aceptación del sistema instalado en su entorno real, según el cumplimiento de los requisitos especificados.

Para ello, el responsable de implantación revisa el plan de pruebas de implantación y los criterios de aceptación del sistema, previamente elaborados. Las pruebas las realizan los técnicos de sistemas y de operación, que forman parte del grupo de usuarios técnicos que ha recibido la formación necesaria para llevarlas a cabo.

Una vez ejecutadas estas pruebas, el equipo de usuario técnicos informa de las incidencias detectadas al responsable de implantación, el cual analiza la información y toma las medidas correctoras que considere necesarias para que el sistema dé respuesta a las especificaciones previstas, momento en el que el equipo de operación lo da por probado.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 5.1	Preparación de las Pruebas de Implantación	- Plan de pruebas		- Jefe de Proyecto - Responsable de Implantación
IAS 5.2	Realización de las Pruebas de Implantación	- Resultado de las pruebas de implantación	- Pruebas de implantación	- Equipo de Implantación
IAS 5.3	Evaluación del resultado de las Pruebas de Implantación	- Evaluación del resultado de las pruebas de implantación		- Jefe de Proyecto - Responsable de Implantación

Actividad IAS 6: Pruebas de Aceptación del Sistema

Las pruebas de aceptación tienen como fin validar que el sistema cumple los requisitos básicos de funcionamiento esperado y permitir que el usuario determine la aceptación del sistema. Por este motivo, estas pruebas son realizadas por el usuario final que, durante este periodo de tiempo, debe plantear todas las deficiencias o errores que encuentre antes de dar por aprobado el sistema definitivamente.

Los Directores de los Usuarios revisan los criterios de aceptación, especificados previamente en el plan de pruebas del sistema, y dirigen las pruebas de aceptación final que llevan a cabo los usuarios expertos. A su vez, éstos últimos deben elaborar un informe que los Directores de los Usuarios analizan y evalúan para determinar la aceptación o rechazo del sistema.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 6.1	Preparación de las Pruebas de Aceptación	- Plan de Pruebas		- Jefe de Proyecto - Directores de los Usuarios
IAS 6.2	Realización de las Pruebas de Aceptación	- Resultado de las Pruebas de Aceptación	- Pruebas de Aceptación	- Usuarios Expertos
IAS 6.3	Evaluación del resultado de las Pruebas de Aceptación	- Evaluación del Resultado de las Pruebas de Aceptación		- Jefe de Proyecto - Directores de los Usuarios

Actividad IAS 7: Preparación del Mantenimiento del Sistema

El objetivo de esta actividad es permitir que el equipo que va a asumir el mantenimiento del sistema esté familiarizado con él antes de que el sistema pase a producción. Para conseguir este objetivo, se ha considerado al responsable de mantenimiento como parte integrante del equipo de implantación. Por lo tanto, se habrá tenido en cuenta su perfil al elaborar el esquema de formación correspondiente.

Una vez que el responsable de mantenimiento ha recibido la formación necesaria y adquirido una visión global del sistema que se va a implantar, se le entregan los productos que serán objeto del mantenimiento. De esta manera, obtiene de una forma gradual un conocimiento profundo del funcionamiento y facilidades que incorpora el sistema, que van a permitirle acometer los cambios solicitados por los usuarios con mayor facilidad y eficiencia. Se reduce, en consecuencia, el esfuerzo invertido en el mantenimiento.

Es importante resaltar que la existencia de una configuración del software permite reducir el esfuerzo requerido y mejora la calidad general del software a mantener, aunque no garantiza un mantenimiento libre de problemas. Una pobre configuración del software puede tener un impacto negativo sobre su facilidad de mantenimiento.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 7.1	Establecimiento de la Infraestructura para el Mantenimiento	- Plan de Mantenimiento	- Diagrama de Representación - Sesiones de trabajo	- Jefe de Proyecto - Responsable de Mantenimiento - Equipo de Soporte Técnico
IAS 7.2	Formalización del Plan de Mantenimiento	- Plan de Mantenimiento	- Sesiones de trabajo	- Responsable de Mantenimiento - Directores de los Usuarios

Actividad IAS 8: Establecimiento del Acuerdo de Nivel de Servicio

Antes de la aprobación definitiva del sistema por parte del Comité de Dirección es conveniente:

- Determinar los servicios que requiere el mismo.
- Especificar los niveles de servicio con los que se va a valorar la calidad de esa prestación.
- Definir qué compromisos se adquieren con la entrega del sistema.

Para ello, en primer lugar, se negocia entre los máximos responsables del usuario y de operación qué servicios y de qué tipo se van a prestar. Una vez acordados, se detallan los niveles de servicio definiendo sus propiedades funcionales y de calidad. Se establece cuáles de ellas son cuantificables y qué indicadores se van a aplicar. Es importante señalar que los niveles de servicio son específicos para cada uno de los subsistemas que componen el sistema de información, y dependen del entorno de operación y de la localización geográfica en que se implante un sistema de información concreto, pudiendo haber servicios básicos para todo el sistema o específicos para un subsistema de información concreto.

Por último, se establece formalmente el acuerdo de nivel de servicio, considerando los recursos necesarios, plazos de restablecimiento del servicio, coste y mecanismos de regulación que están asociados a cada servicio especificado anteriormente.

Según el ámbito y el alcance de los tipos de servicio que se vayan a prestar, se determinan los productos del ciclo de vida del software necesarios para poder establecer el acuerdo de nivel de servicio.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 8.1	Identificación de los Servicios	- Especificación de Tipos de Servicio	- Sesiones de trabajo	- Jefe de Proyecto - Directores de los Usuarios - Comité de Dirección
IAS 8.2	Descripción de las Propiedades de cada Servicio	- Especificación de Tipos de Servicio: o Niveles de Servicio	- Sesiones de trabajo	- Jefe de Proyecto - Directores de los Usuarios - Comité de Dirección - Comité de Seguimiento
IAS 8.3	Determinación del Acuerdo de Nivel de Servicio	- Acuerdo de Nivel de Servicio	- Sesiones de trabajo	- Comité de Dirección - Jefe de Proyecto - Responsable de Implantación

Actividad IAS 9: Presentación y Aprobación del Sistema

Una vez que se han efectuado las pruebas de implantación y de aceptación, y que se ha fijado el acuerdo de nivel de servicio, el Comité de Dirección debe formalizar la aprobación del sistema. Para esto, se lleva a cabo una presentación general del sistema al Comité de Dirección y se espera la confirmación de su aprobación.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 9.1	Convocatoria de la Presentación del Sistema	- Plan de presentación del sistema	- Presentación	- Jefe de Proyecto - Responsable de Implantación - Directores de los Usuarios
IAS 9.2	Aprobación del Sistema	- Aprobación del sistema	- Presentación	- Comité de Dirección - Jefe de Proyecto - Directores de los Usuarios - Responsable de Implantación

Actividad IAS 10: Paso a Producción

Esta actividad tiene como objetivo establecer el punto de inicio en que el sistema pasa a producción, se traspasa la responsabilidad al equipo de mantenimiento y se empiezan a dar los servicios establecidos en el acuerdo de nivel de servicio, una vez que el Comité de Dirección ha aprobado el sistema.

Para ello es necesario que, después de haber realizado las pruebas de implantación y de aceptación del sistema, se disponga del entorno de producción perfectamente instalado en cuanto a hardware y software de base, componentes del nuevo sistema y procedimientos manuales y automáticos.

En función del entorno en el que se hayan llevado a cabo las pruebas de implantación y aceptación del sistema, habrá que instalar los componentes del sistema total o parcialmente. También se tendrá en cuenta la necesidad de migrar todos los datos o una parte de ellos.

Una vez que el sistema ya está en producción, se le notifica al responsable de mantenimiento, al responsable de operación y al Comité de Dirección.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
IAS 10.1	Preparación del Entorno de Producción	- Incidencias del Paso a Producción		- Jefe de Proyecto - Responsable de Implantación - Equipo de Implantación - Equipo de Soporte Técnico
IAS 10.2	Activación del Sistema en Producción	- Sistema en Producción		- Comité de Dirección - Responsable de Implantación

Mantenimiento de Sistemas de Información

Descripción y Objetivos

El objetivo de este proceso es la obtención de una nueva versión de un sistema de información desarrollado con MÉTRICA Versión 3 o Versión 2, a partir de las peticiones de mantenimiento que los usuarios realizan con motivo de un problema detectado en el sistema, o por la necesidad de una mejora del mismo.

En este apartado se realiza el registro de las peticiones de mantenimiento recibidas, con el fin de llevar el control de las mismas y de proporcionar, si fuera necesario, datos estadísticos de peticiones recibidas o atendidas en un determinado periodo, sistemas que se han visto afectados por los cambios, en qué medida y el tiempo empleado en la resolución de dichos cambios. Es recomendable, por lo tanto, llevar un catálogo de peticiones de mantenimiento sobre los sistemas de información, en el que se registren una serie de datos que nos permitan disponer de la información antes mencionada.

En el momento en el que se registra la petición, se procede a diagnosticar de qué tipo de mantenimiento se trata. Atendiendo a los fines, podemos establecer los siguientes tipos de mantenimiento:

- **Correctivo**: son aquellos cambios precisos para corregir errores del producto software.
- **Evolutivo**: son las incorporaciones, modificaciones y eliminaciones necesarias en un producto software para cubrir la expansión o cambio en las necesidades del usuario.
- **Adaptativo**: son las modificaciones que afectan a los entornos en los que el sistema opera, por ejemplo, cambios de configuración del hardware, software de base, gestores de base de datos, comunicaciones, etc.
- **Perfectivo**: son las acciones llevadas a cabo para mejorar la calidad interna de los sistemas en cualquiera de sus aspectos: reestructuración del código, definición más clara del sistema y optimización del rendimiento y eficiencia.

Estos dos últimos tipos quedan fuera del ámbito de MÉTRICA Versión 3 ya que requieren actividades y perfiles distintos de los del proceso de desarrollo.

Una vez registrada la petición e identificado el tipo de mantenimiento y su origen, se determina de quién es la responsabilidad de atender la petición. En el supuesto de que la petición sea remitida, se registra en el catálogo de peticiones de mantenimiento y continúa el proceso. La petición puede ser denegada. En este caso, se notifica al usuario y acaba el proceso.

Posteriormente, según se trate de un mantenimiento correctivo o evolutivo, se verifica y reproduce el problema, o se estudia la viabilidad del cambio propuesto por el usuario. En ambos casos se estudia el alcance de la modificación. Hay que analizar las alternativas de solución identificando, según el tipo de mantenimiento de que se trate, cuál es la más

adecuada. El plazo y urgencia de la solución a la petición se establece de acuerdo con el estudio anterior.

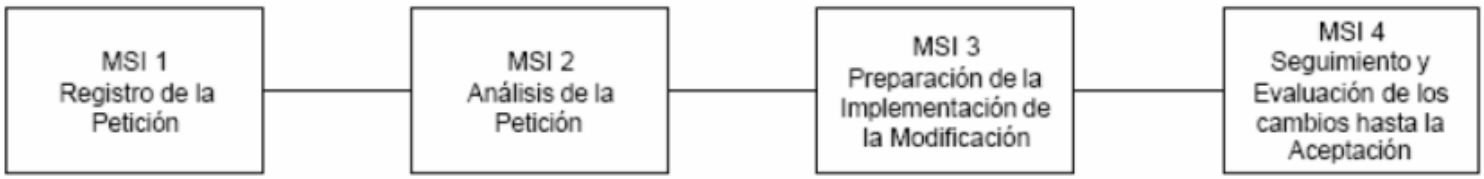
La definición de la solución incluye el estudio del impacto de la solución propuesta para la petición en los sistemas de información afectados. Mediante el análisis de dicho estudio, la persona encargada del Proceso de Mantenimiento valora el esfuerzo y coste necesario para la implementación de la modificación.

Las tareas de los procesos de desarrollo que va a ser necesario realizar son determinadas en función de los componentes del sistema actual afectados por la modificación. Estas tareas pertenecen a actividades de los procesos Análisis, Diseño, Construcción e Implantación.

Por último, y antes de la aceptación del usuario, es preciso establecer un plan de pruebas de regresión que asegure la integridad del sistema de información afectado.

La mejor forma de mantener el coste de mantenimiento bajo control es una gestión del proceso de Mantenimiento efectiva y comprometida. Por lo tanto, es necesario registrar de forma disciplinada los cambios realizados en los sistemas de información y en su documentación. Esto repercutirá directamente en la mayor calidad de los sistemas actuales.

La estructura propuesta para el Proceso de Mantenimiento de MÉTRICA Versión 3 comprende las siguientes actividades:



Actividad MSI 1: Registro de la Petición

El objetivo de esta actividad es establecer un sistema estandarizado de registro de información para las peticiones de mantenimiento, con el fin de controlar y canalizar los cambios propuestos por un usuario o cliente, mejorando el flujo de trabajo de la organización y proporcionando una gestión efectiva del mantenimiento. Es importante asignar responsabilidades para evitar la realización de cambios que beneficien a un usuario, pero que produzca un impacto negativo sobre otros muchos. Por tanto, es necesario, que todas las peticiones de mantenimiento sean presentadas de una forma estandarizada, que permita su clasificación y facilite la identificación del tipo de mantenimiento requerido.

Una vez que la petición ha sido registrada, que ha determinado el tipo de mantenimiento y los sistemas de información a los que inicialmente puede afectar, se comprueba su viabilidad, de acuerdo a las prestaciones de mantenimiento establecidas para dichos sistemas de información.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
MSI 1.1	- Registro de la Petición	- Catálogo de Peticiones	- Catalogación	- Responsable de Mantenimiento
MSI 1.2	- Asignación de la Petición	- Catálogo de Peticiones: o Aceptación / Rechazo de la Petición o Asignación de Responsable	- Catalogación	- Responsable de Mantenimiento

Actividad MSI 2: Análisis de la Petición

En esta actividad se lleva a cabo el diagnóstico y análisis del cambio para dar respuesta a las peticiones de mantenimiento que han sido aceptadas en la actividad anterior.

Se analiza el alcance de la petición en lo referente a los sistemas de información afectados, valorando hasta qué punto pueden ser modificados en función del ciclo de vida estimado para los mismos y determinando la necesidad de desviar la petició hacia el proceso Estudio de Viabilidad del Sistema (EVS) o Análisis del Sistema de Información (ASI), en función del impacto sobre los sistemas de información afectados.

El enfoque de este estudio varía según el tipo de mantenimiento, teniendo en cuenta que en el caso de un mantenimiento correctivo que implique un error crítico debe abordarse el cambio de forma inmediata sin profundizar en el origen del mismo. No obstante, una vez reanudado el servicio, es imprescindible analizar el problema y determinar cuál es la solución definitiva.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
MSI 2.1	Verificación y Estudio de la Petición	- Catálogo de Peticiones: - Verificación de la Petición ○ Resultado del Estudio de la Petición	- Sesiones de trabajo - Catalogación	- Equipo de Mantenimiento

MSI 2.2	Estudio de la Propuesta de Solución	- Propuesta de Solución
		- Catálogo de Peticiones:
		  ○ Estudio del Impacto
		  ○ Aceptación / Rechazo de la solución
			- Sesiones de trabajo
			- Catalogación
				- Responsable de Mantenimiento
				- Equipo de Mantenimiento

Actividad MSI 3: Preparación de la Implementación de la Modificación

Una vez finalizado el estudio previo de la petición y aprobada su implementación, se pasa a identificar de forma detallada cada uno de los elementos afectados por el cambio mediante el análisis de impacto. Este análisis tiene como objetivo determinar qué parte del sistema de información se ve afectada, y en qué medida, dejando claramente definido y documentado qué componentes hay que modificar, tanto de software como de hardware. Con el resultado de este análisis se dispone de los datos cuantitativos sobre los que aplicar los indicadores establecidos. Esto permitirá fijar un plan de acción, valorando la necesidad de realizar un reajuste de dichos indicadores, con el fin de cumplir el plazo máximo de entrega.

Una vez aceptado el plan de acción, se activan los correspondientes procesos de desarrollo para llevar a cabo la implementación de la solución. Al mismo tiempo, se especifican las pruebas de regresión con el fin de evitar el efecto onda en el sistema, una vez realizados los cambios.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
MSI 3.1	- Identificación de Elementos Afectados	- Catálogo de Peticiones: ○ Elementos Afectados - Análisis de Impacto de los Cambios	- Catalogación - Análisis de Impacto	- Equipo de Mantenimiento - Jefe de Proyecto
MSI 3.2	- Establecimiento del Plan de Acción	- Catálogo de Peticiones: ○ Actividades y Tareas de los Procesos de Desarrollo a Realizar - Plan de Acción para la Modificación	- Planificación - Catalogación	- Responsable de Mantenimiento - Equipo de Mantenimiento - Jefe de Proyecto
MSI 3.3	- Especificación del Plan de Pruebas de Regresión	- Plan de Pruebas de Regresión		- Equipo de Mantenimiento - Jefe de Proyecto

Actividad MSI 4: Seguimiento y Evaluación de los Cambios hasta la Aceptación

Se realiza el seguimiento de los cambios que se están llevando a cabo en los procesos de desarrollo, de acuerdo a los puntos de control del ciclo de vida del cambio establecidos en el plan de acción. Durante este seguimiento, se comprueba que sólo se han modificado los elementos que se ven afectados por el cambio y que se han realizado las pruebas correspondientes, especialmente las pruebas de integración y del sistema. Del resultado obtenido se hace una evaluación del cambio para la posterior aprobación.

Una vez finalizado el cambio en desarrollo, se realizan las pruebas de regresión especificadas en la actividad anterior, comprobando que ningún sistema no modificado, pero con posibilidades de verse afectado, ha variado su comportamiento habitual. Se informa si ha habido incidencias con el fin de que se resuelvan del modo más conveniente. Se evalúan las pruebas.

La aprobación de la petición se realiza al finalizar las pruebas de regresión, y después de comprobar que todo lo que ha sido modificado o puede verse afectado por el cambio, funciona correctamente. Con el cierre de la petición se podrán incluir en el catálogo, si se considera oportuno, parte de la información obtenida durante el proceso de mantenimiento que pueda facilitar posteriores análisis.

A continuación se incluye una tabla resumen con las tareas de la presente actividad:

Tarea		Productos	Técnicas y Prácticas	Participantes
MSI 4.1	Seguimiento de los Cambios	- Evaluación del Cambio		- Equipo de Mantenimiento - Responsable de Mantenimiento - Jefe de Proyecto
MSI 4.2	Realización de las Pruebas de Regresión	- Resultado de las Pruebas de Regresión - Evaluación del Resultado de las Pruebas de Regresión	- Pruebas de Regresión	- Responsable de Mantenimiento - Equipo de Mantenimiento - Jefe de Proyecto
MSI 4.3	Aprobación y Cierre de la Petición	- Catálogo de Peticiones: ○ Nueva Versión y Aprobación	- Catalogación	- Directores de los Usuarios - Responsable de Mantenimiento

Bibliografia

- Scribd (Ibiza Ales)

Estrategias de determinación de requerimientos: Entrevistas, Derivación de sistemas existentes, Análisis y Prototipos. La especificación de requisitos software.

Introducción a la Ingeniería de Requerimientos

En la actualidad, son muchos los procesos de desarrollo de software que existen. Con el paso de los años, la Ingeniería del Software ha introducido y popularizado una serie de estándares para medir y certificar la calidad, tanto del sistema a desarrollar, como del proceso de desarrollo en sí. Se han publicado muchos libros y artículos relacionados con este tema, con el modelado de procesos del negocio y la reingeniería. Un número creciente de herramientas automatizadas han surgido para ayudar a definir y aplicar un proceso de desarrollo de software efectivo.

Hoy en día la economía global depende más de sistemas automatizados que en épocas pasadas. Esto ha llevado a los equipos de desarrollo a enfrentarse con una nueva década de procesos y estándares de calidad.

Sin embargo, ¿cómo explicamos la alta incidencia de fallos en los proyectos de software? ¿Por qué existen tantos proyectos de software víctimas de retrasos, presupuestos mal dimensionados y con problemas de calidad? ¿Cómo podemos tener una producción o una economía de calidad, cuando nuestras actividades diarias dependen de la calidad de un sistema que no la tiene?

Tal vez suene ilógico pero, a pesar de los avances que ha dado la tecnología, aún existen procesos de producción informales, parciales y en algunos casos no confiables.

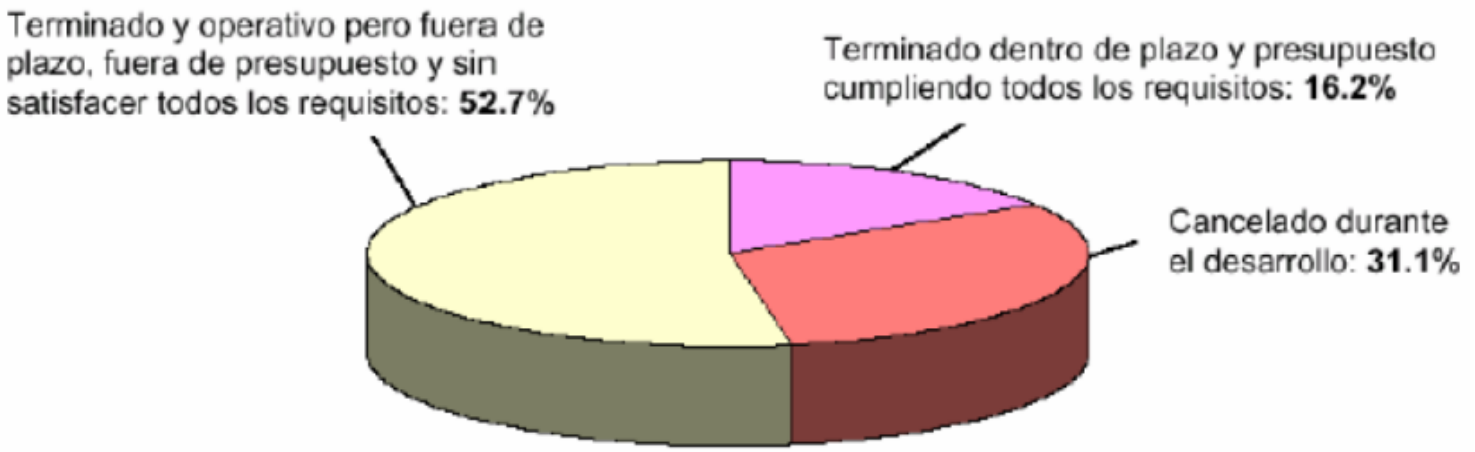
La Ingeniería de Requerimientos cumple un papel primordial en el proceso de producción de software, ya que enfoca un área fundamental: la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema. De esta manera, se pretende minimizar los problemas relacionados al desarrollo de sistemas.

Existe una gran cantidad de proyectos de software que no llegan a cumplir sus objetivos. En nuestro país somos partícipes de este problema a diario, en donde se ha vuelto común la compra de sistemas extranjeros, para luego “personalizarlos” supuestamente a la medida de las empresas.

Tal “personalización” termina retrasando (la mayoría de las veces) el proyecto en meses, o incluso en años. La problemática del año 2000 trajo como consecuencia una serie de cambios apresurados en los sistemas existentes, cambios que no fueron bien planificados.

El reemplazo de plataformas y tecnologías obsoletas, la compra de sistemas completamente nuevos, las modificaciones de todos o de casi todos los programas que forman un sistema, entre otras razones, llevan a desarrollar proyectos en calendarios sumamente ajustados y en algunos casos irreales. Esto ocasiona que se omitan muchos pasos importantes en el ciclo de vida de desarrollo, entre estos, la definición de los requerimientos.

En 1995 se realizó un estudio (informe CHAOS) sobre el resultado general de los proyectos de software. El estudio fue demoledor y esto a pesar de las herramientas existentes para el desarrollo de software (ver figura).



La Ingeniería de Requerimientos

¿Qué son Requerimientos?

Normalmente, un concepto de la Ingeniería de Software tiene diferentes significados. De las muchas definiciones que existen para requerimiento, a continuación se presenta la definición que aparece en el glosario de la IEEE:

1. Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo.
2. Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.
3. Una representación documentada de una condición o capacidad como en 1 o 2.

Los requerimientos pueden dividirse en requerimientos funcionales y requerimientos no funcionales:

- Los requerimientos funcionales definen las funciones que el sistema será capaz de realizar. Describen las transformaciones que el sistema realiza sobre las entradas para producir salidas.
- Los requerimientos no funcionales tienen que ver con características que de una u otra forma puedan limitar el sistema, como por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares, etc.

Características de los Requerimientos

Las características de un requerimiento son sus propiedades principales. Un conjunto de requerimientos en estado de madurez deben presentar una serie de características tanto individualmente como en grupo. A continuación se presentan las más importantes:

- **Necesario**: Un requerimiento es necesario si su omisión provoca una deficiencia en el sistema a construir, y además su capacidad, características físicas o factor de calidad no pueden ser reemplazados por otras capacidades del producto o del proceso.
- **Conciso**: Un requerimiento es conciso si es fácil de leer y entender. Su redacción debe ser simple y clara para aquellos que vayan a consultarlo en un futuro.

- **Completo**: Un requerimiento está completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.
- **Consistente**: Un requerimiento es consistente si no es contradictorio con otro requerimiento.
- **No ambiguo**: Un requerimiento no es ambiguo cuando tiene una sola interpretación. El lenguaje usado en su definición, no debe causar confusiones al lector.
- **Verificable**: Un requerimiento es verificable cuando puede ser cuantificado de manera que permita hacer uso de los siguientes métodos de verificación: inspección, análisis, demostración o pruebas.

Dificultades para definir los Requerimientos

- Los requerimientos no son obvios y vienen de muchas fuentes.
- Son difíciles de expresar en palabras (el lenguaje es ambiguo).
- Existen muchos tipos de requerimientos y diferentes niveles de detalle.
- La cantidad de requerimientos en un proyecto puede ser difícil de manejar.
- Nunca son iguales. Algunos son más difíciles, más arriesgados, más importantes o más estables que otros.
- Los requerimientos están relacionados unos con otros, y a su vez se relacionan con otras partes del proceso.
- Cada requerimiento tiene propiedades únicas y abarcan áreas funcionales específicas.
- Un requerimiento puede cambiar a lo largo del ciclo de desarrollo.
- Son difíciles de cuantificar, ya que cada conjunto de requerimientos es particular para cada proyecto.

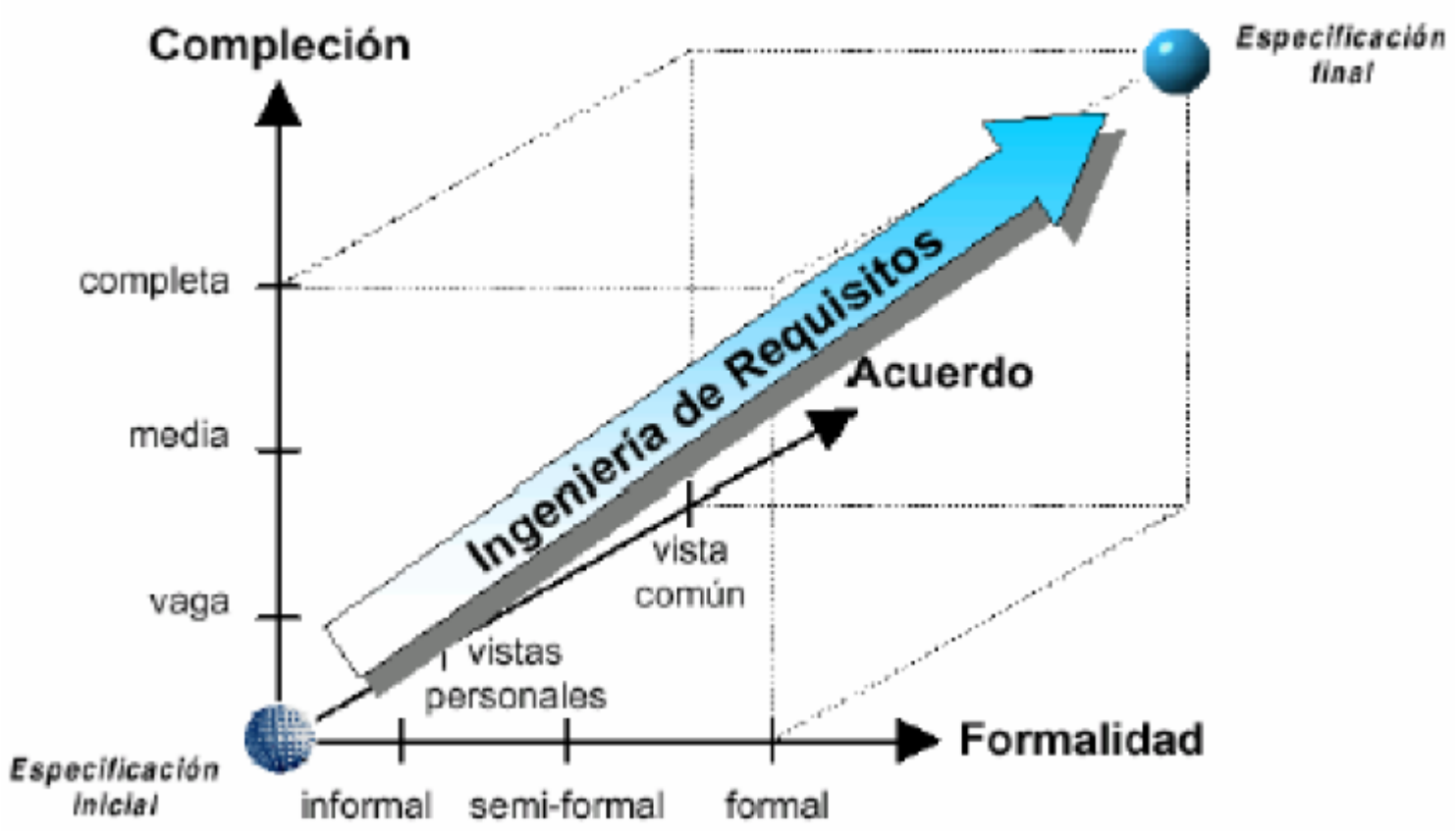
Definiciones par la Ingeniería de Requerimientos

- *Ingeniería de Requerimientos vs Administración de Requerimientos*: El proceso de recopilar, analizar y verificar las necesidades del cliente para un sistema es llamado Ingeniería de Requerimientos (IR). La meta de la IR es entregar una especificación de requerimientos de software correcta y completa. Los requerimientos son solicitudes que hace el usuario hacia el equipo de trabajo para la realización de alguna tarea. Los requerimientos que son aceptados, son definidos entre el cliente, los usuarios y el equipo de desarrollo, para identificar claramente los límites del sistema. Los requerimientos de los usuarios que han sido definidos, serán administrados para atenderlos y darles un adecuado seguimiento. La administración de requerimientos permite adicionalmente tener un control económico de los mismos.
    - La determinación tiene que ver con: Alcance, requerimientos funcionales, requerimientos no funcionales, complejidad.
    - La administración tiene que ver con: Actividades a realizar, responsables, productos a entregar, costo, tiempos.
- Ingeniería de Requerimientos es la disciplina para desarrollar una especificación completa, consistente y no ambigua, la cual servirá como base para acuerdos comunes entre todas las partes involucradas y en dónde se describen las funciones que realizará el sistema.
- Ingeniería de Requerimientos es el proceso por el cual se transforman los requerimientos declarados por los clientes, ya sean hablados o escritos, a especificaciones precisas, no ambigua, consistentes y completas del comportamiento del sistema, incluyendo funciones, interfaces, rendimiento y limitaciones.
- Es el proceso mediante el cual se intercambian diferentes puntos de vista para recopilar y modelar lo que el sistema va a realizar. Este proceso utiliza una combinación de métodos, herramientas y actores, cuyo producto es un modelo del cual se genera un documento de requerimientos.
- Ingeniería de Requerimientos es un enfoque sistémico para recolectar, organizar y documentar los requerimientos del sistema. Es también el proceso que establece y

mantiene acuerdos sobre los cambios de requerimientos, entre los clientes y el equipo del proyecto.
- Una posible visión de la ingeniería de requerimientos es considerarla como un proceso de construcción de una especificación de requerimientos en el que se avanza desde especificaciones iniciales, que no poseen las propiedades idóneas, hasta especificaciones finales completas, formales y acordadas entre todos los participantes.

Tres enfoques (dimensiones) desde los que se debe abordar la IR para lograr las especificaciones finales de los requerimientos de un sistema (ver figura):

- Por un lado están los factores psicológicos y congnitivos que afectan al grado de conocimiento sobre el sistema que se desea desarrollar, es decir, llegar a conocer la totalidad de los requerimientos que debe satisfacer el sistema. Habitualmente, este enfoque se cubre con actividades encaminadas a la adquisición de los requerimientos (elicitación).
- Por otro lado, está el grado de formalismo de la representación del conocimiento sobre dichos requerimientos, teniendo en cuenta que un mayor grado de formalismo no implica necesariamente un mayor conocimiento. El formalismo se logra en las actividades encaminadas al análisis de requerimientos.
- Por último, están los aspectos sociales, ya que al ser un proceso en el que participan personas con diferentes puntos de vista, es necesario llegar a un punto de acuerdo, normalmente mediante algún tipo de negociación. Las actividades de validación permiten a la IR resolver los posibles conflictos.



Importancia de la Ingeniería de Requerimientos

Los principales beneficios que se obtienen de la Ingeniería de Requerimientos son:

- Permite gestionar las necesidades del proyecto en forma estructurada: Cada actividad de la IR consiste de una serie de pasos organizados y bien definidos.
- Mejora la capacidad de predecir cronogramas de proyectos, así como sus resultados: La IR proporciona un punto de partida para controles subsecuentes y actividades de mantenimiento, tales como estimación de costos, tiempo y recursos necesarios.

- Disminuye los costos y retrasos del proyecto: Muchos estudios han demostrado que reparar errores por un mal desarrollo no descubierto a tiempo, es sumamente caro, especialmente aquellas decisiones tomadas durante la IR.
- Mejora la calidad del software: La calidad en el software tiene que ver con cumplir un conjunto de requerimientos (funcionalidad, facilidad de uso, confiabilidad, desempeño, etc.).
- Mejora la comunicación entre equipos: La especificación de requerimientos representa una forma de consenso entre clientes y desarrolladores. Si este consenso no ocurre, el proyecto no será exitoso.
- Evita rechazos de usuarios finales: La ingeniería de requerimientos obliga al cliente a considerar sus requerimientos cuidadosamente y revisarlos dentro del marco del problema, por lo que se le involucra durante todo el desarrollo del proyecto.

Personal Involucrado

Realmente, son muchas las personas involucradas en el desarrollo de los requerimientos de un sistema. Es importante saber que cada una de esas personas tienen diversos intereses y juegan roles específicos dentro de la planificación del proyecto. El conocimiento de cada papel desempeñado asegura que se involucren a las personas correctas en las diferentes fases del ciclo de vida, y en las diferentes actividades de la IR.

No conocer estos intereses puede ocasionar una comunicación poco efectiva entre clientes y desarrolladores, que a la vez traería impactos negativos tanto en tiempo como en presupuesto. Los roles más importantes pueden clasificarse como sigue:

- *Usuarios finales*: Son las personas que usarán el sistema desarrollado. Ellos están relacionados con la usabilidad, la disponibilidad y la fiabilidad del sistema. Están familiarizados con los procesos específicos que debe realizar el software, dentro de los parámetros de su ambiente laboral. Serán quienes utilicen las interfaces y los manuales de usuario.
- *Usuarios líderes*: Son los individuos que comprenden el ambiente del sistema o el dominio del problema en donde será empleado el software desarrollado. Ellos proporcionan al equipo técnico los detalles y requerimientos de las interfaces del sistema.
- *Personal de mantenimiento*: Para proyectos que requieran un mantenimiento eventual, estas personas son las responsables de la administración de cambios, de la implementación y resolución de anomalías. Su trabajo consiste en revisar y mejorar los procesos del producto ya finalizado.
- *Analistas y programadores*: Son los responsables del desarrollo del producto en sí. Ellos interactúan directamente con el cliente.
- *Personal de pruebas*: Se encargan de elaborar y ejecutar el plan de pruebas para asegurar que las condiciones presentadas por el sistema son las adecuadas. Son quienes van a validar si los requerimientos satisfacen las necesidades del cliente.

Otras personas que pueden estar involucradas, dependiendo de la magnitud del proyecto, pueden ser: administradores de proyecto, documentadores, diseñadores de BD, entre otros.

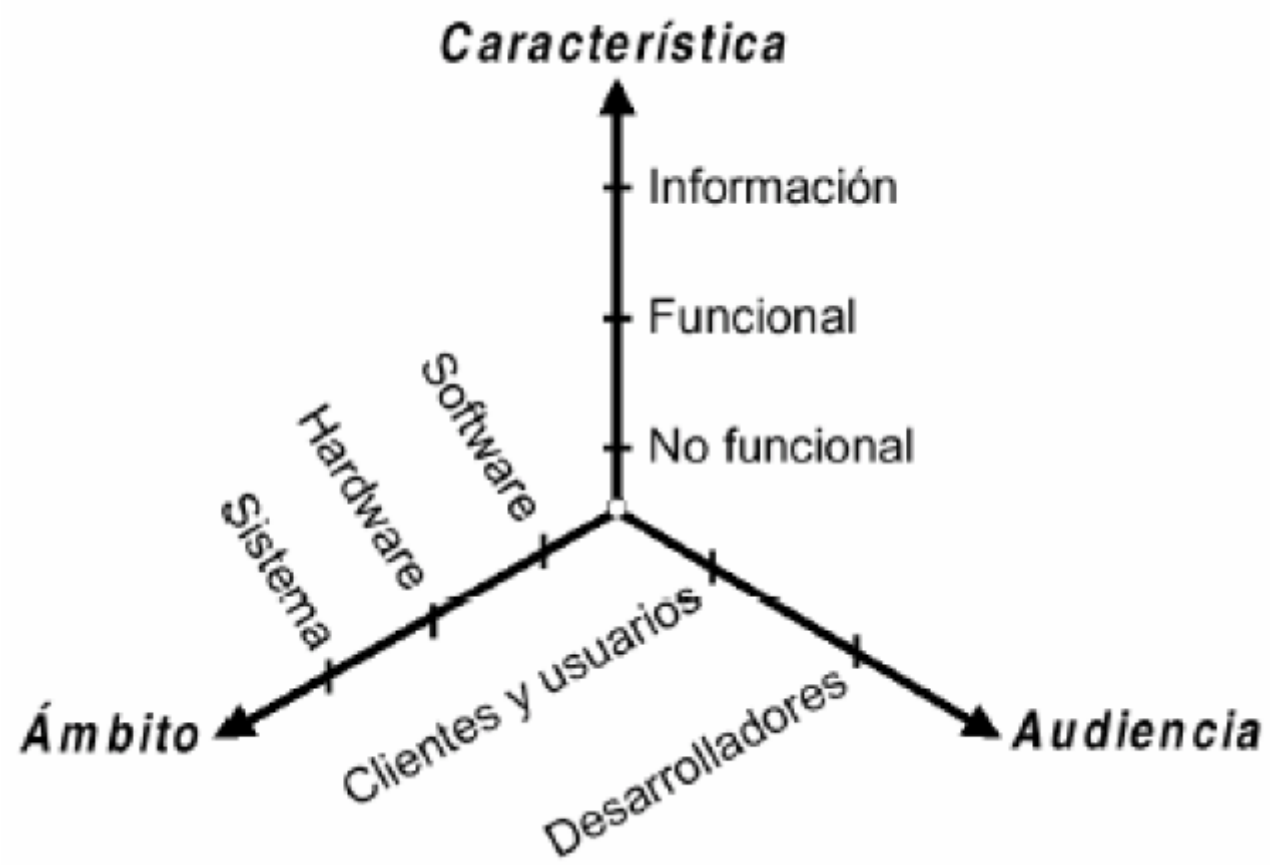
Las Dimensiones de los Requerimientos

Los calificativos que se aplican al término requerimiento muestran distintos ortogonales que a menudo son considerados aisladamente. Aquí se ven agrupados en tres dimensiones (ver figura).

- **Ámbito**: Esta dimensión indica en qué ámbito se debe entender el requerimiento. En general, un ámbito de sistema indica que el requerimiento debe cumplirse a nivel de sistema, entendiendo por sistema un conjunto de hardware y software.

- **Característica que define**: Esta dimensión clasifica los requerimientos en función de la naturaleza de la característica del sistema que se especifica. En [Pohl 1997] aparece una completa clasificación denominada RSM (Requirements Specification Model, Modelo de Especificación de Requerimientos), cuyas principales clases son: requerimientos funcionales, requerimientos de datos y requerimientos no funcionales. Es conveniente separar de los requerimientos funcionales a los requerimientos de datos o de almacenamiento de información, que establecen qué información debe almacenar el sistema por ser relevante para las necesidades y objetivos de clientes y usuarios.
- **Audiencia**: Esta dimensión indica la audiencia a la que está dirigido el requisito, es decir, las personas que deben ser capaces de entenderlo. En general, se pueden distinguir dos tipos de audiencia:
  - Los clientes y usuarios, que no tienen porqué tener formación en ingeniería del software (especificación de requerimientos mediante lenguaje natural).
  - Los desarrolladores de software (especificación de requerimientos utilizando técnicas estructuradas, orientadas a objetos o formales).

En la literatura se suele referir a los requerimientos orientados a clientes y usuarios como requerimientos-C, requisito de usuario o requisito de cliente; y, a los requerimientos orientados al desarrollador como requerimientos-D o requerimientos software.



Puntos a considerar durante la Ingeniería de Requerimientos

- Objetivos del negocio y ambiente de trabajo: Aunque los objetivos del negocio están definidos frecuentemente en términos generales, son usados para descomponer el trabajo en tareas específicas. En ciertas situaciones la IR se enfoca en la descripción de las tareas y en el análisis de sistemas similares. Esta información proporciona la base para especificar el sistema que será construido, aunque frecuentemente se añadan al sistema tareas que no encajan con el ambiente de trabajo planificado. El nuevo sistema cambiará el ambiente de trabajo, sin embargo, es muy difícil anticipar los efectos actuales sobre la organización. Los cambios no ocurren solamente cuando un nuevo software es implementado y puesto en producción, también ocurren cuando cambia el ambiente que lo rodea (nuevas soluciones a problemas, nuevo equipo para

instalar, etc.). La necesidad de cambio es sustentada por el enorme costo de mantenimiento, aunque existen diversas razones que dificultan el mantenimiento del software, la falta de atención a la IR es la principal.
- Frecuentemente la especificación inicial es también la especificación final, lo que obstaculiza la comunicación y el proceso de aprendizaje de las personas involucradas. Ésta es una de las razones por las cuales existen sistemas inadecuados.
- Punto de vista de los clientes. Muchos sistemas tienen diferentes tipos de clientes. Cada grupo de clientes tiene necesidades diferentes y, diferentes requerimientos tienen diferentes grados de importancia para ellos. Por otro lado, pocas veces tenemos que los clientes son los usuarios, trayendo como consecuencia que los clientes soliciten procesos que causan conflictos con los solicitados por el usuario. Diferentes puntos de vistas también pueden tener consecuencias negativas, tales como datos redundantes, inconsistentes y ambiguos.
- El tamaño y complejidad de los requerimientos ocasiona desentendimiento, dificultad para enfocarse en un solo aspecto a la vez y dificultad para visualizar relaciones existentes entre requerimientos.
- Barreras de comunicación: La ingeniería de requerimientos depende de una intensa comunicación entre clientes y analistas de requerimientos. Sin embargo, existen problemas que no pueden ser resueltos mediante la comunicación. Para remediar esto, se deben abordar nuevas técnicas operacionales que ayuden a superar estas barreras y así ganar experiencia dentro del marco del sistema propuesto.
- Evolución e integración del sistema: Pocos sistemas son construidos desde cero. En la práctica, los proyectos se derivan de sistemas ya existentes. Por lo tanto, los analistas de requerimientos deben comprender esos sistemas, que por lo general son una integración de componentes de varios proveedores. Para encontrar una solución a problemas de este tipo, es muy importante hacer planeamientos entre los requerimientos y la fase de diseño. Esto minimizará la cantidad de fallos directos en el código.
- Documentación de requerimientos: Los documentos de ingeniería de requerimientos son largos. La mayoría están compuestos de cientos o miles de páginas, donde cada página contiene muchos detalles que pueden tener efectos profundos en el resto del sistema. Normalmente, las personas se encuentran con dificultades para comprender documentos de este tamaño, sobre todo si lo leen cuidadosamente. Es casi imposible leer un documento de especificación de gran tamaño, pues difícilmente una persona puede memorizar los detalles del documento. Esto causa problemas y errores que no son detectados hasta después de haberse construido el sistema.

Metodología de la Ingeniería de Requerimientos

Objetivo de la Metodología

El objetivo de esta metodología es la definición de las tareas a realizar, los productos a obtener y las técnicas a emplear durante la actividad de elicitación de requerimientos de la fase de ingeniería de requerimientos del desarrollo de software.

En esta metodología se distinguen dos tipos de productos: los productos entregables y los productos no entregables o internos. Los productos entregables son aquellos que se entregan oficialmente al cliente como parte del desarrollo en fechas previamente acordadas, mientras que los no entregables son productos internos al desarrollo que no se entregan al cliente.

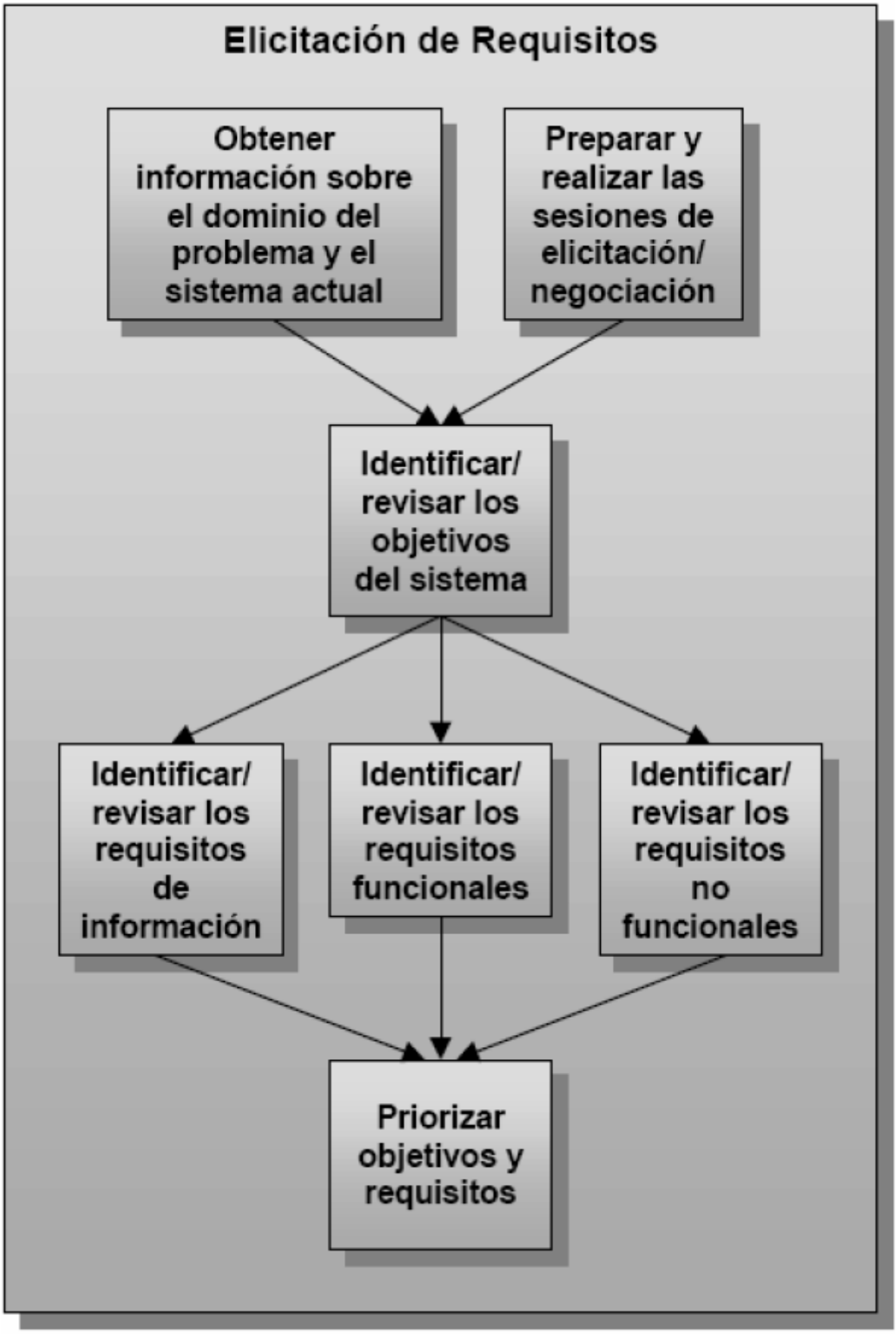
El único producto entregable definido en esta metodología es el Documento de Requerimientos del Sistema (DRS). La estructura de este documento es la siguiente: tareas recomendadas, productos entregables (en este caso el DRS), y por último, se describen algunas de las técnicas recomendadas para obtener productos. También se incluye como apéndice un ejemplo de aplicación de esta metodología.

Tareas Recomendadas

Las tareas recomendadas para obtener los productos descritos en esta metodología son las siguientes:

- **Tarea 1**: Obtener información sobre el dominio del problema y el sistema actual.
- **Tarea 2**: Preparar y realizar las reuniones de elicitación/negociación.
- **Tarea 3**: Identificar/revisar los objetivos del sistema.
- **Tarea 4**: Identificar/revisar los requerimientos de almacenamiento de información.
- **Tarea 5**: Identificar/revisar los requerimientos funcionales.
- **Tarea 6**: Identificar/revisar los requerimientos no funcionales.
- **Tarea 7**: Priorizar objetivos y requerimientos.

El orden recomendado de realización para estas tareas e: 1 … 7, aunque las tareas 4, 5 y 6 pueden realizarse simultáneamente o en cualquier orden que se considere oportuno (ver figura). La tarea 1 es opcional y depende del conocimiento previo que tenga el equipo de desarrollo sobre el dominio del problema y el sistema actual.



Tarea 1: Obtener información sobre el dominio del problema y el sistema actual

Objetivos

- Conocer el dominio del problema.
- Conocer la situación actual.

Descripción

Antes de mantener las reuniones con los clientes y usuarios e identificar los requerimientos es fundamental conocer el dominio del problema y los contexto organizacional y operacional, es decir, la situación actual.

Enfrentarse a un desarrollo sin conocer las características principales ni el vocabulario propio de su dominio suele provocar que el producto final no sea el esperado por clientes ni usuarios.

Por otro lado, mantener reuniones con clientes y usuarios sin conocer las características de su actividad hará que probablemente no se entiendan sus necesidades, y que su confianza inicial hacia el desarrollo se vea deteriorada enormemente.

Esta tarea es opcional, ya que puede que no sea necesario realizarla si el equipo de desarrollo tiene experiencia en el dominio del problema y el sistema actual es conocido.

Productos internos

- Información recopilada: libros, artículos, folletos comerciales, desarrollos previos sobre el mismo dominio, etc.
- Modelos del sistema actual.

Productos entregables

- Introducción, participantes en el proyecto (principalmente clientes y desarrolladores), y descripción del sistema actual como parte del DRS.

Técnicas recomendadas

Obtener información de fuentes externas al negocio del cliente: folletos, informes sobre el sector, publicaciones, consultas con expertos, etc.

- En el caso de que se trate de un dominio muy específico puede ser necesario recurrir a fuentes internas al propio negocio del cliente, en cuyo caso pueden utilizarse las técnicas auxiliares de elicitación de requerimientos como el estudio de documentación, observación in situ, cuestionarios, inmersión o apredizaje, etc.
- Modelado del sistema actual.

Tarea 2: Preparar y realizar las sesiones de elicitación/negociación

Objetivos

- Identificar a los usuarios participantes.
- Conocer las necesidades de clientes y usuarios.
- Resolver posibles conflictos.

Descripción

Teniendo en cuenta la información recopilada en la tarea anterior, en esta tarea se deben preparar y realizar las reuniones con los clientes y usuarios participantes con objeto de obtener sus necesidades y resolver posibles conflictos que se hayan detectado en iteraciones previas al proceso.

Esta tarea es especialmente crítica y ha de realizarse con especial cuidado, ya que generalmente el equipo de desarrollo no conoce los detalles específicos de la organización para la que se va a desarrollar el sistema y, por otra parte, los clientes y posibles usuarios no saben qué necesita saber el equipo de desarrollo para llevar a cabo su labor.

Productos internos

- Notas tomadas durante las reuniones, transcripciones o actas de reuniones, formularios, grabaciones en cinta o vídeo de las reuniones o cualquier otra documentación que se considere oportuna.

Productos entregables

- Participantes en el proyecto, en concreto los usuarios participantes, como parte del DRS.
- Objetivos, requerimientos o conflictos, que se hayan identificado claramente durante las sesiones de elicitación, como parte del DRS.

Técnicas recomendadas

- Técnicas de elicitación de requerimientos.
- Técnicas de negociación como WinWin.

Tarea 3: Identificar/revisar los objetivos del sistema

Objetivos

- Identificar los objetivos que se esperan alcanzar mediante el sistema software a desarrollar.
- Revisar, en el caso de que haya conflictos, los objetivos previamente identificados.

Descripción

A partir de la información obtenida en la tarea anterior, en esta tarea se deben identificar qué objetivos se esperan alcanzar una vez que el sistema software a desarrollar se encuentre en explotación o revisarlos en función de los conflictos identificados. Puede que los objetivos hayan sido proporcionados antes de comenzar el desarrollo.

Productos internos

- No hay productos internos en esta tarea.

Productos entregables

- Objetivos del sistema como parte del DRS.

Técnicas recomendadas

- Análisis de factores críticos de éxito o alguna técnica similar de identificación de objetivos.
- Especificación de los objetivos del sistema.

Tarea 4: Identificar/revisar los requerimientos de almacenamiento de información

Objetivos

- Identificar los requerimientos de almacenamiento de información que deberá cumplir el sistema software a desarrollar.
- Revisar, en el caso de que haya conflictos, los requerimientos de almacenamiento de información previamente identificados.

Descripción

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos identificados en la tarea 3 y el resto de los requerimientos, en esta tarea se debe identificar (o revisar si existen conflictos) qué información relevante para el cliente deberá gestionar y almacenar el sistema software a desarrollar.

Inicialmente se partirán de conceptos generales para posteriormente ir detallándolos hasta obtener todos los datos relevantes.

Productos internos

- No hay productos internos en esta tarea.

Productos entregables

- Requerimientos de almacenamiento de información como parte del DRS.

Técnicas recomendadas

- Definición de requerimientos de almacenamiento de información

Tarea 5: Identificar/revisar los requerimientos funcionales

Objetivos

- Identificar los actores del sistema de software a desarrollar.
- Identificar los requerimientos funcionales (casos de uso) que deberá cumplir el sistema software a desarrollar.
- Revisar, en el caso de que haya conflictos, los requerimientos funcionales previamente identificados.

Descripción

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos identificados en la tarea 3 y el resto de los requerimientos, en esta tarea se debe identificar (o revisar si existen conflictos) qué debe hacer el sistema a desarrollar con la información identificada en la tarea anterior.

Inicialmente se identificarán los actores que interactuarán con el sistema, es decir aquellas personas u otros sistema que serán los orígenes o destinos de la información que consumirá o producirá el sistema a desarrollar y que forman su entorno.

A continuación se identificarán los casos de uso asociados a los actores, los pasos de cada caso de uso, y posteriormente se detallarán los casos de uso con las posibles excepciones hasta definir todas las situaciones posibles.

Productos internos

- No hay productos internos en esta tarea.

Productos entregables

- Requerimientos funcionales como parte del DRS.

Técnicas recomendadas

- Casos de uso.
- Definición de actores.
- Definición de los requerimientos funcionales.

Tarea 6: Identificar/revisar los requerimientos no funcionales

Objetivos

- Identificar los requerimientos no funcionales del sistema software a desarrollar.

Descripción

A partir de la información obtenida en las tareas 1 y 2, y teniendo en cuenta los objetivos identificados en la tarea 3 y el resto de los requerimientos, en esta tarea se deben identificar (o revisar si existen conflictos), los requerimientos no funcionales, normalmente de carácter técnico o legal.

Algunos tipos de requerimientos que se suelen incluir en esta sección son los siguientes:

- **Requerimientos de comunicaciones del sistema**: Son requerimientos de carácter técnico relativos a las comunicaciones que deberá soportar el sistema software a desarrollar. Por ejemplo: el sistema deberá utilizar el protocolo TCP/IP para las comunicaciones con otros sistemas.
- **Requerimientos de interfaz de usuario**: Este tipo de requerimientos especifica las características que deberá tener el sistema en su comunicación con el usuario. Por ejemplo: la interfaz de usuario del sistema deberá ser consistente con los estándares definidos en IBM’s Common User Access. Se debe ser cuidadoso con este tipo de requerimientos, ya que, en esta fase de desarrollo todavía no se conocen bien las dificultades que pueden sufrir ala hora de diseñar e implementar las interfaces, por esto no es conveniente entrar en detalles demasiado específicos.
- **Requerimientos de fiabilidad**: Los requerimientos de fiabilidad deben establecer los factores que se requieren para la fiabilidad del software en tiempo de explotación. La fiabilidad mide la probabilidad del sistema de producir una respuesta satisfactoria a las demandas del usuario. Por ejemplo: la tasa de fallos del sistema no podrá ser superior a 2 fallos por semana.
- **Requerimientos de entorno de desarrollo**: Este tipo de requerimientos especifican si el sistema debe desarrollarse con un producto específico. Por ejemplo: el sistema deberá desarrollarse con Oracle 7 como servidor y clientes Visual Basic 4.
- **Requerimientos de portabilidad**: Los requerimientos de portabilidad definen qué características deberá tener el software para que sea fácil utilizarlo en otra máquina o bajo otro sistema operativo. Por ejemplo: el sistema deberá funcionar en los sistemas operativos Windows 95, Windows 98 y Windows NT 4.0, siendo además posible el acceso al sistema a través de Internet usando cualquier navegador compatible con HTML 4.0.

Productos internos

- No hay productos internos en esta tarea.

Productos entregables

- Requerimientos no funcionales del sistema como parte del DRS.

Técnicas recomendadas

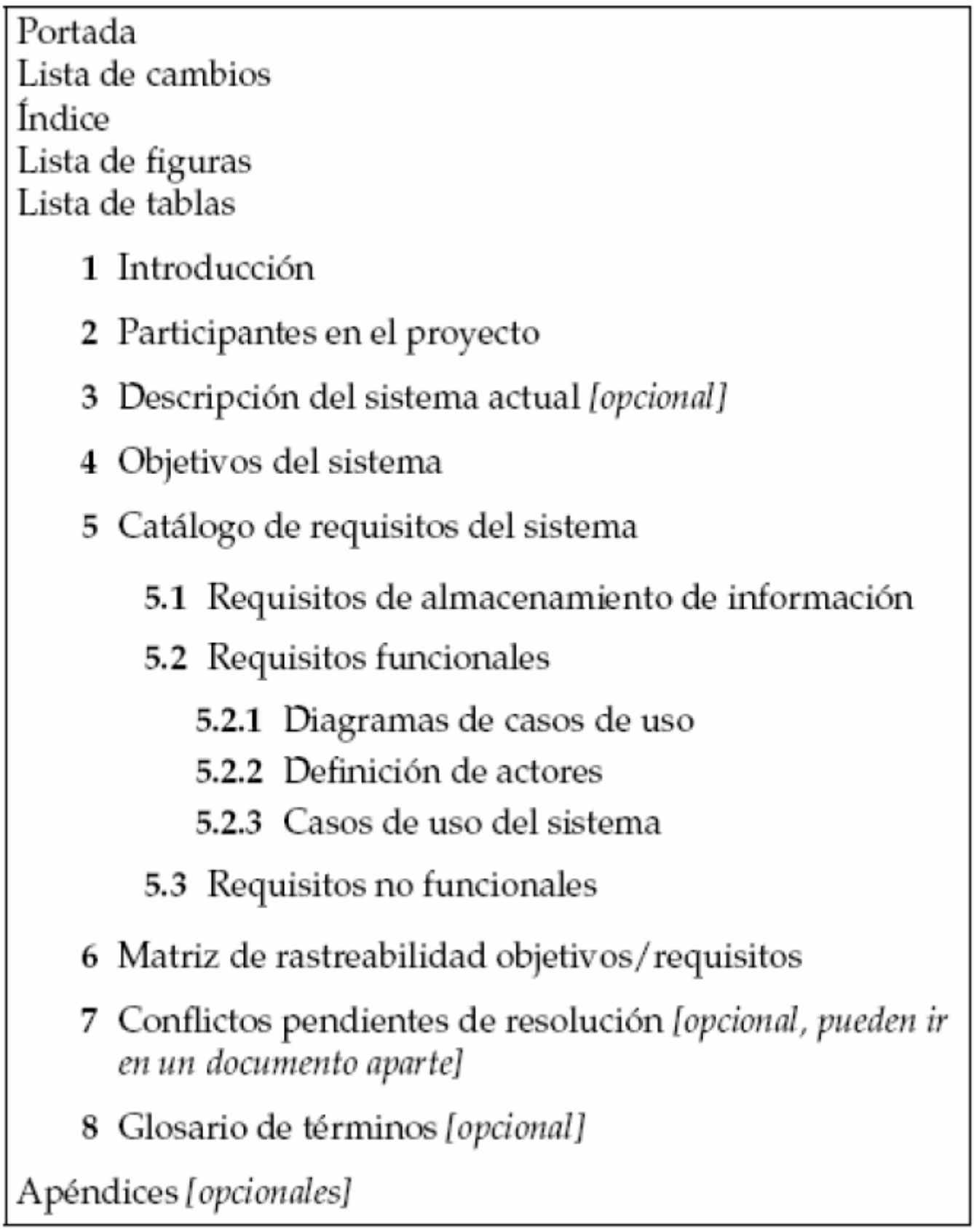
- Definición de requerimientos no funcionales

Productos Entregables

El único producto entregable que se contempla en esta metodología es el Documento de Requerimientos del Sistema (DRS).

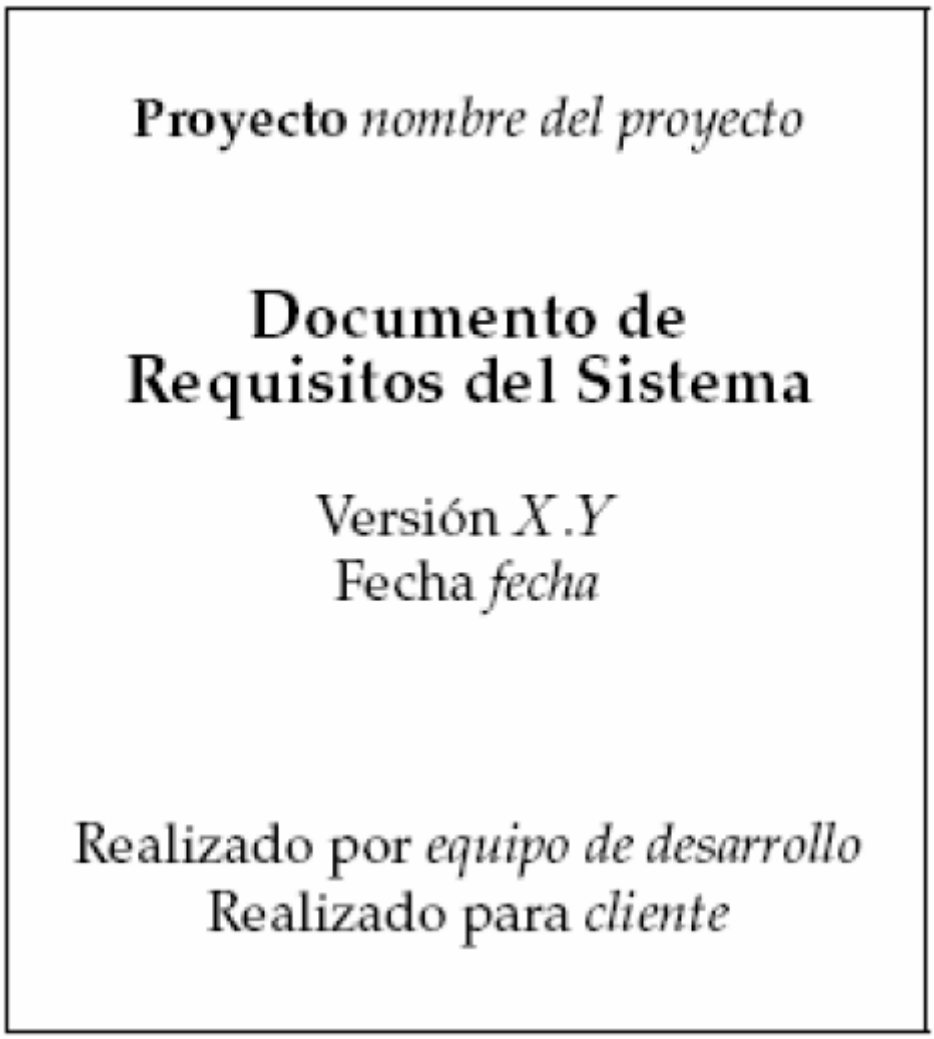
Documento de Requerimientos del Sistema

La estructura del DRS puede verse en la siguiente figura. En las siguientes secciones se describe con detalle cada sección del DRS.



Portada

La portada del DRS debe tener el formato que puede verse en la siguiente figura:



Los elementos que deben aparecer son los siguientes:

- **Nombre del proyecto**: el nombre del proyecto al que pertenece el DRS.
- **Versión**: la versión del DRS que se entrega al cliente. La versión se compone de dos números X e Y. El primero indica la versión, y se debe incrementar cada vez que se hace una nueva entrega formal al cliente. Cuando se incremente el primer número, el segundo debe volver a comenzar en cero. El segundo número indica cambios dentro de la misma versión aún no entregada,y se debe incrementar cada vez que se publica una versión con cambios respecto a la última que se publicó y que no se vaya a entregar formalmente todavía. Este tipo de versiones pueden ser internas al equipo de desarrollo o ser entregadas al cliente a título orientativo.
- **Fecha**: fecha de la publicación de la versión.
- **Equipo de desarrollo**: nombre de la empresa o equipo de desarrollo.
- **Cliente**: nombre del cliente, normalmente otra empresa.

Lista de cambios

El documento debe incluir una lista de cambios en la que se especifiquen, para cada versión del documento, los cambios producidos en el mismo con un formato similar al que puede verse en la siguiente figura. Para cada cambio realizado se debe incluir el número de orden, la fecha, una descripción y los autores.

Núm.	Fecha	Descripción	Autores
0	$fecha_0$	Versión x.y	$autor_0$
1	$fecha_1$	$descripción\ cambio_1$	$autor_1$
⋮	⋮	⋮	⋮
n	$fecha_n$	$descripción\ cambio_n$	$autor_n$

Índice

El índice del DRS debe indicar la página en la que se comienza cada sección, subsección o apartado del documento. En la medida de los posible, se sangrarán las entradas del índice para ayudar a comprender la estructura del documento.

Listas de figuras y tablas

El DRS deberá incluir listas de las figuras y tablas que aparezcan en el mismo. Dichas listas serán dos índices que indicarán el número, la descripción y la página en que aparece cada figura o tabla del DRS.

Introducción

Esta sección debe contener una descripción breve de las principales características del sistema software que se va a desarrollar, la situación actual que genera la necesidad del nuevo desarrollo, la problemática que se acomete, y cualquier otra consideración que sitúe al posible lector en el contexto oportuno para comprender el resto del documento.

Participantes en el proyecto

Esta sección debe contener una lista con todos los participantes en el proyecto, tanto desarrolladores como clientes y usuarios. Para cada participante se deberá indicar su nombre, el papel que desempeña en el proyecto, la organización a la que pertenece y cualquier otra información adicional que se considere oportuna.

Descripción del sistema actual

Esta sección debe contener una descripción del sistema actual en el caso de que se haya acometido su estudio. Para describir el sistema actual puede utilizarse cualquier técnica que se considere oportuno.

Objetivos del sistema

Esta sección debe contener una lista con los objetivos que se esperan alcanzar cuando el sistema software a desarrollar esté en explotación.

Catálogo de requerimientos del sistema

Esta sección se divide en subsecciones en las que se describen los requerimientos del sistema.

Cada uno de los grandes grupos de requerimientos (de almacenamiento de información, funcionales y no funcionales) podrá dividirse para ayudar a la legibilidad del documento, por ejemplo dividiendo cada subsección en requerimientos asociados a un determinado objetivo, requerimientos con características comunes, etc.

Requerimientos de almacenamiento de información

Esta subsección debe contener la lista de requerimientos de almacenamiento de información que se hayan identificado.

Requerimientos funcionales

Esta subsección debe contener la lista de requerimientos funcionales que se hayan identificado, dividiéndose en los siguientes apartados que se describen a continuación:

- **Diagrama de casos de uso**: Este apartado debe contener los diagramas de casos de uso del sistema que se hayan realizado.
- **Definición de los actores**: Este apartado debe contener una lista con los actores que se hayan identificado.
- **Casos de uso del sistema**: Este apartado debe contener los casos de uso que se hayan identificado.

Requerimientos no funcionales

Esta subsección debe contener la lista de los requerimientos no funcionales del sistema que se hayan identificado.

Matriz de rastreabilidad objetivos/requerimientos

Esta sección debe contener una matriz objetivo-requisito, de forma que para cada objetivo se pueda conocer con qué requerimientos está asociado. El formato de la matriz de rastreabilidad puede verse en la siguiente figura:

	OBJ–01	OBJ–02	. . .	OBJ-*n*
RI–01	•	•		
RI–02		•		
. . .				
RF–01	•			
RF–02	•	•		
. . .				
RNF–01				•
RNF–02		•		
. . .				

Conflictos pendientes de resolución

Esta sección, que se incluirá en el caso de que no se opte por registrar los conflictos en un documento aparte, deberá contener los conflictos identificados durante el proceso y que aún están pendientes de resolución.

Glosario de términos

Esta sección, que se incluirá si se considera oportuno, deberá contener una lista ordenada alfabéticamente de los términos específicos del dominio del problema, acrónimos y abreviaturas que aparezcan en el documento y que se considere que su significado deba ser aclarado. Cada término deberá acompañarse de su significado.

Apéndices

Los apéndices se usarán para proporcionar información adicional a la documentación obligatoria del documento. Sólo deben aparecer si se consideran oportunos y se identificarán con letras ordenadas alfabéticamente: A, B, C, etc.

Técnicas

A continuación, se describen algunas de las técnicas que se proponen en esta metodología para obtener los productos de las tareas que se han descrito. Las técnicas más habituales en la elicitación de requerimientos son las entrevistas, el Joint Application Development (JAD) o Desarrollo Conjunto de Aplicaciones, el brainstorming o tormenta de ideas y la utilización de escenarios, más conocidos como casos de uso.

Estas técnicas, que se describen en los siguientes apartados, se suelen completar con otras técnicas complementarias como la observación in situ, el estudio de documentación, los cuestionarios, la inmersión en el negocio del cliente o haciendo que los ingenieros de requerimientos sean aprendices del cliente.

Entrevistas

Las entrevistas son la técnica de elicitación más utilizada, y de hecho son practicamente inevitables en cualquier desarrollo ya que son una de las formas de comunicación más naturales entre personas. En las entrevistas se pueden identificar tres fases: preparación, realización y análisis.

- **Preparación de entrevistas**: Las entrevistas no deben improvisarse, por lo que conviene realizar las siguientes tareas previas:
  - **Estudiar el dominio del problema**: Conocer las categorías y conceptos de la comunidad de clientes y usuarios es fundamental para poder entender las necesidades de dicha comunidad y su forma de expresarlas, y para generar en los clientes y usuarios la confianza de que el ingeniero de requerimientos entiende sus problemas. Para conocer el dominio del problema se puede recurrir a técnicas de estudio de documentación, a bibliografía sobre el tema, documentación de proyectos similares realizados anteriormente, la inmersión dentro de la organización para la que se va a desarrollar o a periodos de aprendizaje por parte de los ingenieros de requerimientos.
  - **Seleccionar a las personas a las que se va a entrevistar**: Se debe minimizar el número de entrevistas a realizar, por lo que es fundamental seleccionar a las personas a entrevistar. Normalmente se comienza por los directivos (que pueden ofrecer una visión global), y se continúa con los futuros usuarios (que pueden aportar información más detallada) y con el personal técnico (que aporta detalles sobre el entorno operacional de la organización). Conviene también estudiar el perfil de los entrevistados, buscando puntos en común con el entrevistador que ayuden a romper el hielo.
  - **Determinar el objetivo y contenido de las entrevistas**: Para minimizar el tiempo de la entrevista es fundamental fijar el objetivo que se pretende alcanzar y determinar previamente su contenido. Previamente a su realización, se pueden enviar cuestionarios (que los futuros entrevistados deben rellenar y devolver) y un pequeño documento de introducción al proyecto de desarrollo (de forma que el entrevistado conozca los temas que se van a tratar, y el entrevistador recoja información para preparar la entrevista). Es importante que los cuestionarios, si se usan, se preparen cuidadosamente teniendo en cuenta quién los va a responder y no incluir conceptos que se asuman conocidos cuando puedan no serlo.
  - **Planificar las entrevistas**: La fecha, hora, lugar y duración de la entrevista deben fijarse teniendo en cuenta siempre la agenda del entrevistado. En

general, se deben buscar sitios agradables donde no se produzcan interrupciones y que resulten naturales a los entrevistados.

- **Realización de entrevistas**: Dentro de la realización de las entrevistas se distinguen tres etapas:
  - **Apertura**: El entrevistador debe presentarse e informar al entrevistado sobre la razón de la entrevista, qué se espera conseguir, cómo se utilizará la información, la mecánica de las preguntas, etc. Si se va a utilizar algún tipo de notación gráfica o matemática que el entrevistado no conozca debe explicarse antes de utilizarse. Es fundamental causar buena impresión en los primeros minutos.
  - **Desarrollo**: La entrevista en sí no debería durar más de dos horas, distribuyendo el tiempo en un 20% para el entrevistador y un 80% para el entrevistado. Se deben evitar los monólogos y mantener el control por parte del entrevistador, contemplando la posibilidad de que una tercera persona tome notas durante la entrevista o grabar la entrevista en cinta de vídeo o audio, siempre que el entrevistado esté de acuerdo. Durante esta fase se pueden emplear distintas técnicas:
    - **Preguntas abiertas**: También denominadas de libre contexto, estas preguntas no pueden responderse con un “sí” o un “no”, permiten una mayor comunicación y evitan la sensación de interrogatorio. Por ejemplo, “¿Qué se hace para registrar un pedido?”, “Dígame qué se debe hacer cuando un cliente pide una factura” o “¿Cómo se rellena un albarán?”. Estas preguntas se suelen utilizar al comienzo de la entrevista, pasando posteriormente a preguntas más concretas. En general, se debe evitar la tendencia de anticipar una respuesta a las preguntas que se formulan.
    - **Utilizar palabras apropiadas**: Se deben evitar tecnicismos que no conozca el entrevistado y palabras o frases que puedan perturbar emocionalmente la comunicación.
    - **Mostrar interés en todo momento**: Es fundamental cuidar la comunicación no verbal durante la entrevista: tono de voz, movimiento, expresión facial, etc. Por ejemplo, para animar a alguien a hablar puede asentirse con la cabeza, decir “ya entiendo”, “sí”, repetir algunas respuestas dadas, hacer pausas, poner una postura de atención, etc. Debe evitarse bostezar, reclinarse en el sillón, mirar hacia otro lado, etc.
  - **Terminación**: Al terminar la entrevista se debe recapitular para confirmar que no ha habido confusiones en la información recogida, agradecer al entrevistado su colaboración y citarle para una nueva entrevista si fuera necesario, dejando siempre abierta la posibilidad de volver a contactar para aclarar dudas que surjan al estudiar la información o al contrastarla con otros entrevistados.
- **Análisis de las entrevistas**: Una vez realizada la entrevista es necesario leer las notas tomadas, pasarlas a limpio, reorganizar la información, contrastarla con otras entrevistas o fuentes de información, etc. Una vez elaborada la información, se puede enviar al entrevistado para confirma los contenidos. También es importante evaluar la propia entrevista para determinar los aspectos mejorables.

Joint Application Development (JAD)

La técnica denominada JAD (Joint Application Development, Desarrollo Conjunto de Aplicaciones), desarrollada por IBM en 1977, es una alternativa a las entrevistas individuales que se desarrolla a lo largo de un conjunto de reuniones en grupo durante un periodo de 2 a 4 días. En estas reuniones se ayuda a los clientes y usuarios a formular problemas y explorar posibles soluciones, involucrándolos y haciéndolos sentirse partícipes del desarrollo.

Esta técnica se basa en cuatro principios: dinámica de grupo, el uso de ayudas visuales para mejorar la comunicación (diagramas, transparencias, multimedia, herramientas CASE, etc.), mantener un proceso organizado y racional y una filosofía de documentación WYSIWYG (What You See Is What You Get, lo que se ve es lo que se obtiene), por la que durante las reuniones se trabaja directamente sobre los documentos a generar.

El JAD tienen dos grandes pasos, el JAD/Plan (cuyo objetivo es elicitar y especificar requerimientos) y el JAD/Design (en el que se aborda el diseño del software). En este documento sólo se verá con detalle el primero de ellos. Debido a las necesidades de organización que requiere y a que no suele adaptarse bien a los horarios de trabajo de los clientes y usuarios, esta técnica no suele emplearse con frecuencia, aunque cuando se aplica suele tener buenos resultados, especialmente para elicitar requerimientos en el campo de los sistemas de información.

En comparación con las entrevistas individuales, presenta las siguientes ventajas:

- Ahorra tiempo al evitar que las opiniones de los clientes se contrasten por separado.
- Todo el grupo, incluyendo los clientes y los futuros usuarios, revisa la documentación generada, no sólo los ingenieros de requerimientos.
- Implica más a los clientes y usuarios en el desarrollo.

El JAD queda definida a través de los siguientes elementos:

- **Participantes del JAD**: Se pueden distinguir seis clases de participantes o roles en el JAD:
  - **Jefe del JAD**: Es el responsable de todo el proceso y asume el control durante las reuniones. Debe tener dotes de comunicación y liderazgo. Algunas habilidades importantes que debe tener son: entender y promover la dinámica de grupo, iniciar y centrar discusiones, reconocer cuándo la reunión se está desviando del tema y reconducirla, manejar las distintas personalidades y formas de ser de los participantes, evitar que decaiga la reunión aunque sea larga y difícil, etc.
  - **Analista**: Es el responsable de la producción de los documentos que se deben generar durante las sesiones JAD. Debe tener la habilidad de organizar bien las ideas y expresarlas claramente por escrito. En el caso de que se utilicen herramientas software durante las sesiones, debe ser capaz de manejarlas eficientemente.
  - **Patrocinador ejecutivo**: Es el que tiene la decisión final de que se lleve a cabo el desarrollo. Debe proporcionar a los demás participantes información sobre la necesidad del nuevo sistema y los beneficios que se espera obtener de él.
  - **Representantes de los usuarios**: Durante el JAD/Plan suelen ser directivos con una visión global del sistema. Durante el JAD/Design suelen incorporarse futuros usuarios finales.
  - **Representantes de sistemas de información**: Son personas expertas en sistemas de información que deben ayudar a los usuarios a comprender qué es o no factible con la tecnología actual y el esfuerzo que implica.
  - **Especialistas**: Son personas que pueden proporcionar información detallada sobre aspectos muy concretos, tanto desde el punto de vista de los usuarios porque conocen muy bien el funcionamiento de una parte de la organización, como desde el punto de vista de los desarrolladores porque conocen perfectamente ciertos aspectos técnicos de la instalación hardware de la organización.
- **Fases del JAD**: Dentro de la técnica del JAD se distinguen tres fases:
  - **Adaptación**: Es responsabilidad del jefe del JAD, ayudado por uno o dos analistas, adaptar la técnica del JAD para cada proyecto. La adaptación debe comenzar por definir el proyecto a alto nivel, para lo cual pueden ser necesarias entrevistas previas con algunos clientes y usuarios. También suele ser necesario recabar información sobre la organización para familiarizarse con el dominio del problema, por ejemplo utilizando técnicas complementarias como el estudio de documentación o la observación in situ. Una vez obtenida una primera idea de los objetivos del proyecto, es necesario seleccionar a los participantes,citarles para las reuniones y proporcionarles una lista con los temas que se van a tratar en las reuniones para que las puedan preparar. El jefe del JAD debe decidir la

duración y el número de sesiones a celebrar, definir el formato de la documentación sobre la que se trabajará y preparar transparencias introductorias y todo el material audiovisual que considere oportuno.

- **Celebración de las sesiones JAD**: Durante las sesiones, los participantes exponen sus ideas y se discuten, analizan y refinan hasta alcanzar un acuerdo. Los pasos que se recomienda seguir para este proceso son los siguientes:
  - **Presentación**: Se presenta y se da la bienvenida a todos los participantes por parte del patrocinador ejecutivo y del jefe del JAD. El patrocinador ejecutivo expone brevemente las necesidades que han llevado al desarrollo y los beneficios que se esperan obtener. El jefe del JAD explica la mecánica de las sesiones y la planificación prevista.
  - **Definir objetivos y requerimientos**: El jefe del JAD promueve la discusión para elicitar los objetivos o requerimientos de alto nivel mediante preguntas como: “¿Por qué se construye el sistema?”, “¿Qué beneficios se esperan del nuevo sistema?”, “¿Cómo puede beneficiar a la organización en el futuro?”, “¿Qué restricciones de recursos disponibles, normas o leyes afectan al proyecto?”, “¿Es importante la seguridad de los datos?”. A medida que se van elicitando requerimientos, el analista los escribe en transparencias o en algún otro medio que permita que permanezcan visibles durante la discusión.
  - **Delimitar el ámbito del sistema**: Una vez obtenido un número importante de requerimientos, es necesario organizarlos y llegar a un acuerdo sobre el ámbito del nuevo sistema. En el caso de los sistemas de información, es útil identificar a los usuarios potenciales (actores) y determinar qué tareas les ayudará a realizar (casos de uso).
  - **Documentar temas abiertos**: Aquellas cuestiones que hayan surgido durante la sesión que no se han podido resolver, deben documentarse para las siguientes sesiones y ser asignadas a una persona responsable de su solución para una fecha determinada.
  - **Concluir la sesión**: El jefe del JAD concluye la sesión revisando con los demás participantes la información elicitada y las decisiones tomadas. Se da la oportunidad a todos los participantes de expresar cualquier consideración adicional, fomentando por parte del jefe del JAD el sentimiento de propiedad y compromiso de todos los participantes sobre los requerimientos elicitados.
- **Conclusión**: Una vez terminadas las sesiones es necesario transformar las transparencias, notas y demás documentación generada en documentos formales. Se distinguen tres pasos:
  - **Completar la documentación**: Los analistas recopilan la documentación generada durante las sesiones en documentos conformes a las normas o estándares vigentes en la organización para la que se desarrolla el proyecto.
  - **Revisar la documentación**: La documentación generada se envía a todos los participantes para que la comenten. Si los comentarios son lo suficientemente importantes, se convoca otra reunión para discutirlos.
  - **Validar la documentación**: Una vez revisados todos los comentarios, el jefe del JAD envía el documento al patrocinador ejecutivo para su aprobación. Una vez aprobado el documento se envían copias definitivas a cada uno de los participantes.

Brainstorming

El brainstorming o tormenta de ideas es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios. Las sesiones de brainstorming suelen estar formadas por un número de cuatro a diez participantes, uno de los cuales es el jefe de sesión, encargado más de comenzar la sesión que de controlarla. Como técnica de elicitación de requerimientos, el brainstorming puede ayudar a generar

una gran variedad de vistas del problema, y a formularlo de diferentes formas, sobre todo al comienzo del proceso de elicitación cuando los requerimientos son todavía muy difusos.

Frente al JAD, el brainstorming tiene la ventaja de que es muy fácil de aprender y requiere poca organización, de hecho, hay propuestas de realización de brainstorming por vídeoconferencia a través de internet. Por otro lado, al ser un proceso poco estructurado, puede no producir resultados con la misma calidad o nivel de detalle que otras técnicas.

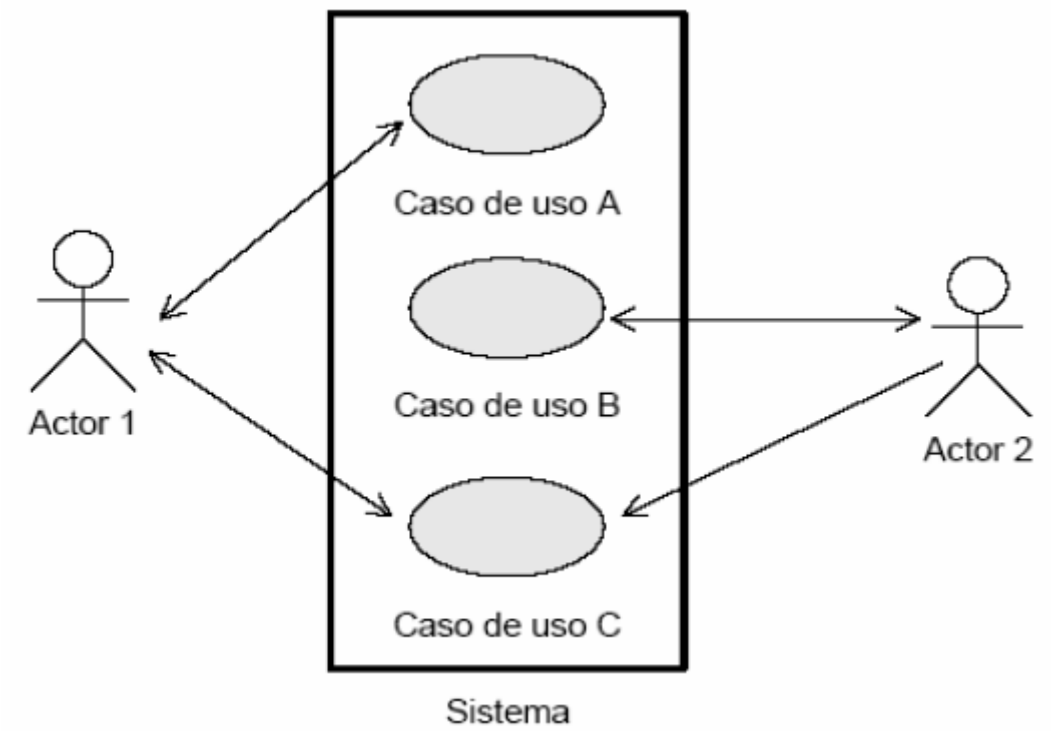
El brainstorming se puede describir a través de los siguientes elementos:

- **Fases del brainstorming**: En elbrainstorming se distinguen las siguientes fases:
  - **Preparación**: La preparación para una sesión de brainstorming requiere que se seleccione a los participantes y al jefe de la sesión, citarlos y preparar la sala donde se llevará a cabo la sesión. Los participantes en una sesión de brainstorming para elicitación de requerimientos son normalmente clientes, usuarios, ingenieros de requerimientos, desarrolladores y, si es necesario, algún experto en temas relevantes para el proyecto.
  - **Generación**: El jefe abre la sesión exponiendo un enunciado general del problema a tratar, que hace de semilla para que se vayan generando ideas. Los participantes aportan libremente nuevas ideas sobre el problema semilla, bien por un orden establecido por el jefe de la sesión, bien aleatoriamente. El jefe es siempre el responsable de dar la palabra a un participante. Este proceso continúa hasta que el jefe decide parar, bien porque no se están generando suficientes ideas, en cuyo caso la reunión se pospone, bien porque el número de ideas sea suficiente para pasar a la siguiente fase. Durante esta fase se deben observar las siguientes reglas:
    - Se prohíbe la crítica de ideas, de forma que los participantes se sientan libres de formular cualquier idea.
    - Se fomentan las ideas más avanzadas, que aunque no sean factibles, estimulan a los demás participantes a explorar nuevas soluciones más creativas.
    - Se debe generar un gran número de ideas, ya que cuantas más ideas se presenten más probable será que se generen mejores ideas.
    - Se debe alentar a los participantes a combinar o completar las ideas de otros participantes. Para ello, es necesario, al igual que en la técnica del JAD, que todas las ideas generadas estén visibles para todos los participante en todo momento. Una posibilidad es utilizar como semilla objetivos del sistema e ir identificando requerimientos.
  - **Consolidación**: En esta fase se deben organizar y evaluar las ideas generadas durante la fase anterior. Se suelen seguir tres pasos:
    - **Revisar ideas**: Se revisan las ideas generadas para clarificarlas. Es habitual identificar ideas similares, en cuyo caso se unifican en un solo enunciado.
    - **Descartar ideas**: Aquellas ideas que los participantes consideren excesivamente avanzadas se descartan.
    - **Priorizar ideas**: Se priorizan las ideas restantes, identificando las absolutamente esenciales, las que estarían bien pero que no son esenciales, y las que podrían ser apropiadas para una próxima versión del sistema a desarrollar.
  - **Documentación**: Después de la sesión, el jefe produce la documentación oportuna conteniendo las ideas priorizadas y comentarios generados durante la consolidación.

Casos de uso

Los casos de uso son una técnica para la especificación de requerimientos funcionales propuesta inicialmente en Jacobson y que actualmente forma parte de UML.

Un caso de uso es la descripción de una secuencia de interacciones entre el sistema y uno o más actores en la que se considera al sistema como una caja negra y en la que los actores obtienen resultados observables en la siguiente figura:



Los actores son personas u otros sistemas que interactúan con el sistema cuyos requerimientos se están describiendo.

Los casos de uso presentan ciertas ventajas sobre la descripción meramente textual de los requerimientos funcionales, ya que facilitan la elicitación de requerimientos y son fácilmente comprensibles por los clientes y usuarios. Además, pueden servir de base a las pruebas del sistema y a la documentación para los usuarios.

A pesar de ser una técnica ampliamente aceptada, existen múltiples propuestas para su utilización concreta. En esta metodología se propone la utilización de los casos de uso como técnica tanto de elicitación como de especificación de los requerimientos funcionales del sistema.

Diagramas de casos de uso

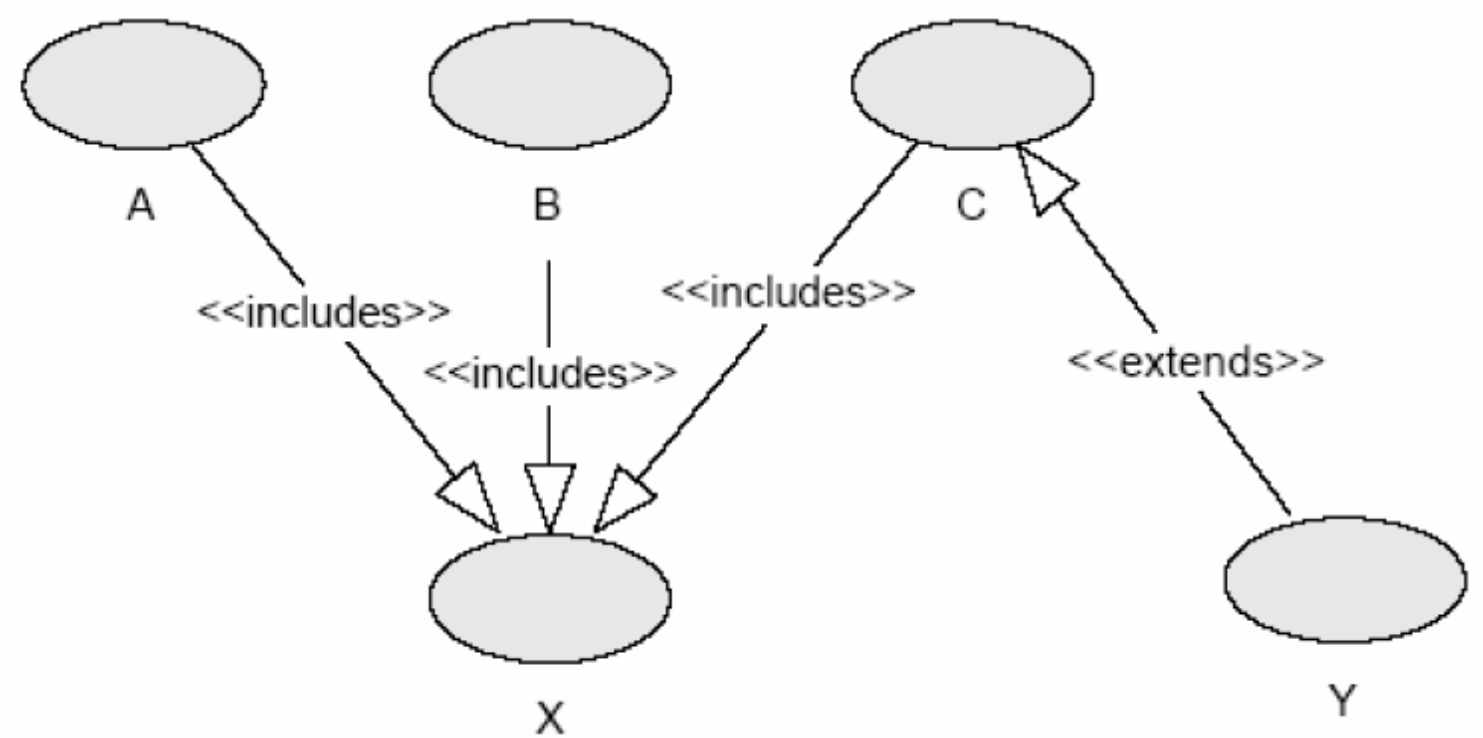
Los casos de uso tienen una representación gráfica en los denominados diagramas de casos de uso. En estos diagramas, los actores se representan en forma de pequeños monigotes, y los casos de uso se representan por elipses contenidos dentro de un rectángulo que representa al sistema. La participación de los actores en los casos de uso se indica por una flecha entre el actor y el caso de uso que apunta en la dirección en la que fluye la información. Un ejemplo de este tipo de diagramas se vio en la imagen anterior.

Los diagramas de casos de uso sirven para proporcionar una visión global del conjunto de casos de uso de un sistema así como de los actores y los casos de uso en los que éstos intervienen. Las interacciones concretas entre los actores y el sistema no se muestran en este tipo de diagramas.

Relaciones entre casos de uso

A veces conviene establecer relaciones entre distintos casos de uso para simplificar su descripción. Las dos relaciones posibles y su semántica, cuya representación gráfica puede verse en el siguiente ejemplo son las siguientes:

- **includes**: Se dice que un caso de uso A incluye el caso de uso B, cuando B es una parte del caso de uso A, es decir, la secuencia de interacciones de B forma parte de la secuencia de interacciones de A. El caso de uso B se realiza siempre dentro del caso de uso A. Además, siempre que ocurre A ocurre también B, por lo que se dice que B es un caso de uso abstracto. Un caso de uso es abstracto si no puede ser realizado por sí mismo, por lo que sólo tiene significado cuando se utiliza para describir alguna funcionalidad que es común a otros casos de uso. Por otra parte, un caso de uso será concreto si puede ser iniciado por un actor y realizado por sí mismo. Se suele utilizar esta relación cuando se detectan subsecuencias de interacciones comunes a varios casos de uso. Dichas subsecuencias comunes se extren como “factor común” de los casos de uso que las contienen. Posteriormente son incluido por los casos de uso de los que se han “extraído”. De esta forma se evita repetir las mismas subsecuencias de interacciones una y otra vez en varios casos de uso.
- **extends**: Un caso de uso A extiende a otro caso de uso B cuando A es una subsecuencia de interacciones de B, que ocurre en una determinada circunstancia. En cierta forma, A completa la funcionalidad de B. El caso de uso A puede realizarse o no cuando se realiza el caso de uso B, según las circunstancias. Por otro lado, el caso de uso A puede ser un caso de uso abstracto o concreto, en cuyo caso puede ocurrir sin necesidad de que ocurra el caso de uso B.

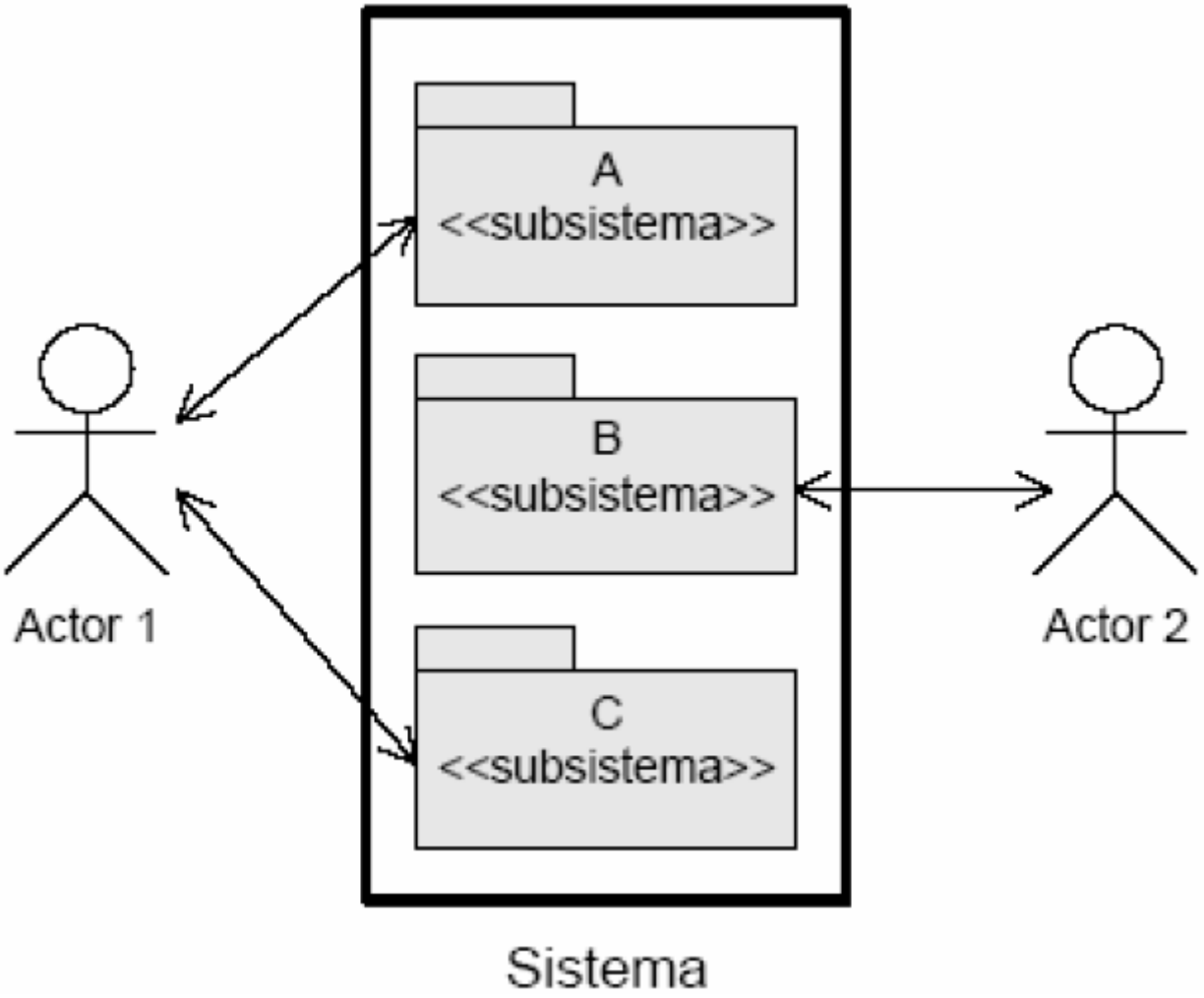


Organización de casos de uso

En la mayoría de sistemas, el número de casos de uso es lo suficientemente elevado como para que sea oportuno organizarlos de alguna forma en lugar de tener una lista plana por la que no es fácil navegar. Una posible forma de organizar los casos de uso es recurrir a los paquetes. De esta forma, los casos de uso pueden organizarse en niveles, facilitando así su comprensión.

Cada paquete contiene a otros paquetes o a varios casos de uso.

En el caso de que los casos de uso se agrupen por criterios funcionales, los paquetes que los agrupan pueden estereotiparse como subsistemas, tal como puede verse en la siguiente figura:



Prototipado

Los prototipos permiten al desarrollador crear un modelo del software que debe ser construido.

Un prototipo es la primera versión de un nuevo tipo de producto, en el que se han incorporado sólo algunas características del sistema final, o no se han realizado completamente.

Es un modelo o maqueta del sistema que se construye para comprender mejor el problema y sus posibles soluciones. Esto permite:

- Evaluar mejor los requerimientos.
- Probar opciones de diseño.

El uso principal es la ayuda a los clientes y los desarrolladores para entender los requerimientos del sistema. El prototipo puede ser usado para entrenamiento antes de que el sistema final sea entregado. El prototipo también puede ser utilizado para pruebas.

Las características de los prototipos son:

- Funcionalidad limitada.
- Poca fiabilidad.
- Características de operación pobres.

Hay que tener en cuenta que el prototipo representa aproximadamente un 10% del presupuesto del proyecto, y normalmente supone pocos días de desarrollo.

Al igual que todos los enfoques orientados al proceso de desarrollo del software, el prototipado comienza con la captura de requerimientos. Desarrolladores y clientes se reúnen y definen los objetivos globales del software, identifican todos los requerimientos que son conocidos, y señalan áreas en las que será necesaria la profundización en las definiciones. Después de esto, tiene lugar un “diseño rápido”. El diseño rápido se centra en una representación de aquellos aspectos del software que serán visibles al usuario (por ejemplo, entradas y formatos de las salidas). El diseño rápido lleva a la construcción de un prototipo. El prototipo es evaluado por el cliente y el usuario, y utilizado para refinar los requerimiento del software a ser desarrollado. Un proceso de iteración tiene lugar a medida que el prototipo es perfeccionado para satisfacer las necesidades del cliente y permitir al mismo tiempo una mejor comprensión del problema por parte del desarrollador.

Existen principalmente dos tipos de prototipos:

- **Prototipo rápido (concept prototype)**: El prototipado rápido es un mecanismo para lograr la validación inicial. Se utiliza para validar requerimientos en una etapa previa al diseño específico. En este sentido, el prototipo puede ser visto como una aceptación tácita de que los requerimientos no son totalmente conocidos o entendidos antes del diseño y la implementación. El prototipo rápido puede ser usado como un medio para explorar nuevos requerimientos y así ayudar a controlar su constante evolución.
- **Prototipo evolutivo**: Desde una perspectiva diferente, todo el ciclo de vida de un producto puede ser visto como una serie incremental de detallados prototipos acumulativos. Tradicionalmente, el ciclo de vida está dividido en dos fases distintas: desarrollo y mantenimiento. La experiencia ha demostrado que esta distinción es arbitraria y va en contra de la realidad, ya que, la mayor parte del costo del software ocurre después de que el producto se ha entregado. El desarrollo evolutivo del ciclo de vida del software considera a la primera entrega como un prototipo inicial en uso. Modificaciones y mejoras subsecuentes resultan en nuevas entregas de prototipos más maduros. Este proceso continúa hasta que se haya desarrollado el producto final (véase la figura). La adopción de esta óptica elimina la distinción arbitraria entre desarrollo y mantenimiento, resultando en un importante cambio de mentalidad que afecta las estrategias para la estimación de costos, enfoques de desarrollo y adquisición de productos.

Herramientas para el prototipado

- Lenguajes dinámicos de alto nivel. Los más usados son:
  - Smalltalk (basado en objetos, sistemas interactivos)
  - Java (basado en objetos, sistemas interactivos)

    - Prolog (lógico, procesamiento simbólico)
    - LISP (basado en listas, procesamiento simbólico)
- Lenguajes de cuarta generación (4GLs) (programación de BBDD):
    - La mayoría de aplicaciones de gestión son interactivas e implican la manipulación de una BD y la producción de salidas que involucran organizar y dar formato a esos datos.
    - Lenguaje de programación de BBDD (y su entorno de desarrollo), que contiene conocimiento de la BD y operaciones para manipulación de la misma.
    - Lenguaje no procedimental.
    - Reducen claramente los costos del desarrollo.
    - Muy usados en prototipado evolutivo.
    - Muchos 4GLs permiten el desarrollo de interfaces de BBDD basadas en navegadores web.
    - General SQL o código en lenguaje “de bajo nivel” como COBOL.
    - Menos eficientes que los lenguajes de programación convencionales. Por ejemplo, un programa en 4GL reescrito en C++ tiene un 50% menos de requerimientos de memoria y es 10 veces más rápido.
    - Reducen claramente los costos del desarrollo, aunque no sucede lo mismo con el mantenimiento de los mismos, ya que:
        - Son programas no estructurados difíciles de mantener.
        - No están estandarizados ni son uniformes, y por tanto, los usuarios pueden tener que reescribir totalmente los programas debido a que el lenguaje ha quedado obsoleto.
- Ensamblaje de componentes y aplicaciones.

El desarrollo de prototipos con reutilización comprende dos niveles:

- El nivel de aplicación, en el que una aplicación completa se integra con el prototipo. Por ejemplo: si el prototipo requiere procesamiento de textos, se puede integrar un sistema estándar de procesamiento de textos (MS Office).
- El nivel de componente, en el que los componentes se integran en un marco de trabajo estándar.

Dentro del nivel de componentes se usan diversos tipos de lenguajes de programación, cada uno de ellos adecuado para tareas diversas:

- Visual Basic, TCL/TK, Python, Perl, …
    - Lenguajes de alto nivel sin tipos, con facilidades gráficas.
    - Desarrollo rápido de aplicaciones pequeñas y relativamente sencillas, construidas por una persona o conjunto de personas.
    - No existe una arquitectura explícita del sistema.
- CORBA, DCOM, JavaBeans.
    - Junto con un marco arquitectónico, es más apropiado para sistemas grandes.
    - Programación distribuida.

Bibliografía

- Scribd (Alfredo Márquez)

Análisis estructurado. Diagramas de flujo de datos. Diagramas de estructura. Diccionario de datos. Flujogramas.

Análisis Estructurado

Concepto

Cuando los analistas comienzan a trabajar sobre un proyecto de sistema de información, tienen que profundizar en un área de la Organización, de la cual tienen poco conocimiento. Del trabajo del analista se espera que se produzca una mejora en el sistema.

Así que el analista debe ser capaz de:

- Aprender los detalles y procedimientos del sistema en uso.
- Prever necesidades futuras de la Organización en función del crecimiento, cambios futuros en el sector, introducción de nuevas tecnologías, etc.
- Documentar detalles del sistema actual para su comprensión y discusión por otros profesionales de la organización.
- Evaluar la efectividad y eficiencia del sistema actual y sus procedimientos.
- Recomendar modificaciones del sistema actual, o proponer un nuevo sistema completo, justificándolo en cada caso.
- Documentar las características del nuevo sistema con un nivel de detalle que permita comprender a otros sus componentes.
- Fomentar la participación de gerentes y empleados en todo el proceso.

A todas estas tareas, se les une la de cumplir los plazos establecidos. De modo que una de las claves del éxito será la de estructurar el proceso para el desarrollo del nuevo sistema.

Análisis Estructurado ¿Para qué?

Por la propia naturaleza los sistemas de información, éstos no están bien estructurados, no siguen leyes como las ciencias, dependen de muchas circunstancias para su funcionamiento (personas, influencias políticas de la organización, restricciones, etc). El analista debe luchar contra estas circunstancias y determinar los requerimientos de los sistemas de información.

Ante esta realidad, surgen preguntas como: ¿Deben dos analistas desarrollar una lista idéntica de requerimientos cuando estudian de forma independiente la misma situación? ¿Para una situación dada tenemos un único diseño correcto posible? La respuesta es que dos analistas que examinan de forma independiente una situación, sin herramientas y técnicas preestablecidas, recopilan información diferente que describa el sistema y por lo tanto en determinación de requerimientos diferentes.

Esto obliga a normalizar, a estructurar el análisis de sistemas de información.

Podemos definir análisis estructurado como:

El método para el análisis de sistemas manuales o automatizados, que conduce al desarrollo de especificaciones para sistemas nuevos o para efectuar modificaciones a los ya existentes.

El análisis estructurado permite al analista conocer un proceso (actividad) en una forma lógica y manejable al mismo tiempo que proporciona la base para asegurar que no se omite ningún detalle pertinente.

Por otra parte una de las claves del éxito de un buen análisis será el que exista una buena comunicación entre usuarios y analistas, esto obliga a disponer de un lenguaje común, sencillo y fiable de modo que permita minimizar costes y errores, y maximizar calidad.

¿Qué debemos estructurar?

El objetivo que persigue el análisis estructurado es organizar las tareas asociadas con la determinación de requerimientos para obtener la comprensión completa y exacta para una situación dada. A partir de aquí se determinan los requerimientos que serán la base de un sistema nuevo o modificado.

La palabra *estructura* significa:

1. El método intenta estructurar el proceso de determinación de los requerimientos comenzando con la documentación del sistema existente.
2. El proceso intenta incluir todos los detalles relevantes que describen al sistema en uso.
3. Fácil verificar cuando se han omitido datos relevantes.
4. La identificación de los requerimientos será similar entre varios analistas e incluirá las mejores soluciones y estrategias para las oportunidades de desarrollo de sistemas.
5. Los documentos de trabajo generados para documentar los sistemas existentes y propuestos son dispositivos de comunicación eficientes.

Componentes del análisis estructurado

El análisis estructurado hace uso de los siguientes componentes:

1. Símbolos gráficos. Iconos y convenciones para identificar y describir los componentes de un sistema junto con las relaciones entre esos componentes.
2. Diccionario de datos. Descripción de todos los datos utilizados en el sistema.
3. Descripciones de procesos y procedimientos. Declaraciones formales que emplean técnicas y lenguajes que permiten a los analistas describir actividades importantes que forman parte del sistema.
4. Reglas. Estándares para describir y documentar el sistema en forma correcta y completa.

El método de análisis estructurado es sinónimo de análisis de flujo de datos que es una herramienta para documentar el sistema existente o actual y determinar los requerimientos de información de forma estructurada.

¿Qué es análisis de flujo de datos?

Los analistas desean conocer las respuestas a cuatros preguntas: ¿Qué procesos integran el sistema? ¿Qué datos emplea cada proceso? ¿Qué datos son almacenados? ¿Qué datos entran y salen del sistema?

Como vemos el elemento fundamental en una Organización (sistema de información), van a ser los datos. Los datos son las guías de las actividades de la Organización, inician eventos, son procesados para dar información útil al personal, etc.

Seguir el flujo de datos por todos los procesos de la organización, además de ser la finalidad del análisis de flujo de datos, proporciona a los analistas información de cómo se alcanzan los objetivos en la Organización.

El análisis de flujo de datos estudia el empleo de los datos en cada actividad. Se basa en los diagramas de flujo de datos que muestra de forma gráfica la relación entre procesos y datos, y en los diccionarios de datos que describen de manera formal los datos del sistema y los sitios donde son utilizados.

La estrategia de los flujos de datos

El análisis puede pensarse de tal manera que se estudien actividades del sistema desde el punto de vista de los datos, donde se originan, cómo se utilizan o cambian, hacia dónde van. Los componentes de la estrategia de flujo de datos abarcan tanto la determinación de los requerimientos como el diseño de sistemas. Una notación bien establecida facilita la documentación del sistema actual y su análisis por todos los participantes en el proceso de determinación de requerimientos.

Herramientas para el análisis de flujo de datos

Las herramientas tienen el objetivo de ayudar a entender las características del sistema. Por lo tanto no deben de ser un fin, sino un medio para el estudio del sistema.

Las herramientas utilizadas en el análisis de flujo de datos son:

1. *Diagrama de flujo de datos.*
   Una herramienta gráfica empleada para describir y analizar el movimiento de datos a través de un sistema, incluyendo procesos, almacenamiento de datos y retrasos del sistema. Los diagramas de flujo de datos es la herramienta más importante y la base sobre la cual se desarrollan otros componentes.
   La transformación de datos de entrada en salida por medio de procesos puede describirse en forma lógica e independiente de los componentes físicos. Estos diagramas reciben el nombre de diagramas lógicos de flujo de datos, en contraste de los diagramas físicos del flujo de datos que muestran la implantación y movimiento real de datos entre personas, departamentos y estaciones de trabajo.
2. *Diccionario de datos.*
   El diccionario de datos contiene las características lógicas de los sitios donde se almacenan los datos del sistema, incluyendo nombre, descripción, alias, contenidos y organización, así como los procesos donde se emplea los datos y los sitios donde se necesita el acceso inmediato a la información. Servirán para identificar los requerimientos de las bases de datos durante el diseño del sistema.
3. *Diagrama entidad-relación.*
   Este diagrama es una descripción de la relación entre entidades (personas, lugares, eventos y objetos) de un sistema y el conjunto de información relacionado con la entidad. No considera el almacenamiento físico de datos.
4. *Gráfica de estructura* (Especificación de procesos).
   Herramienta de diseño que muestra con símbolos la relación entre módulos de procesamiento y el software de la computadora. Incluye el análisis de las transformaciones entrada transformación salida y el análisis de transacciones.

Ventajas del análisis de flujo de datos

Los analistas deben trabajar con los usuarios para hacerles comprender el funcionamiento del sistema actual y el sistema futuro, para ello se hace aconsejable utilizar un lenguaje común, sencillo y fiable, estas son las características de los diagramas de flujo de datos. Los usuarios pueden hacer sugerencias para modificar los diagramas con la finalidad de

describir las actividades con mayor exactitud, y permitirá evitar los errores desde el inicio pudiendo prevenir una posible falla del sistema.

Diagrama de Flujo de Datos (DFD)

El objetivo del diagrama de flujo de datos es la obtención de un modelo lógico de procesos que represente el sistema, con independencia de las restricciones físicas del entorno. Así se facilita su comprensión por los usuarios y los miembros del equipo de desarrollo.

El sistema se divide en distintos niveles de detalle, con el objetivo de:

- Simplificar la complejidad del sistema, representando los diferentes procesos de que consta.
- Facilitar el mantenimiento del sistema.

Descripción

Un diagrama de flujo de datos es una técnica muy apropiada para reflejar de una forma clara y precisa los procesos que conforman el sistema de información. Permite representar gráficamente los límites del sistema y la lógica de los procesos, estableciendo qué funciones hay que desarrollar. Además, muestra el flujo o movimiento de los datos a través del sistema y sus transformaciones como resultado de la ejecución de los procesos.

Esta técnica consiste en la descomposición sucesiva de los procesos, desde un nivel general, hasta llegar al nivel de detalle necesario para reflejar toda la semántica que debe soportar el sistema en estudio.

El diagrama de flujo de datos se compone de los siguientes elementos:

- **Entidad externa**: representa un ente ajeno al sistema que proporciona o recibe información del mismo. Puede hacer referencia a departamentos, personas, máquinas, recursos u otros sistemas. El estudio de las relaciones entre entidades externas no forma parte del modelo.
  Puede aparecer varias veces en un mismo diagrama, así como en los distintos niveles del DFD para mejorar la claridad del diagrama.
- **Proceso**: representa una funcionalidad que tiene que llevar a cabo el sistema para transformar o manipular datos. El proceso debe ser capaz de generar los flujos de datos de salida a partir de los de entrada, más una información constante o variable al proceso.
  El proceso nunca es el origen ni el final de los datos, puede transformar un flujo de datos de entrada en varios de salida y siempre es necesario como intermediario entre una entidad externa y un almacén de datos.
- **Almacén de datos**: representa la información en reposo utilizada por el sistema independientemente del sistema de gestión de datos (por ejemplo un fichero, base de datos, archivador, etc). Contiene la información necesaria para la ejecución del proceso.
  El almacén no puede crear, transformar o destruir datos, no puede estar comunicado con otro almacén o entidad externa y aparecerá por primera vez en aquel nivel en que dos o más procesos accedan a él.
- **Flujo de datos**: representa el movimiento de los datos, y establece la comunicación entre los procesos y los almacenes de datos o las entidades externas.
  Un flujo de datos entre dos procesos sólo es posible cuando la información es síncrona, es decir, el proceso destino comienza cuando el proceso origen finaliza su función.
  Los flujos de datos que comunican procesos con almacenes pueden ser de los siguientes tipos:
  – *De consulta*: representan la utilización de los valores de uno o más campos de un almacén o la aprobación de que los valores de los campos seleccionados cumplen

unos criterios determinados.
– *De actualización*: representan la alteración de los datos de un almacén como consecuencia de la creación de un nuevo elemento, por eliminación o modificación de otros ya existentes.
– *De diálogo*: es un flujo entre un proceso y un almacén que representa una consulta y una actualización.

Existen sistemas que precisan de información orientada al control de datos y requieren flujos y procesos de control, así como los mecanismos que desencadenan su ejecución. Para que resulte adecuado el análisis de estos sistemas, se ha ampliado la notación de los diagramas de flujo de datos incorporando los siguientes elementos:

- **Proceso de control**: representa procesos que coordinan y sincronizan las actividades de otros procesos del diagrama de flujo de datos.
- **Flujo de control**: representa el flujo entre un proceso de control y otro proceso. El flujo de control que sale de un proceso de control activa al proceso que lo recibe y el que entre le informa de la situación de un proceso. A diferencia de los flujos tradicionales, que pueden considerarse como procesadores de datos porque reflejan el movimiento y transformación de los mismos, los flujos de control no representan datos con valores, sino que en cierto modo, se trata de eventos que activan los procesos (señales o interrupciones).

Descomposición o explosión por niveles

Los diagramas de flujo de datos han de representar el sistema de la forma más clara posible, por ello su construcción se basa en el principio de descomposición o explosión en distintos niveles de detalle.

La descomposición por niveles se realiza de arriba abajo (top-down), es decir, se comienza en el nivel más general y se termina en el más detallado, pasando por los niveles intermedios necesarios. De este modo se dispondrá de un conjunto de particiones del sistema que facilitarán su estudio y desarrollo.

La explosión de cada proceso de un DFD origina otro DFD y es necesario comprobar que se mantiene la consistencia de información entre ellos, es decir, que la información de entrada y de salida de un proceso cualquiera se corresponde con la información de entrada y de salida del diagrama de flujo de datos en el que se descompone.

En cualquiera de las explosiones puede aparecer un proceso que no necesite descomposición. A éste se le denomina Proceso primitivo y sólo se detalla en él su entrada y su salida, además de una descripción de lo que realiza. En la construcción hay que evitar en lo posible la descomposición desigual, es decir, que un nivel contenga un proceso primitivo, y otro que necesite ser particionado en uno o varios niveles más.

El modelo de procesos deberá contener:

- Un diagrama de contexto (Nivel 0).
- Un diagrama 0 (Nivel 1).
- Tantos diagramas 1, 2, 3, …, n como funciones haya en el diagrama 0 (Nivel 2).
- Tantos niveles intermedios como sea necesario.
- Varios DFD en el último nivel de detalle.

El diagrama de contexto tiene como objetivo delimitar el ámbito del sistema con el mundo exterior definiendo sus interfaces. En este diagrama se representa un único proceso que corresponde al sistema en estudio, un conjunto de entidades externas que representan la procedencia y destino de la información y un conjunto de flujos de datos que representan los caminos por los que fluye dicha información.

A continuación, este proceso se descompone en otro DFD, en el que se representan los procesos principales o subsistemas. Un subsistema es un conjunto de procesos cuyas funcionalidades tienen algo en común. Éstos deberán ser identificados en base a determinados criterios, como por ejemplo: funciones organizativas o administrativas propias del sistema, funciones homogéneas de los procesos, localización geográfica de los mismos, procesos que actualicen los mismos almacenes de datos, etc.

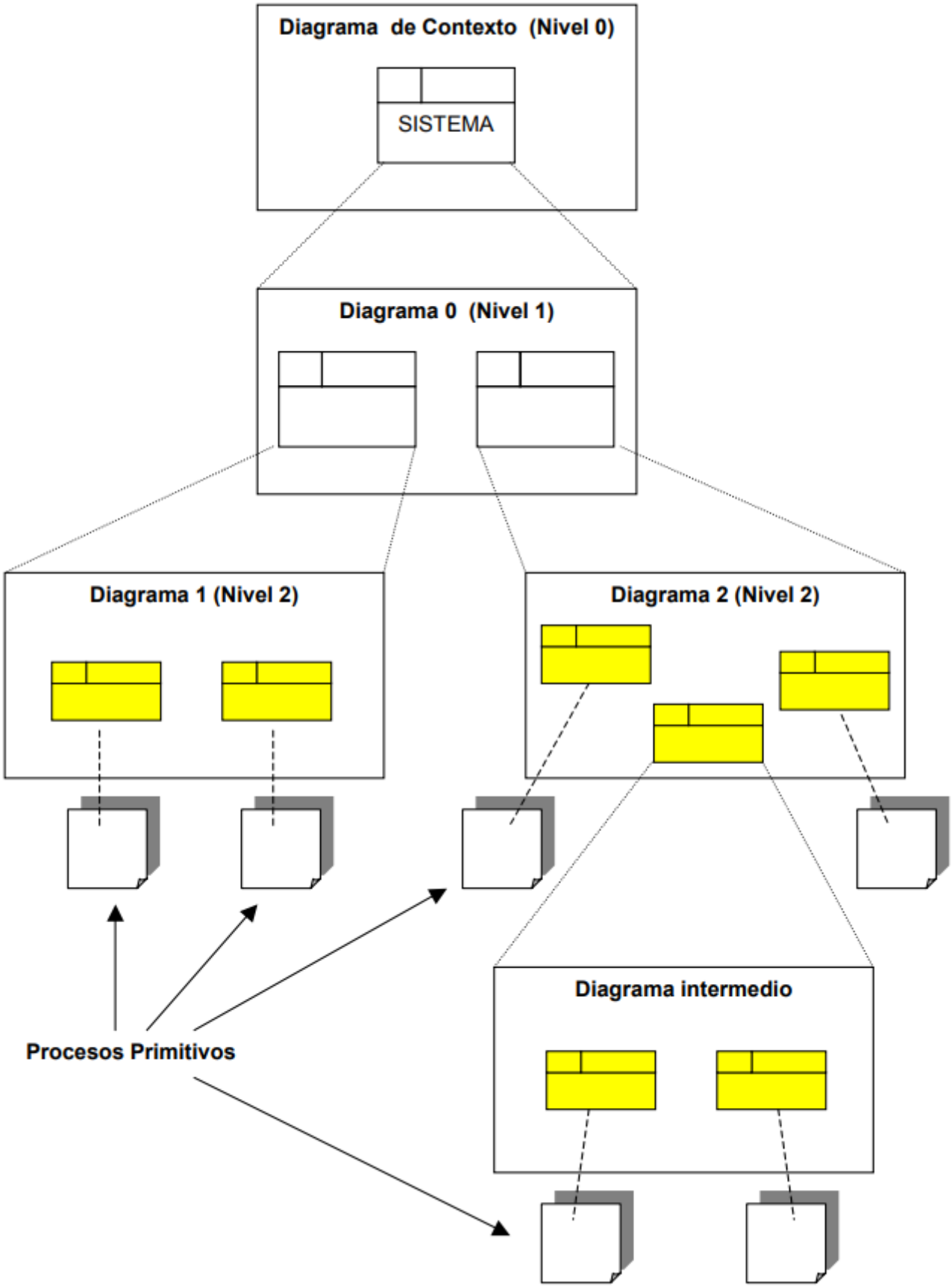
Cada uno de los procesos principales se descompone a su vez en otros que representan funciones más simples y se sigue descomponiendo hasta que los procesos estén suficientemente detallados y tengan una funcionalidad concreta, es decir, sean procesos primitivos.

Como resultado se obtiene un modelo de procesos del sistema de información que consta de un conjunto de diagramas de flujo de datos de diferentes niveles de abstracción, de modo que cada uno proporciona una visión más detallada de una parte definida en el nivel anterior.

Además de los diagramas de flujo de datos, el modelo de procesos incluye la especificación de los flujos de datos, de los almacenes de datos y la especificación detallada de los procesos que no precisan descomposición, es decir los procesos de último nivel o primitivos. En la especificación de un proceso primitivo se debe describir, de una manera más o menos formal, cómo se obtienen los flujos de datos de salida a partir de los flujos de datos de entrada y características propias del proceso.

Dependiendo del tipo de proceso se puede describir el procedimiento asociado utilizando un lenguaje estructurado o un pseudocódigo, apoyándose en tablas de decisión o árboles de decisión.

A continuación se muestra un ejemplo gráfico que representa la descomposición jerárquica de los diagramas de flujo de datos.

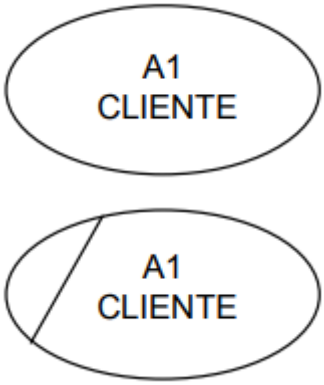


Notación

Entidad externa

Se representa mediante una elipse con un identificador y un nombre significativo en su interior.

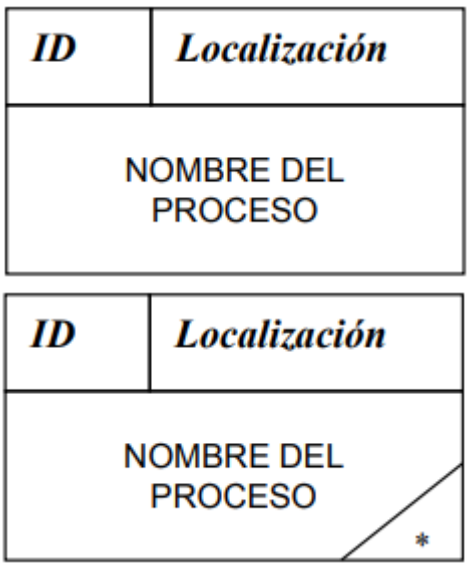
Si la entidad externa aparece varias veces en un mismo diagrama, se representa con una línea inclinada en el ángulo superior izquierdo.



Proceso

Se representa por un rectángulo subdividido en tres casillas donde se indica el nombre del proceso, un número identificativo y la localización.

Si el proceso es de último nivel, se representa con un asterisco en el ángulo inferior derecho separado con una línea inclinada.



El nombre del proceso debe ser lo más representativo posible. Normalmente estará constituido por un verbo más un sustantivo.

El número identificativo se representa en la parte superior izquierda e indica el nivel del DFD en que se está. Hay que resaltar que el número no indica orden de ejecución alguno entre los procesos ya que en un DFD no se representa una secuencia en el tratamiento de los datos. El número que identifica el proceso es único en el sistema y debe seguir el siguiente estándar de notación:

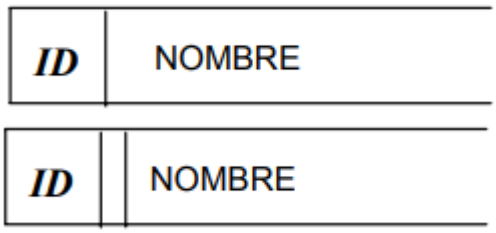
- El proceso del diagrama de contexto se numera como cero.
- Los procesos del siguiente nivel se enumeran desde 1 y de forma creciente hasta completar el número de procesos del diagrama.
- En los niveles inferiores se forma con el número del proceso en el que está incluido seguido de un número que lo identifica en ese contexto.

La localización expresa el nombre del proceso origen de la descomposición que se esté tratando.

Almacén de datos

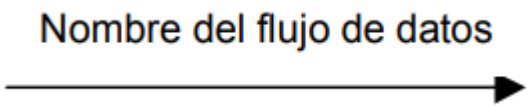
Se representa por dos líneas paralelas cerradas en un extremo y una línea vertical que las une. En la parte derecha se indica el nombre del almacén de datos y en la parte izquierda el identificador de dicho almacén en el DFD.

Si un almacén aparece repetido dentro de un DFD se puede representar con dos líneas verticales.

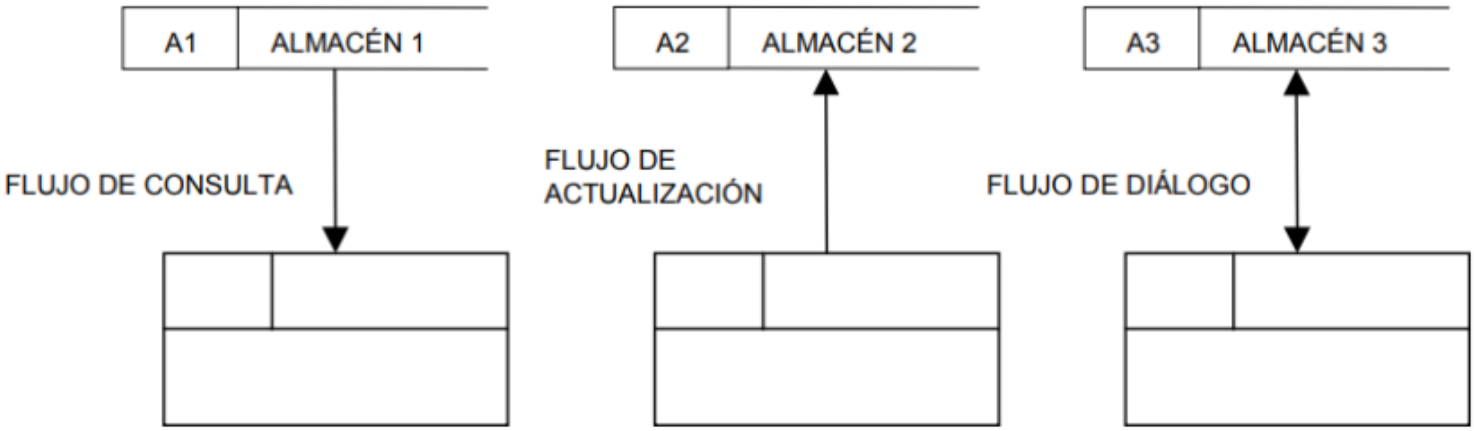


Flujo de datos

Se representa por una flecha que indica la dirección de los datos, y que se etiqueta con un nombre representativo.

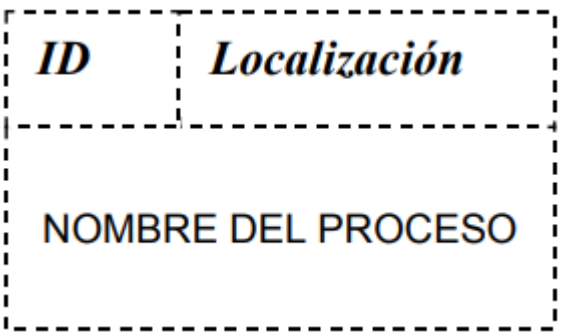


La representación de los flujos de datos entre procesos y almacenes es la siguiente:



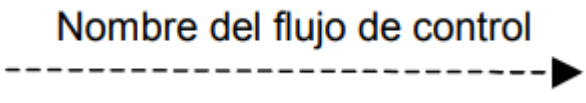
Proceso de control

Se representa por un rectángulo, con trazo discontinuo, subdividido en tres casillas donde se indica el nombre del proceso, un número identificativo y la localización.



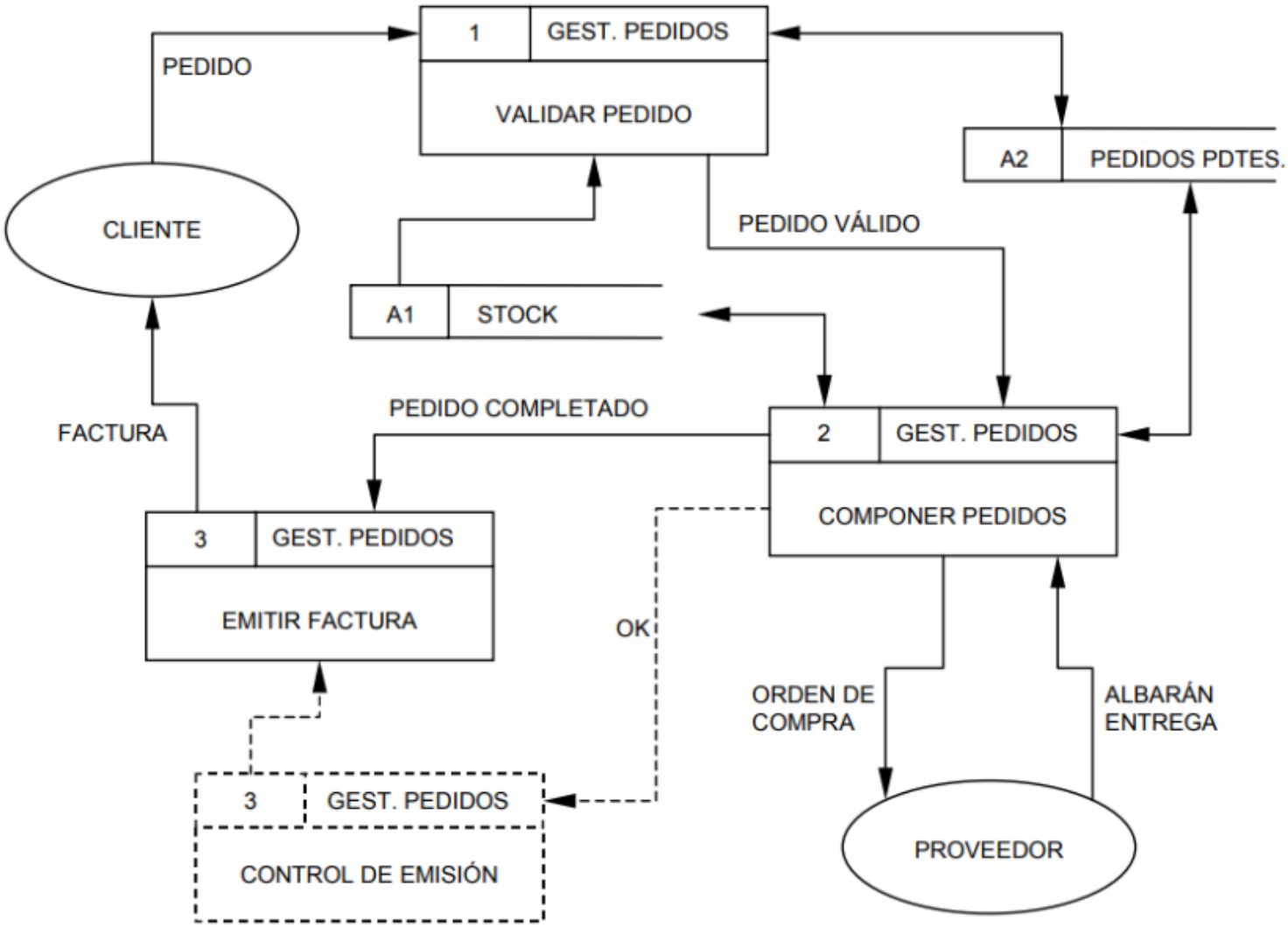
Flujo de control

Se representa por una flecha con trazo discontinuo que indica la dirección de flujo y que se etiqueta con un nombre representativo.



Ejemplo

La figura es un diagrama de flujos de un Sistema Gestor de Pedidos. En él están representados todos los elementos que pueden intervenir en un Diagrama de Flujo de Datos.



Consistencia de los diagramas de flujo de datos

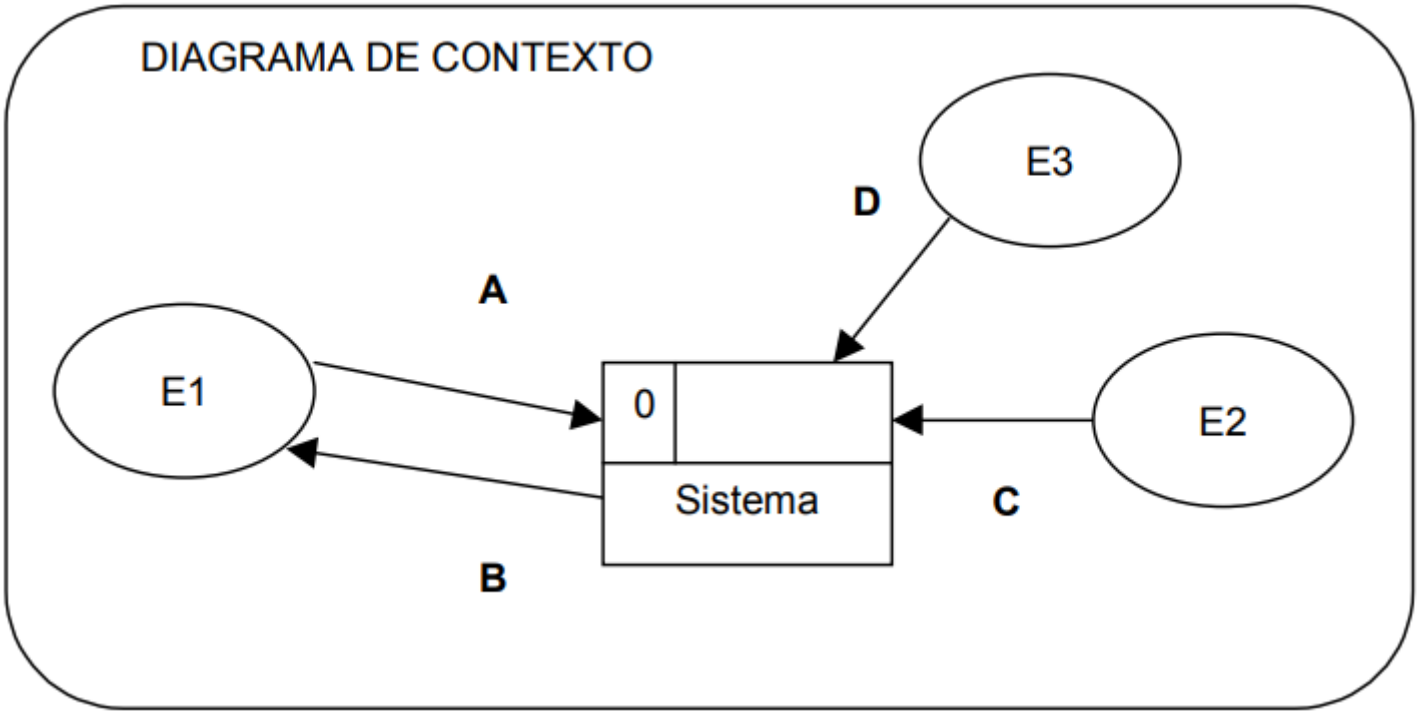
Una vez construidos los diagramas de flujo de datos que componen el modelo de procesos del sistema de información, es necesario comprobar y asegurar su validez. Para ello, se debe estudiar cada diagrama comprobando que es legible, de poca complejidad y si los nombres asignados a sus elementos ayudan a su comprensión sin ambigüedades.

Además, los diagramas deben ser consistentes. En los diagramas hay que comprobar que en un DFD resultado de una explosión:

- No falten flujos de datos de entrada o salida que acompañaban al proceso del nivel superior.
- No aparezca algún flujo que no estuviese ya asociado al proceso de nivel superior.
- Todos los elementos del DFD resultante deben estar conectados directa o indirectamente con los flujos del proceso origen.

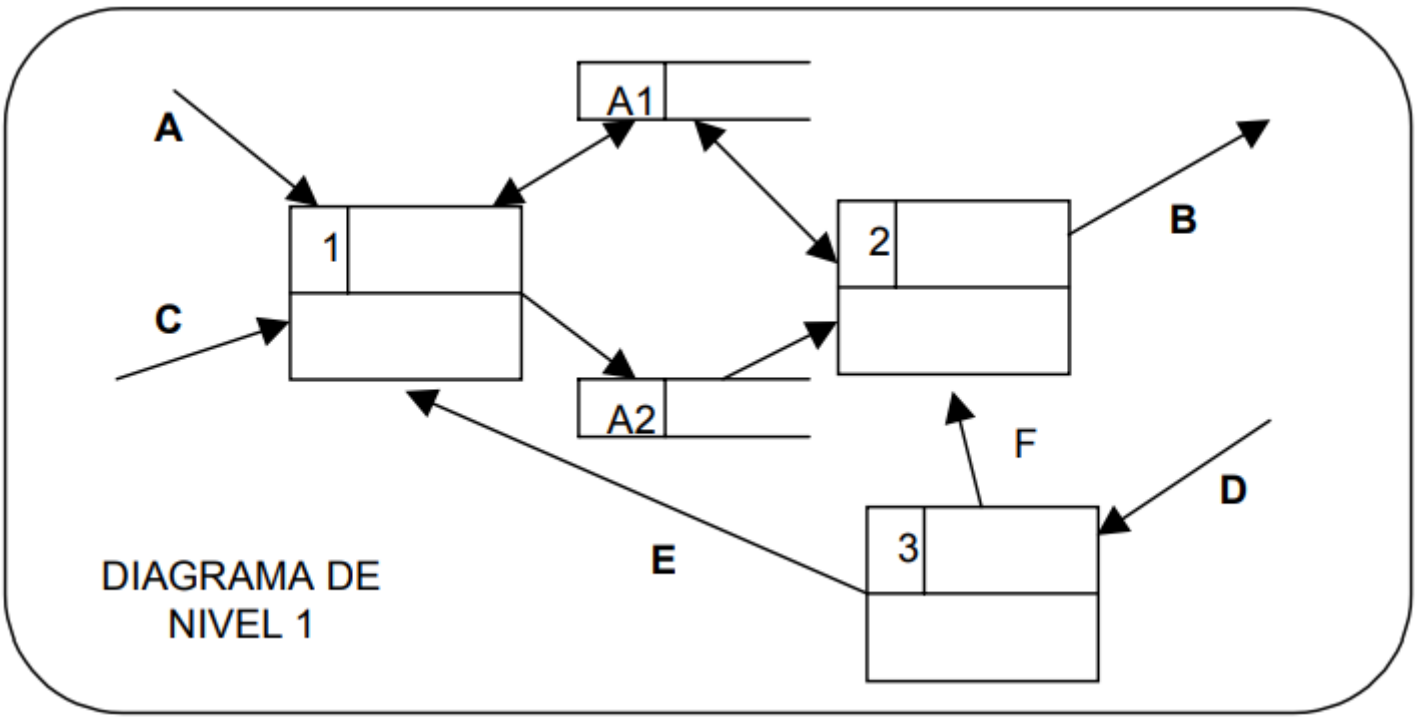
A continuación se incluyen ejemplos de la consistencia o inconsistencia de los diagramas de flujo de datos.

Sea el diagrama de contexto de la figura. Los flujos A, C y D, entran al sistema, y el flujo B sale de él.

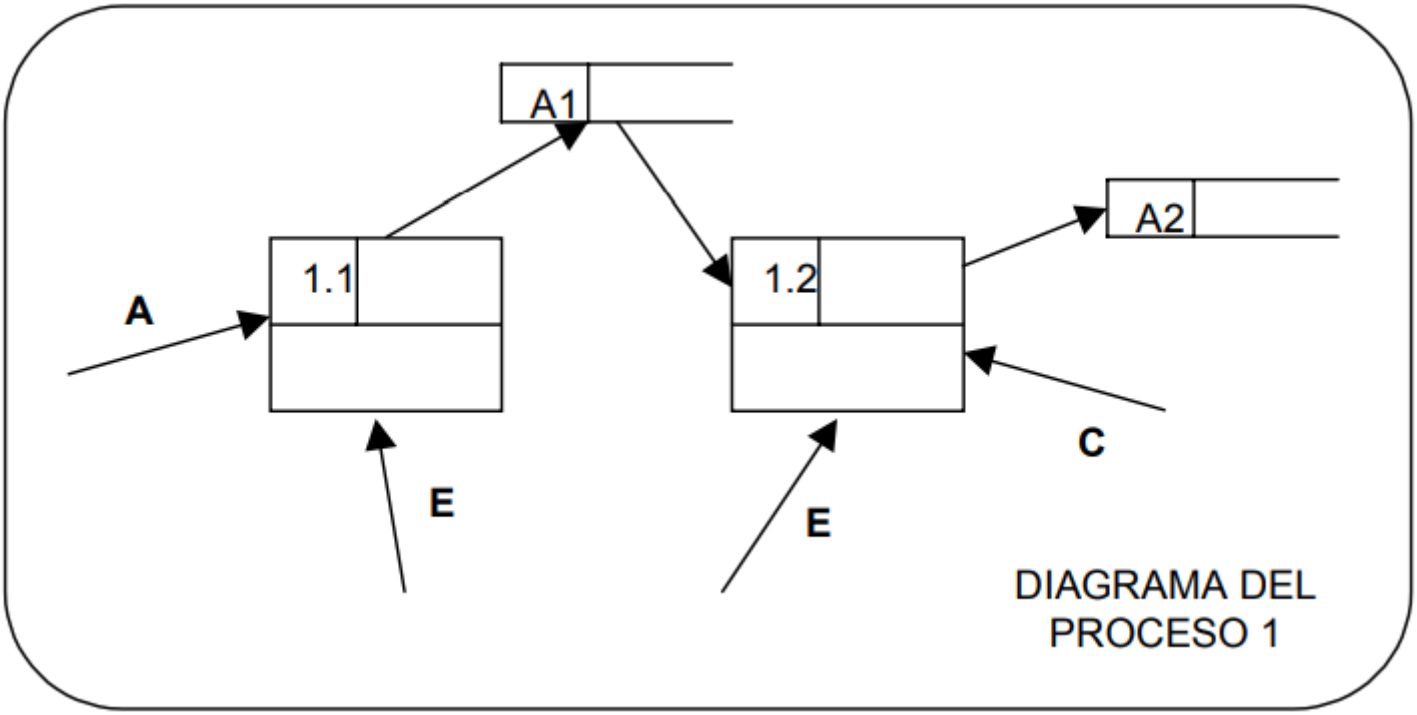


Ejemplo de consistencia de diagramas de flujo de datos

En la explosión del sistema en el diagrama de nivel 1, aparecen todos los flujos, y en su sentido correcto: A y C entran al subsistema o proceso 1, B sale del proceso 2, y D entra en el proceso 3. Se observa que el proceso 3, origina dos flujos de salida: E que va al proceso 1, y F al proceso 2.



La descomposición del proceso 1, muestra los flujos A, C y E correctamente, como entradas a las funciones del diagrama.

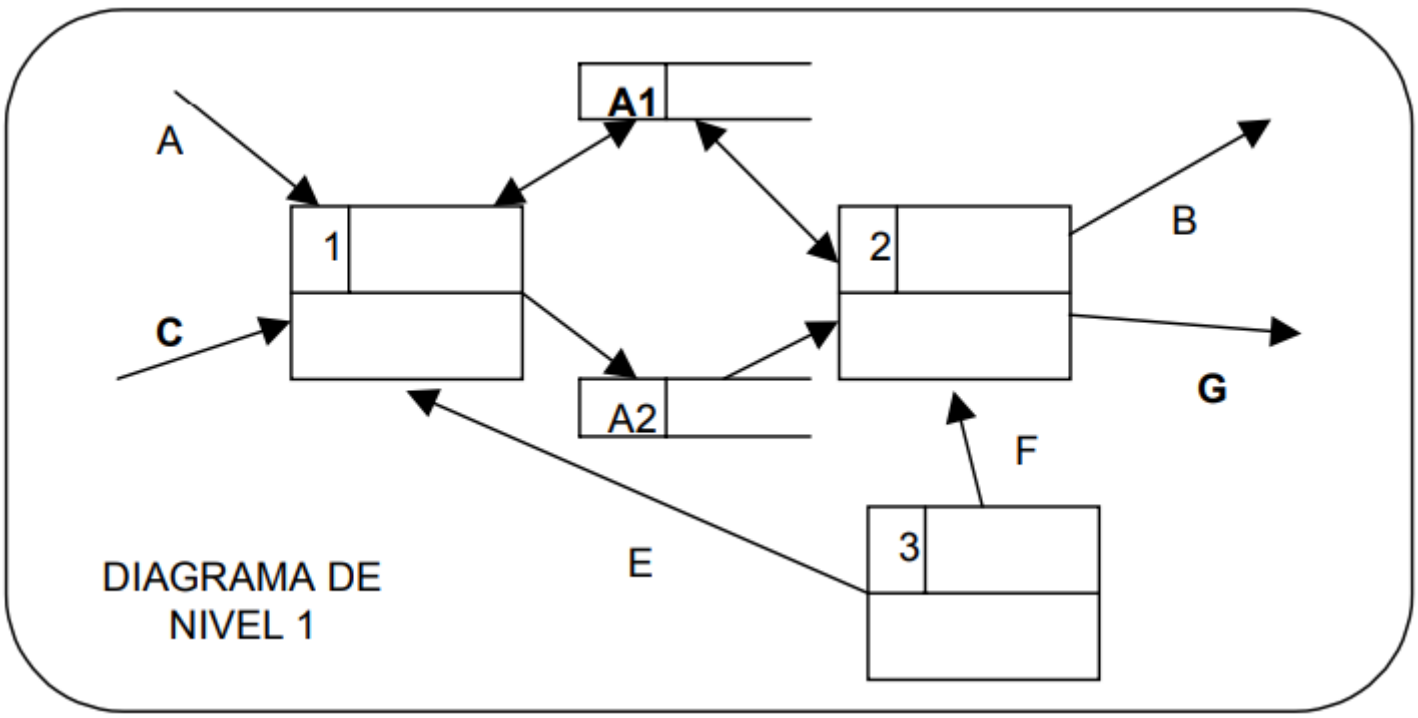


Los demás flujos están enlazados con los almacenes A1 y A2 del mismo modo que en el diagrama anterior.

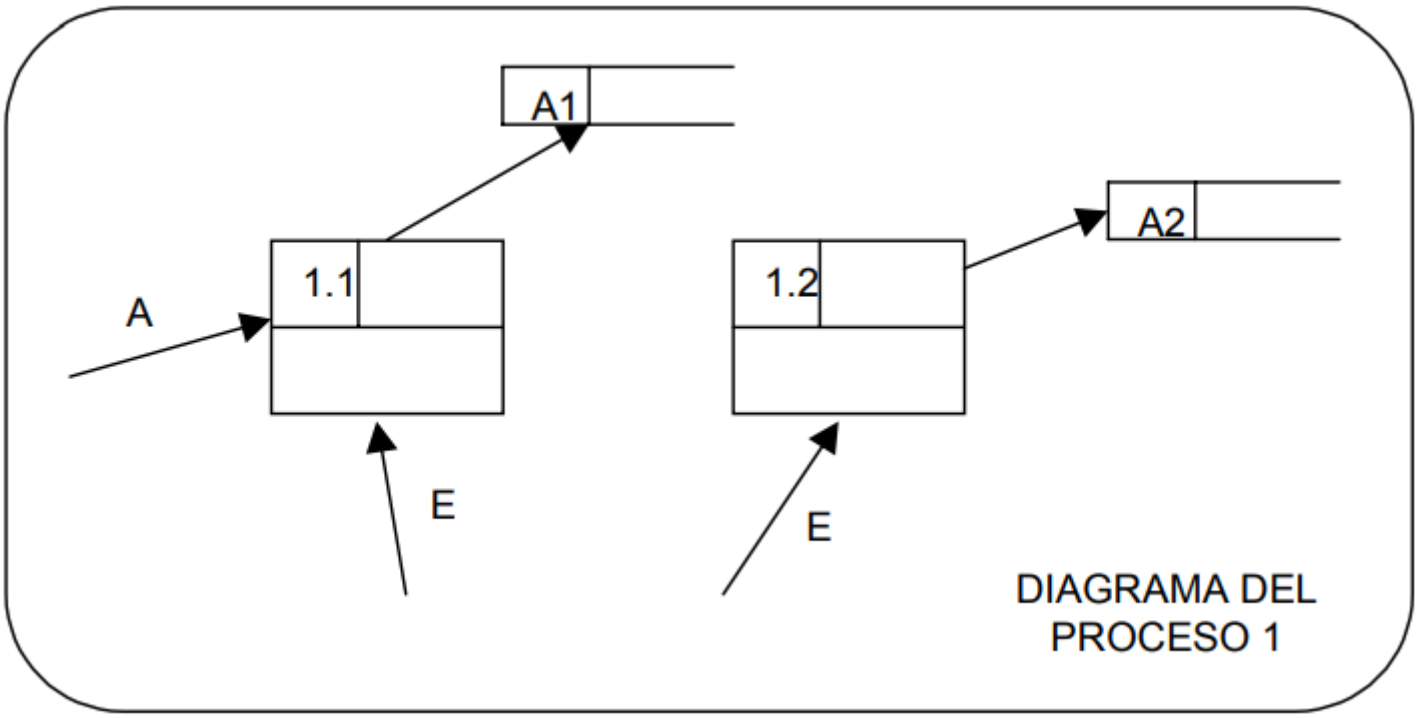
Ejemplo de inconsistencia de diagramas de flujo de datos

Partiendo del mismo diagrama de contexto utilizado en el anterior ejemplo, los flujos A, C y D, que entran al sistema, y el flujo B, que sale de él, deben aparecer en la primera descomposición, el diagrama de nivel 1. En la figura se aprecia que falta el flujo D, y hay un flujo G que o bien falta en el nivel anterior, sobra en este.

Por otro lado, en el proceso 3 no entra ningún flujo, no es posible por tanto que transforme datos saliendo los flujos E y F y además está desconectado del nivel anterior.



En el siguiente paso, la inconsistencia más clara es la falta del flujo C, que entra al proceso 1, y sin embargo no aparece en su explosión.



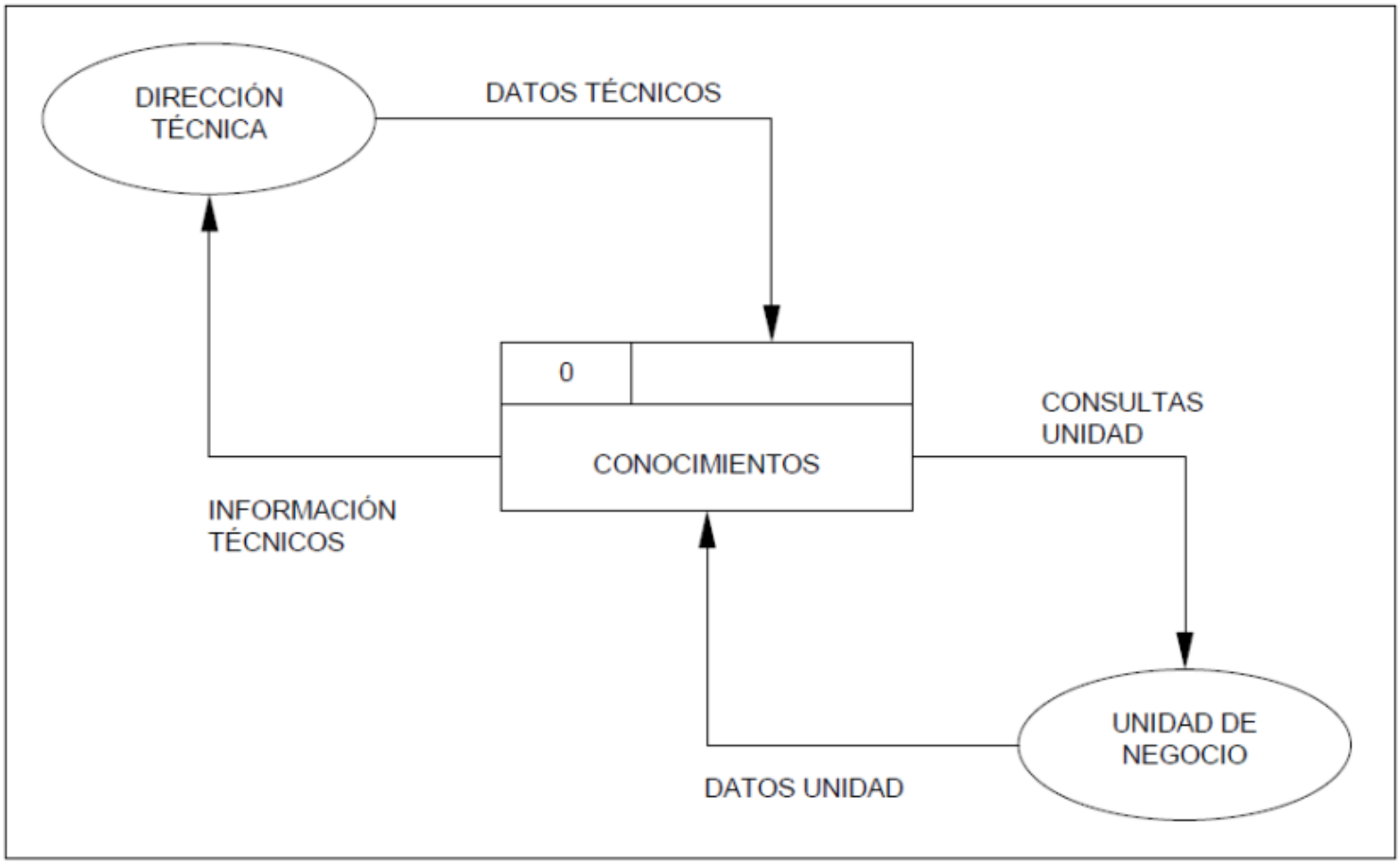
Además, hay otra inconsistencia respecto al almacén A1: en el diagrama del nivel anterior, el proceso 1 se conectaba con un flujo de entrada-salida a este almacén, cosa que no se refleja en el diagrama de este proceso, en el que sólo aparece uno de entrada.

Ejemplo de construcción

El caso en estudio es un modelo de procesos de un sistema de información de Conocimientos de técnicos. Según estos conocimientos, los técnicos podrán ser asignados a determinados proyectos de la organización.

El sistema recogerá la información referente a los técnicos, procedente de la Dirección técnica de la organización y de los proyectos, procedente de cualquier sección o Unidad de Negocio en las que está dividida dicha organización.

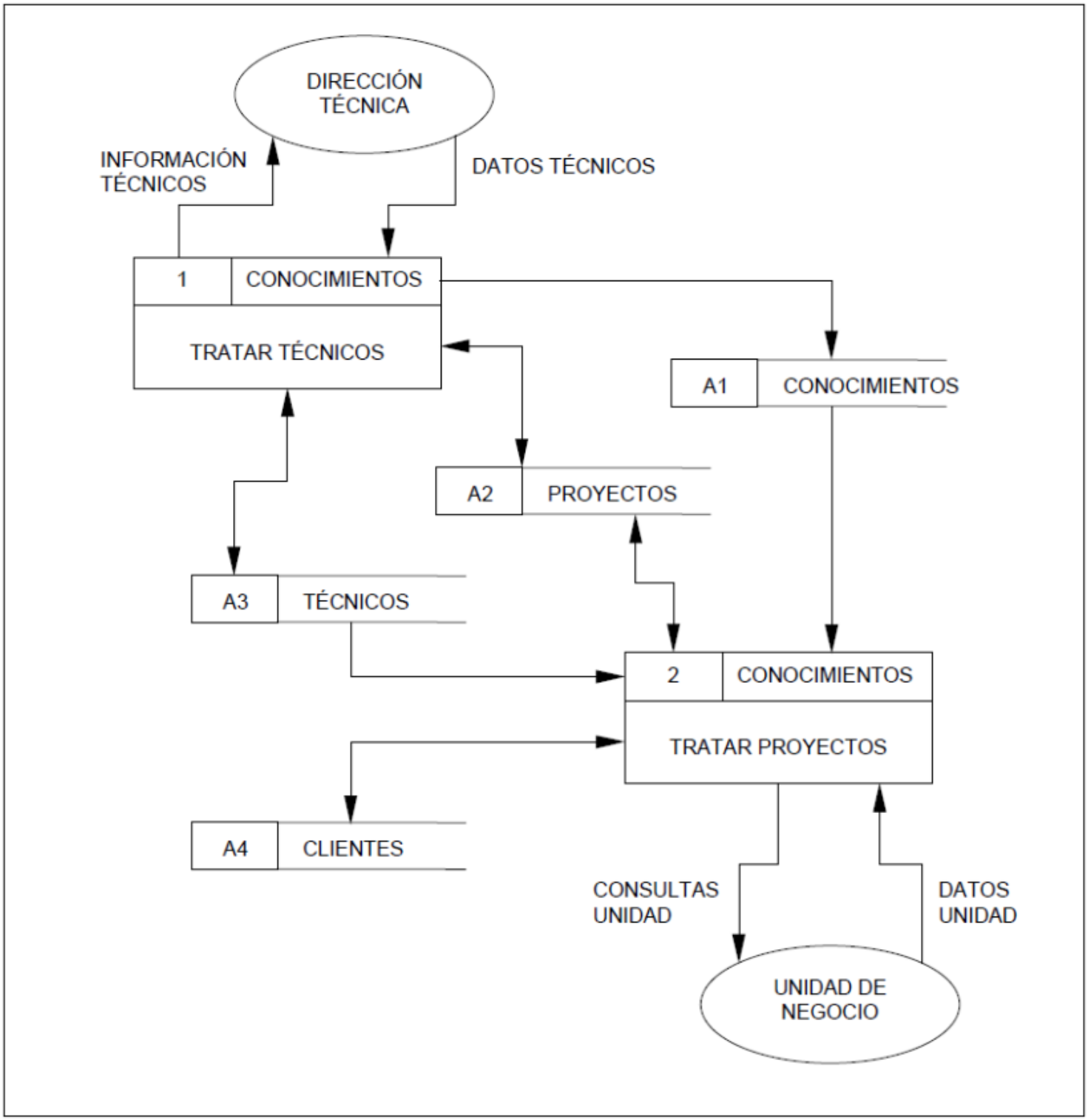
Las entidades externas son pues Dirección Técnica y Unidad de Negocio, que introducen los datos al sistema y hacen peticiones de consultas e informes sobre los técnicos y sus conocimientos. El diagrama de contexto será el siguiente:



Los flujos de entrada son: Datos Técnicos, con datos de los técnicos introducidos por la Dirección Técnica, así como posibles peticiones de información sobre ellos; y Datos Unidad, que proviene de la Unidad de Negocio, conteniendo datos referentes a la unidad, de proyectos y clientes, así como posibles peticiones de consultas sobre los mismos.

Los flujos de salida son: Información Técnicos, que contendrá datos de técnicos, de consulta o informes, para uso de la Dirección Técnica y Consultas Unidad, con datos requeridos por la Unidad de Negocio.

El sistema de Conocimientos se descompone en el diagrama de nivel 1, conteniendo do subsistemas. El subsistema 1 recogerá las funciones a realizar con los datos de los técnicos de la organización (actualizaciones, consultas, informes, etc), por lo que se denomina Tratar Técnicos. El subsistema 2 contendrá las funciones asociadas al procesamiento de datos de proyectos, por lo que se le da el nombre Tratar Proyectos.



En el diagrama se encuentran cuatro almacenes, tres de los cuales son accedidos por funciones de los dos subsistemas: A1 Conocimientos, A2 Proyectos y A3 Técnicos. El cuarto, A4 Clientes, sólo es accedido por el subsistema Tratar Proyectos.

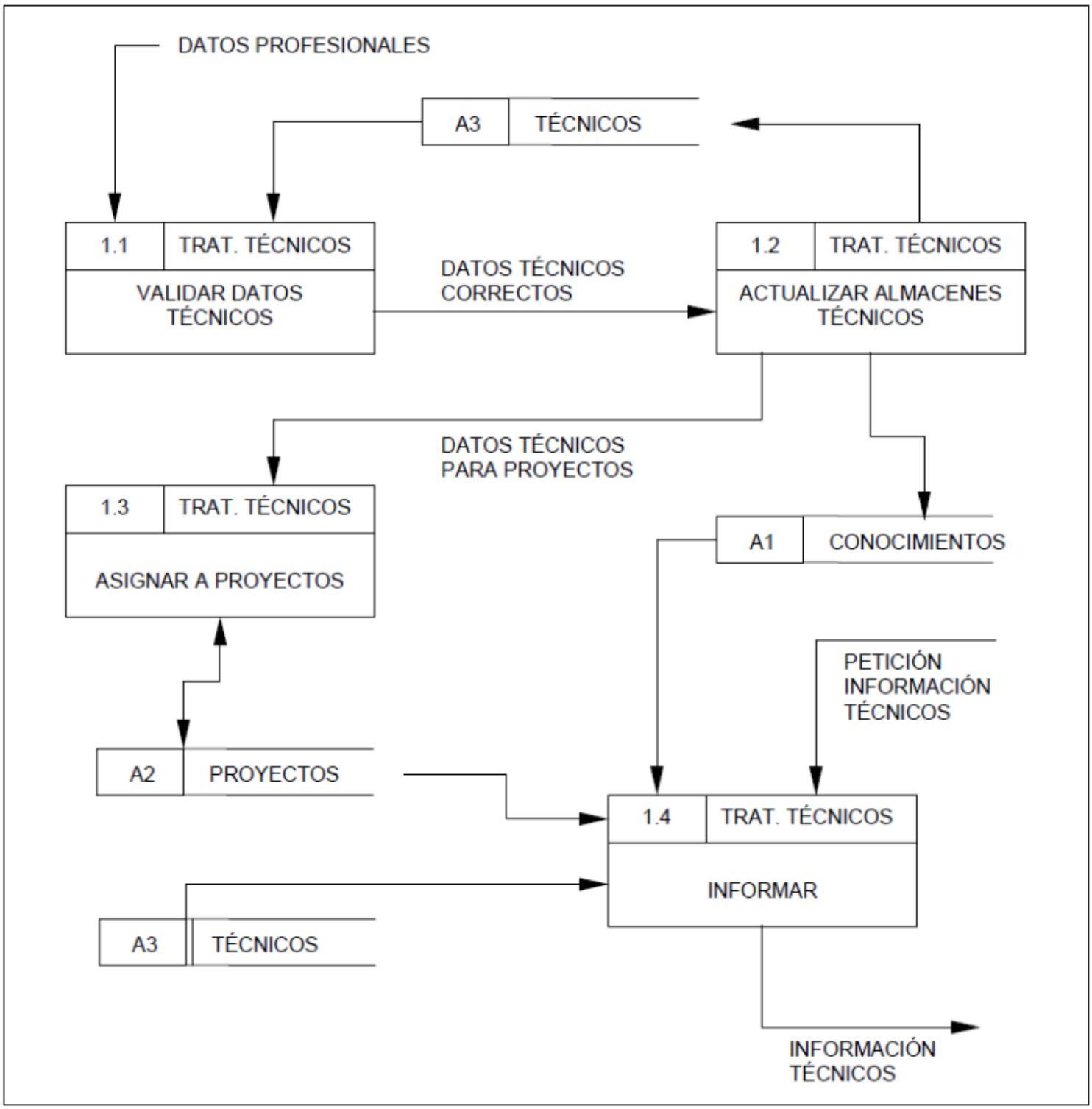
Los flujos sin nombre indican que hay entrada y/o salida de todos los datos del almacén. En este diagrama siguen apareciendo las entidades externas para la mayor comprensión del mismo.

A partir de ahora, se centrará el ejemplo en la descomposición del subsistema 1 Tratar Técnicos, hasta llegar a su nivel más detallado.

En el diagrama resultado de la explosión de Tratar Técnicos, se incluyen cuatro procesos o funciones para el tratamiento completo de éstos.

El flujo de entrada Datos Técnicos se compone tanto de los datos profesionales de los técnicos, como de datos de peticiones de información sobre los mismos, por lo cual se ha

dividido en dos: Datos Profesionales, que es entrada del proceso 1.1 Validad datos Técnicos y Peticiones Información Técnicos, que entra en la función 1.4 Informar.

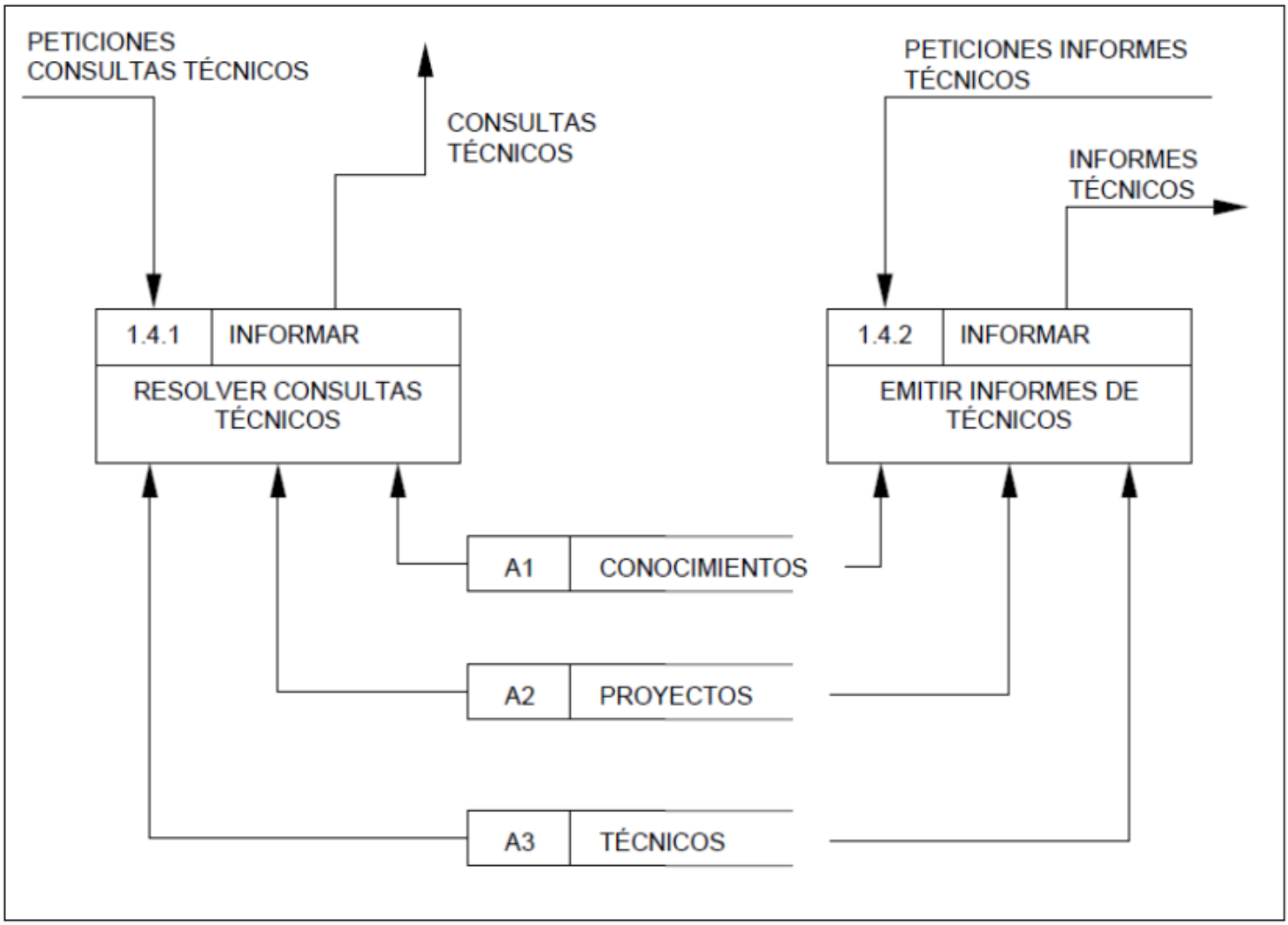


Para la validación, el proceso 1.1 Validar Datos Técnicos obtiene información del almacén A3 Técnicos y genera una salida, el flujo Datos Técnicos Correctos, que lleva los datos válidos a la función 1.2 Actualizar Almacenes Técnicos. Esta función se encarga de actualizar los almacenes A3 Técnicos y A1 Conocimientos, pero también emite un flujo al proceso 1.3 Asignar a Proyectos. Éste se encarga de hacer asignaciones de técnicos en el almacén A2 Proyectos.

La función 1.4 Informar, recibe las peticiones de información sobre técnicos, las procesa utilizando los almacenes necesarios y genera el flujo Información Técnicos que irá a la entidad Dirección Técnica, según muestran los primeros diagramas.

Obsérvese que para mayor claridad no se ha incluido ya ninguna entidad externa, y además, se ha repetido el almacén A3 Técnicos, evitando que el cruce de flujos oscurezca la lectura del diagrama.

En este momento, todos los procesos se consideran primitivos, excepto el proceso 1.4 Informar, del que se obtiene su descomposición. Sus funciones han de obtener Informes Técnicos y Consultas Técnicos, flujos que componen Información Técnicos que aparecía en el nivel anterior.



Por otro lado, también aparece dividido el flujo de entrada Peticiones Información Técnicos, diferenciando la entrada al proceso de consultas o al de emisión de informes.

Por último, se puede apreciar que los almacenes son los mismos que se conectaban con el proceso en el nivel anterior y los flujos son de entrada a las funciones.

(Nota.- Esta notación es la más habitual, pero MÉTRICA Versión 3 no exige su utilización).

Diagrama de Estructura

El objetivo de este diagrama es representar la estructura modular del sistema o de un componente del mismo y definir los parámetros de entrada y salida de cada uno de los módulos.

Para su realización se partirá del modelo de procesos obtenido como resultado de la aplicación de la técnica de diagrama de flujo de datos (DFD).

Descripción

Un diagrama de estructura se representa en forma de árbol con los siguientes elementos:

- **Módulo**: división del software clara y manejable con interfaces modulares perfectamente definidas. Un módulo puede representar un programa, subprograma o rutina dependiendo del lenguaje a utilizar. Admite parámetros de llamada y retorno. En el diseño de alto nivel hay que ver un módulo como una *caja negra*, donde se contemplan exclusivamente sus entradas y sus salidas y no los detalles de la lógica interna del módulo. Para que se reduzca la complejidad del cambio ante una determinada modificación, es necesario que los módulos cumplan las siguientes condiciones:
  - Que sean de pequeño tamaño.
  - Que sean independientes entre sí.
  - Que realicen una función clara y sencilla.
- **Conexión**: representa una llamada de un módulo a otro.
- **Parámetro**: información que se intercambia entre los módulos. Pueden ser de dos tipos en función de la clase de información a procesar:
  - *Control*: son valores de condición que afectan a la lógica de los módulos llamados. Sincronizan la operativa de los módulos.
  - *Datos*: información compartida entre módulos y que es procesada en los módulos llamados.

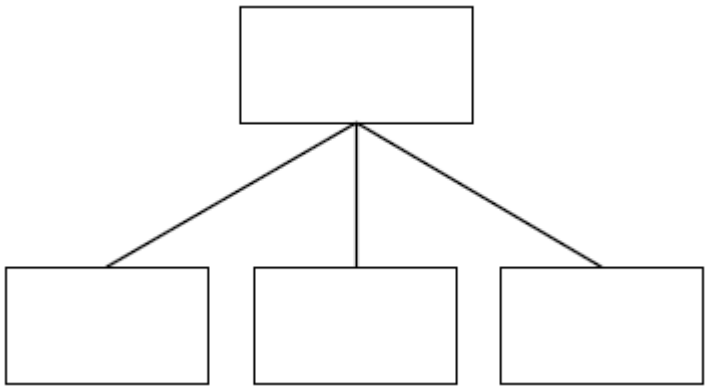
Otros componentes que se pueden representar en el diagrama de estructura son:

- **Módulo predefinido**: es aquel módulo que está disponible en la biblioteca del sistema o de la propia aplicación, y por tanto no es necesario codificarlo.
- **Almacén de datos**: es la representación física del lugar donde están almacenados los datos del sistema.
- **Dispositivo físico**: es cualquier dispositivo por el cual se puede recibir o enviar información que necesite el sistema.

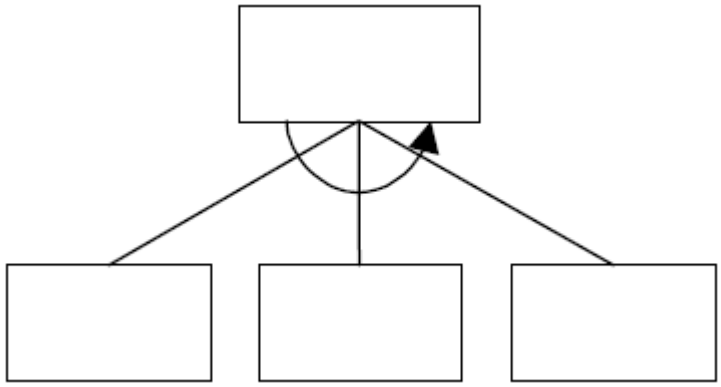
Estructuras del diagrama

Existen ciertas representaciones gráficas que permiten mostrar la secuencia de las llamadas entre módulos. Las posibles estructuras son:

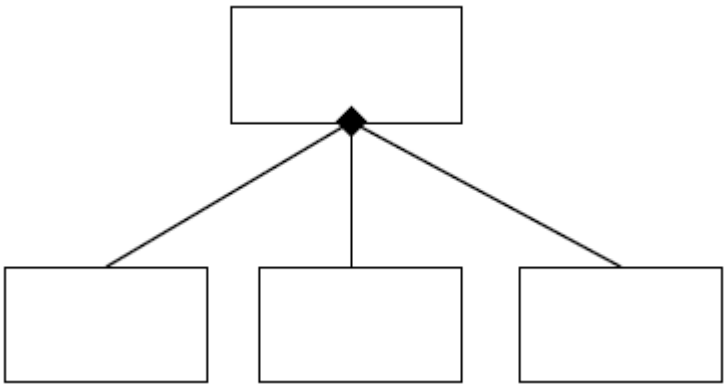
- **Secuencial**: un módulo llama a otros módulos una sola vez y, se ejecutan de izquierda a derecha y de arriba a abajo.



- **Repetitiva**: cada uno de los módulos inferiores se ejecuta varias veces mientras se cumpla una condición.



- **Alternativa**: cuando el módulo superior, en función de una decisión, llama a un módulo u otro nivel de los de nivel inferior.



Principios del diseño estructurado

El diagrama de estructura se basa en tres principios fundamentales:

- La descomposición de los módulos, de manera que los módulos que realizan múltiples funciones se descompongan en otros que sólo realicen una. Los objetivos que se persiguen con la descomposición son:
  - Reducir el tamaño del módulo.
  - Hacer el sistema más fácil de entender y modificar y por lo tanto facilitar el mantenimiento del mismo.
  - Minimizar la duplicidad de código.
  - Crear módulos útiles.
- La jerarquía entre los módulos, de forma que los módulos de niveles superiores coordinen a los de niveles inferiores. Al dividir los módulos jerárquicamente, es posible controlar el número de módulos que interactúan con cualquiera de los otros.
- La independencia de los módulos, de manera que cada módulo se ve como una caja negra, y únicamente es importante su función y su apariencia externa, y no los detalles de su construcción.

Estrategias de diseño

Dependiendo de la estructura inicial del diagrama de flujo de datos sobre el que se va a realizar el diseño, existen dos estrategias para obtener el diagrama de estructura. El uso de una de las dos estrategias no implica que la otra no se utilice, eso dependerá de las características de los procesos representados en DFD. Estas estrategias son:

- Análisis de transformación.
- Análisis de transacción.

Análisis de Transformación

El análisis de transformación es un conjunto de pasos que permiten obtener, a partir de un DFD con características de transformación, la estructura del diseño de alto nivel del

sistema. Un DFD con características de transformación es aquél en el que se pueden distinguir:

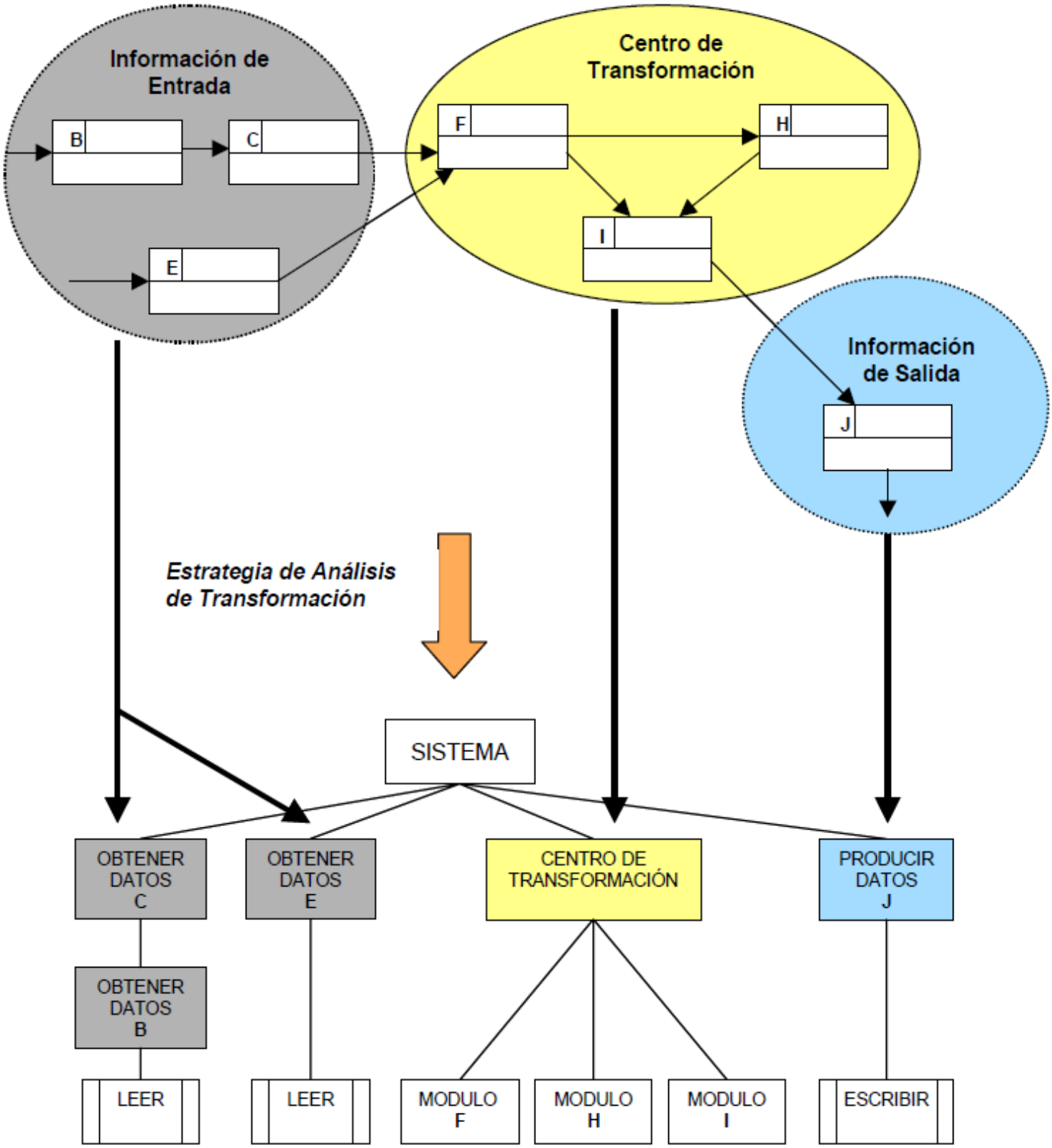
- Flujo de llegada o entrada.
- Flujo de transformación o centro de transformación que contiene los procesos esenciales del sistema y es independiente de las características particulares de la entrada y la salida.
- Flujo de salida.

Los datos que necesita el sistema se recogen por los módulos que se encuentren en las ramas de la izquierda, de modo que los datos que se intercambian en esa rama serán ascendentes. En las ramas centrales habrá movimiento de información compartida, tanto ascendente como descendente. En las ramas de la derecha, la información será de salida y, por lo tanto, descendente.

Los pasos a realizar en el análisis de transformación son:

1. Identificar el centro de transformación. Para ello será necesario delimitar los flujos de llegada y salida de la parte del DFD que contiene las funciones esenciales del sistema.
2. Realizar el “primer nivel de factorización” o descomposición del diagrama de estructura. Habrá que identificar tres módulos subordinados a un módulo de control del sistema:
    - Módulo controlador del proceso de información de entrada.
    - Módulo controlador del centro de transformación.
    - Módulo controlador del proceso de la información de salida.
3. Elaborar el “segundo nivel de factorización”. Se transforma cada proceso del DFD en un módulo del diagrama de estructura.
4. Revisar la estructura del sistema utilizando medidas y guías de diseño.

A continuación se muestra un gráfico explicativo de dicha estrategia de diseño:



Análisis de Transacción

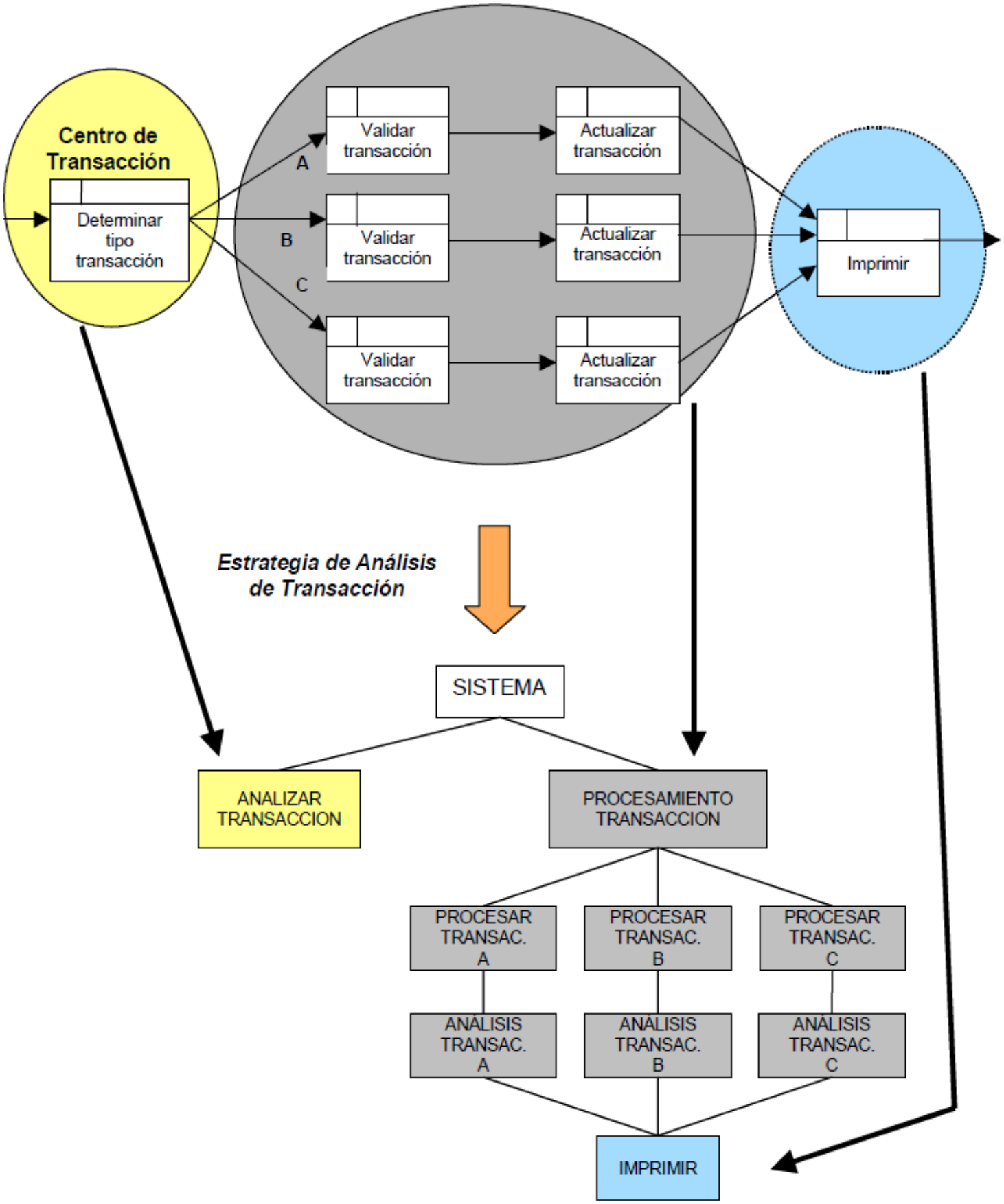
El análisis de transacción se aplica cuando en un DFD existe un proceso que en función del flujo de llegada, determina la elección de uno o más flujos de información.

Se denomina *centro de transacción* al proceso desde el que parten los posibles caminos de información. Los pasos a realizar en el análisis de transacción son:

1. Identificar el centro de transacción. Se delimita la parte del DFD en la que a partir de un camino de llegada se establecen varios caminos de acción.
2. Transformar el DFD en la estructura adecuada al proceso de transacciones. El flujo de transacciones se convierte en una estructura de programa con una bifurcación de entrada y una de salida.

3. Factorizar la estructura de cada camino de acción. Cada camino se convierte en una estructura que se corresponde con las características específicas del flujo (de transacción o de transformación).
4. Refinar la estructura del sistema utilizando medidas y guías de diseño.

A continuación se muestra un gráfico explicativo de dicha estrategia de diseño:



Evaluación del diseño

Una vez que hayan sido elaborados los diagramas de estructura, habrá que evaluar el diseño estudiando distintos criterios y medidas. Se utilizan dos métricas que miden la calidad estructural de un diseño:

- Acoplamiento.
- Cohesión.

El ***acoplamiento*** se puede definir como el grado de interdependencia existente entre los módulos, por tanto, depende del número de parámetros que se intercambian. El objetivo es que el acoplamiento sea el mínimo posible, es decir, conseguir que los módulos sean lo más independientes entre sí.

Es deseable un bajo acoplamiento, debido a que cuantas menos conexiones existan entre dos módulos, menor será la posibilidad de que aparezcan efectos colaterales al modificar uno de ellos. Además, se mejora el mantenimiento, porque al cambiar un módulo por otro, hay menos riesgo de actualizar la lógica interna de los módulos asociados. Los diferentes grados de acoplamiento son:

- *De datos*: los módulos se comunican mediante parámetros que constituyen elementos de datos simples.
- *De marca*: es un caso particular del acoplamiento de datos, donde la comunicación entre módulos es a través de estructuras de datos.
- *De control*: aparece cuando uno o varios de los parámetros de comunicación son de control, es decir variables que controlan las decisiones de los módulos subordinados o superiores.
- *Externo*: los módulos están ligados a componentes externos (dispositivos E/S, protocolos de comunicaciones, etc).
- *Común*: varios módulos hacen referencia a un área común de datos. Los módulos asociados al área común de datos pueden modificar los valores de los elementos de datos o estructuras de datos que se incluyen en dicha área.
- *De contenido*: ocurre cuando un módulo cualquiera accede o hace uso de los datos de una parte de otro módulo.

La ***cohesión*** es una medida de la relación funcional de los elementos de un módulo, es decir, la sentencia o grupo de sentencias que lo componen, las llamadas a otros módulos o las definiciones de los datos. Un módulo con alta cohesión realiza una tarea concreta y sencilla.

El objetivo es intentar obtener módulos con una cohesión alta o media. Los distintos niveles de cohesión, de mayor a menor, son:

- *Funcional*: todos los elementos que componen el módulo están relacionados en el desarrollo de una única función.
- *Secuencial*: un módulo empaqueta en secuencia varios módulos con cohesión funcional.
- *De comunicación*: todos los elementos de procesamiento utilizan los mismos datos de entrada y de salida.
- *Procedimental*: todos los elementos de procesamiento de un módulo están relacionados y deben ejecutarse en un orden determinado. En este tipo existe paso de controles.
- *Temporal*: un módulo contiene tareas relacionadas por el hecho de que todas deben realizarse en el mismo intervalo de tiempo.
- *Lógica*: un módulo realiza tareas relacionadas de forma lógica (por ejemplo un módulo que produce todas las salidas independientemente del tipo).
- *Casual*: un modulo realiza un conjunto de tareas que tienen poca o ninguna relación entre sí.

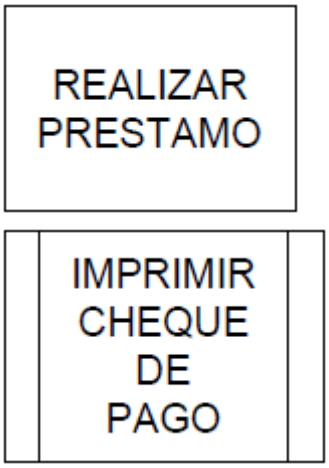
Un buen diseño debe ir orientado a conseguir que los módulos realicen una función sencilla e independiente de las demás (máxima cohesión), y que la dependencia con otros módulos sea mínima (acoplamiento mínimo), lo cual facilita el mantenimiento del diseño.

Notación

Módulo

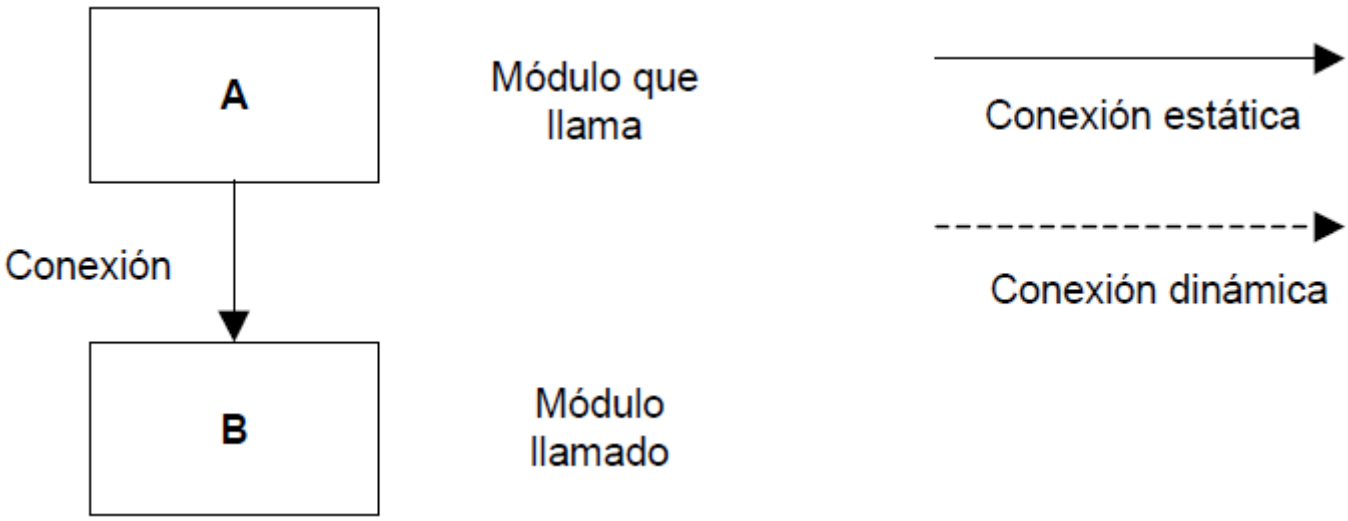
Se representa mediante un rectángulo con su nombre en el interior.

Un módulo predefinido se representa añadiendo dos líneas verticales y paralelas en el interior del rectángulo.



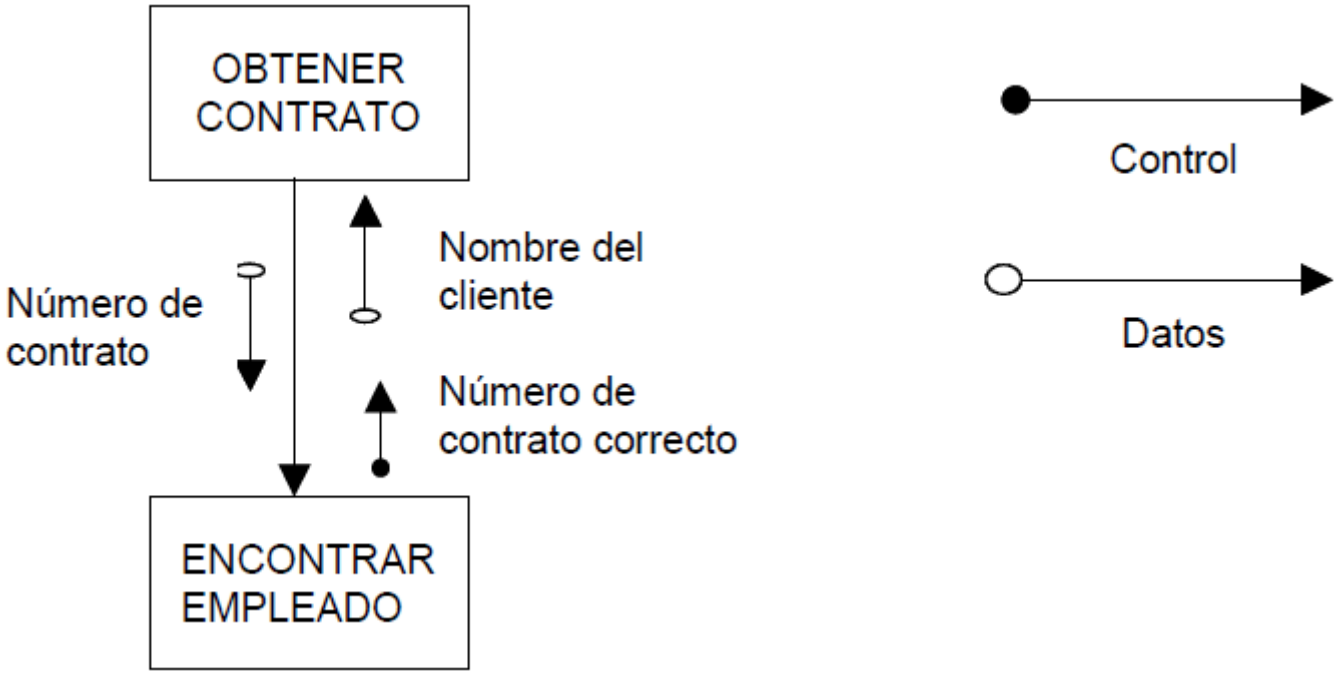
Conexión

Se representa mediante una línea terminada en punta de flecha cuya dirección indica el módulo llamado. Para llamadas a módulos estáticos se utiliza trazo continuo y para llamadas a módulos dinámicos trazo discontinuo.

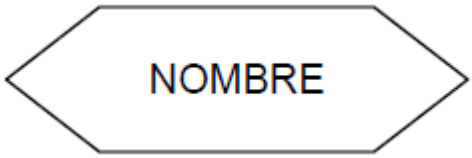


Parámetros

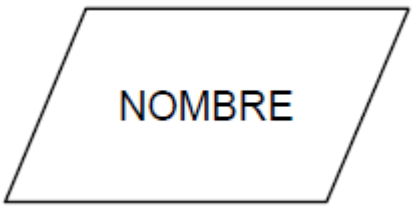
La representación varía según su tipo: control (flags) o datos.



Almacén de datos



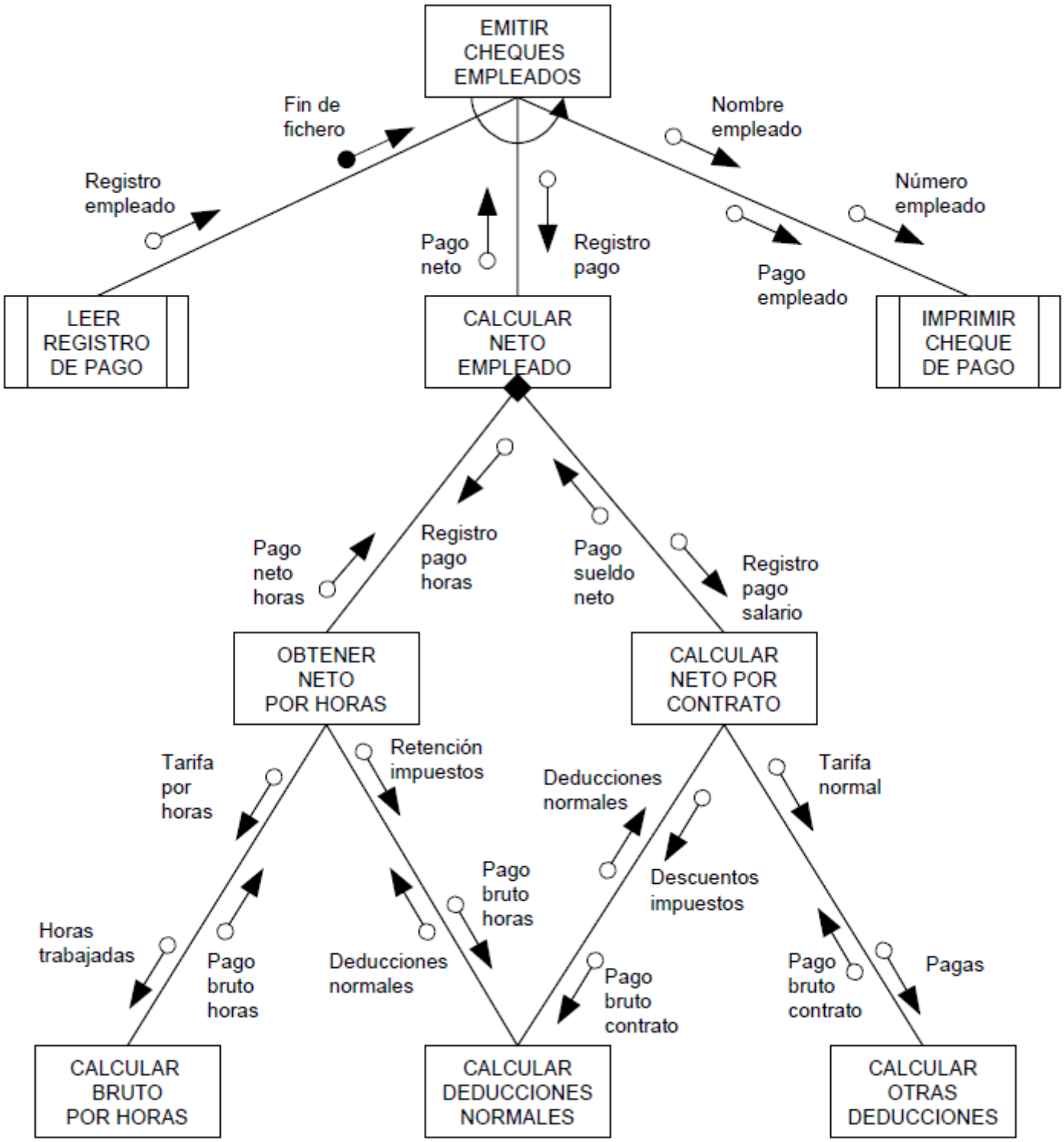
Dispositivo físico



(Nota.- Esta notación es la más habitual, pero MÉTRICA Versión 3 no exige su utilización).

Ejemplo

En el siguiente ejemplo muestra un proceso de emisión de cheques para el pago de nóminas de los empleados de una empres. En él se diferencian los cálculos relativos a los trabajadores empleados por horas y los que poseen contrato. La lectura del fichero de empleados y la impresión de los cheques son módulos ya disponibles en las librerías del sistema, es decir, módulos predefinidos.



Diccionario de Datos

Documentación del sistema

Hasta el momento hemos descrito las técnicas utilizadas en el desarrollo de sistemas, pero el desarrollo de modelos no queremos hacerlo sobre hojas sueltas, con el peligro de extraviarlas, y tener dificultad en mantenerlo. Por el contrario, necesitamos organizar el seguimiento de los modelos, principalmente por dos razones:

- Dar significado a los componentes del modelo, ayudando a gestionar la complejidad del sistema.
- Soportar el mantenimiento, ya que cualquier trabajo puede pasar de una persona a otra.

A esta forma de seguimiento organizado del trabajo producido durante el análisis y diseño del sistema se llama documentación del sistema.

La documentación del sistema es tanto una herramienta de comunicación, como de comunicación porque contiene un almacén de todo el trabajo hecho cada día y lo pone a disposición de todas las personas que trabajan en un proyecto grande. También es una herramienta de dirección, porque asegura una alta eficiencia, ya que todas las personas tienen acceso a lo último realizado. Dado que un proyecto se divide en fases, se establece la documentación que se debe aportar en cada fase, lo que ayuda a conocer la situación en cada momento del proyecto.

Para que sea útil, la documentación debe ser estructural y fácil de usar. La documentación en un primer momento se divide en informes de proyecto y una descripción del sistema.

Informes del proyecto

Los informes del proyecto incluyen la información requerida por la dirección del proyecto. Los informes incluyen un resumen de la fase actual, unas recomendaciones para la siguiente fase y un plan con los recursos propuestos.

La información específica de la fase depende de la fase del proyecto, por ejemplo el informe de viabilidad incluirá los costes esperados del proyecto, y una recomendación para seguir o abandonarlo.

Diccionario del sistema

El Diagrama de Flujo de Datos describe el sistema. El diagrama E/R, describe los datos del sistema. El componente descripción del proceso describe los procesos del DFD y el Diccionario de datos que describe los datos del sistema (flujos y almacenes de datos). Los usuarios del sistema y como lo utilizan se incluyen en la descripción del usuario.

Descripción de procesos

La descripción de procesos incluyen una entrada por cada proceso del diagrama de flujo de datos. Cada entrada del proceso incluye el número de DFD para él, junto con la descripción del proceso. Como ejemplo la descripción de un proceso de alto nivel, incluye el número y nombre del proceso, los nombres de los flujos de datos de entrada y salida y una descripción del proceso.

Para la descripción de procesos en los niveles inferiores de DFD, se usará un método de descripción de procesos, no así en los DFD de alto nivel que basta una descripción narrativa del proceso.

Diccionario de Datos

El diccionario de datos es una lista organizada de todos los datos pertenecientes al sistema, con una serie de definiciones precisas y rigurosas para que tanto el analista como el usuario comprendan entradas, salidas, elementos de los almacenamientos y cálculos intermedios.

En el diccionario de datos incluimos almacenes de datos, flujos de datos, estructuras de datos, elementos de datos y en algunos casos el modelo E/R.

El diccionario de datos (DD) define los datos en cuanto que:

1. Describe el *significado* de los flujos de datos y los almacenes que muestran los DFDs.
2. Describe la *composición* de la estructura de datos que se mueven a lo largo de los flujos.

3. Describe la composición de la estructura de datos en los almacenes.
4. Describe los detalles de las relaciones entre almacenes que aparecen en un diagrama entidad/relación.

Los analistas utilizan los diccionarios de datos por cuatro razones:

1. Para manejar los detalles en sistemas grandes ya que es imposible de recordar todo lo referente a un sistema.
2. Para comunicar un significado común para todos los elementos del sistema. Esto es muy importante cuando trabajan varios analistas y no pueden reunirse todos los días para comunicarse.
3. Para documentar las características del sistema.
4. Localizar errores en el sistema.

Contenido de un Diccionario de Datos

El DD contiene las siguientes:

1. Definiciones lógicas de datos:
    - Elemento de Dato (Atributos de la Entidad).
    - Estructura de Dato.
    - Flujos de Datos.
    - Almacenes de Datos.
2. Definiciones lógicas de procesos.
3. Definición lógica de entidad externa.

Los elementos de dato se agrupan para formar una estructura de dato.

1. *Elemento de dato*: Ninguna unidad más pequeña tiene significado para los analistas o usuarios. Son los bloques básicos para todos los demás datos del sistema, por sí solo no lleva ningún significado al usuario. Son los atributos de las entidades.
   Por ejemplo: nº factura, fecha expedición, cantidad adeudada.
2. *Estructura de dato*: es un grupo de datos elementales que en conjunto describen un componente del sistema.
   Por ejemplo: Factura.

Los flujos de datos, almacenes de datos son estructuras de datos.

Notación del Diccionario de Datos

1. *Notación del elemento dato*: Cada uno está identificado con un nombre, una descripción, un alias, una longitud, un intervalo de valores. Veamos las reglas a seguir para cada elemento.
    - Nombre de los datos: se deben asignar nombres que sean significativos, es decir, que tengan significado en el contexto del desarrollo del sistema.
      Por ejemplo: Fecha-Factura. Un nombre no debe ser mayor de 30 caracteres y tampoco debe contener espacios en blanco.
    - Descripción de los datos: indica de manera breve lo que éste representa en el sistema, y debe escribirse de forma comprensible para el lector y pensando que quien lo lea no sabe nada con respecto al sistema.
    - Alias: es cuando el mismo dato recibe varios nombres, según quien haga uso del dato.
      Ejemplo: factura puede tener como alias documento de pago o nota de pago etc.
      No so alias los siguientes casos: factura autorizada, factura verificada.
    - Longitud: indica la cantidad de espacio necesario para cada dato sin considerar la forma en que serán almacenados.
    - Valores de los datos: si los valores de los datos están restringidos a un intervalo específico, debe reflejarse en la entrada del DD.
      Por ejemplo: Talla unidad [centímetros], rango [1-200].

2. *Descripción de las estructuras de datos*: Las estructuras de datos se construyen sobre cuatro relaciones de componentes (datos o estructuras) que son:
    - Relación secuencial: Define los componentes (datos o estructuras) que siempre se incluyen en una estructura de datos en particular, es decir, también se llama concatenación de dos o más datos.
    - Relación de selección: Define alternativas para datos o estructuras incluidas en una estructura de datos.
    - Relación de iteración: Define la repetición de un componente cero o más veces.
    - Relación opcional: Es un caso especial de la iteración, es decir, una o ninguna relación.
3. *Descripción de los flujos de datos*: Representamos los flujos de datos siempre y cuando el flujo no sea un único atributo. Está formado por una o más estructuras previamente definidas. Del flujo nos interesa el contenido, fuente, destino, volumen.
    - Nombre del flujo de datos: se deben asignar nombres que sean significativos, es decir, que tengan significado en el contexto del desarrollo del sistema.
      Por ejemplo: factura.
    - Fuente: indica cual es el proceso fuente de la información. Se indicará el número del proceso.
    - Destino: indica cual es el proceso destino de la información. Se indicará el número del proceso.
    - Definición: explica el contenido del flujo de datos.
    - Contenido: describe cuales son las estructuras de datos incluidas.
4. *Descripción de los almacenamientos de datos*: Representamos los almacenamientos de datos. Se documenta su contenido, flujos de entrada, flujos de salida.
    - Nombre de almacenamiento de datos: se asignan nombres que sean significativos, es decir, que tengan significado en el contexto del desarrollo del sistema.
      Ejemplo: histórico facturas.
    - Flujos de entrada: indica cuales son los flujos que alimentan el almacenamiento de datos.
    - Flujos de salida: indica cuales son los flujos que extraen información del almacenamiento de datos.
    - Definición: describe el contenido del almacenamiento de datos.
    - Contenido: especifica el contenido del almacenamiento.
5. *Descripción de los procesos*: Representamos los procesos del sistema. Se documenta su contenido, flujos de entrada, flujos de salida.
    - Nombre de proceso: se asignan nombres que sean significativos, es decir, que tengan significado en el contexto del desarrollo del sistema.
      Por ejemplo: verificar_crédito.
    - Entradas: indica cuales son los procesos, almacenamientos de datos que ejercen de fuente de datos.
    - Flujos de salida: indica cuales son los procesos, almacenamientos de datos que ejercen de destino de datos.
    - Definición: indica la misión del proceso.
    - Descripción: describe el proceso. Para ello utilizaremos: Forma narrativa, árboles de decisión, tablas de decisión, lenguaje estructurado.
6. *Descripción de las entidades externas*: Representamos las entidades externas del sistema. Se documenta a quien representa, flujos de datos relacionados, volumen, etc.
    - Nombre de entidad externa: se asignan nombres que sean significativos, que representen a la entidad.
      Por ejemplo: clientes.
    - Flujos de datos asociados: indica cuales son los flujos (entrada/salida) asociados.
    - Definición: indica quienes son la entidad.
    - Volumen: número de componentes de la entidad.

Sintaxis del Diccionario de Datos

Conocida la forma de describir los datos y estructuras de datos, explicados en el apartado anterior, a continuación se va a establecer una sintaxis estandarizada que nos permitirá expresar dichos significados:

- **=** está compuesto por
- **+** y
- **()** opcional, puede o no puede estar presente
- **[]** selección entre varias alternativas
- **{}** iteración, repetir los mismo varias veces
- ** comentario
- @ clave principal de un almacenamiento
- **|** separador de alternativas en selección

Ejemplo:

Datos elementales

Son datos, que dentro del contexto del usuario, no tiene sentido descomponerlo. Es importante verificar: Valores permitidos y unidad de medida.

```
peso_persona =* *
                * unidad: kilo ; rango: 1..150 *
sexo = * Masculino o Femenino *
       * valores: [ M | F ] *
```

Datos opcionales

```
Dirección_cliente = (dirección_entrega) + dirección_facturación)
Dirección_cliente = [ dirección_entrega | dirección_facturación |
                      dirección entrega + dirección_facturación ]
Dirección_cliente = dirección_entrega + (dirección_facturación)
```

Iteración

Repetición de uno o más datos elementales o grupo de datos. ‘Cero o más ocurrencias’.

```
pedido = nombre_cliente + dirección_entrega + { producto }
```

Selección

‘Una y solo una de las alternativas’.

```
sexo = [ Masculino | Femenino ]
```

Dominio (No Yourdon)

Consiste definir una única vez cada tipo de Dato Elemental y referenciarlo para cada representación del tipo.

```
fecha = * *
          * unidad: días ; rango : 0..36500 *
fec_nacimiento = fecha
fec_factura = fecha
```

Alias (Sinónimo)

No se debe confundir con el dominio. Es un nombre alternativo para un dato elemental.

```
fecha_contable = fecha
fecha_efectiva = * alias de: fecha_contable *
Nombre = Tratamiento + Nombre_pila + Primer_apellido + Segundo_apellido
Tratamiento = [ Sr. | Sra. | Srta. | D. | Dr. ]
Nombre_pila = {carácter}
Primer_apellido = {carácter}
Segundo_apellido = {carácter}
carácter = [ A-Z | a-z | - ]
```

Definición de un Diccionario de Datos

Definición de datos secuenciales

Una definición se realiza mediante el símbolo = que significa *se define como* por lo tanto una expresión como A = B + C, se podría leer igual que de forma matemática, es decir, *A está compuesto de B y C*, pero para completarla se debería añadir: el significado de dicho dato en el contexto de la aplicación, el rango y tipo de valores que cada dato puede tomar.

Por ejemplo: En un sistema informático de un hospital:

```
Datos_del_Paciente = nombre_completo +
                     *nombre completo del paciente *
                     *tipo: array de caracteres*
                     dirección +
                     *dirección completa del paciente*
                     *tipo: array de caracteres*
                     peso +
                     *peso del paciente*
                     *unidad: kilogramos; rango: 1-200*
                     talla +
                     *talla del paciente*
                     *unidad: centímetros; rango: 20-250*
                     fecha_ingreso +
                     *fecha de entrada en el hospital *
                     *tipo: fecha*
```

Definición de datos opcionales

Es aquel dato que puede o no formar parte de la composición de un dato compuesto.

Ejemplo: La dirección de un cliente puede ser:

- Única: tanto la dirección comercial como de administración están en el mismo lugar que producción o almacén.
- Dos direcciones: tiene el almacén y producción separado físicamente de la administración.

Esta situación en un DD se trataría así:

```
Cliente = nombre_completo + dni_cliente + dirección_comercial +
          (dirección_mercancías)
```

Definición de selección

Sólo una de entre varias posibilidades será posible. Esta se define mediante [].

Ejemplo: Un cliente puede ser una empresa o un particular, por lo tanto los tipos de datos son distintos según sea uno u otro.

```
cliente = [nombre_cliente | nombre_empresa] + [dni_cliente | cif_cliente] +
          dirección_comercial + (dirección_mercancías)
```

Definición de iteración

La iteración se expresa mediante {} y sirve para indicar la repetición de una cierta ocurrencia dentro de una definición.

Ejemplo:

```
Factura = fecha_factura + nombre_cliente + numero_factura +
          {linea_factura} + total_factura
```

El dato línea_factura es un componente de la estructura de datos factura que puede tener una o varias ocurrencias, ya que una factura puede tener muchas líneas de facturación de artículos.

Alias (Sinónimos)

Son nombres que dentro del Sistema de información tienen el mismo significado, entonces lo que se hace es declarar los sinónimos por medio del símbolo =.

Ejemplo:

```
Acreedor = cliente
           ** definido ya anteriormente.
```

Hemos visto el contenido del Diccionario de Datos, que deberá mostrarse al usuario siempre conjuntamente con las técnicas:

- Diagrama de Flujo de Datos (DFD).
- Modelo Entidad/Relación (DER).
- Especificación de Procesos (EP).

Implementación del Diccionario de Datos

Varias posibilidades para la implementación de los DD, cada una con sus características y ventajas.

- Repositorio de datos:
  - Herramientas automáticas integradas dentro de un entorno CASE.
  - Dispone de más posibilidades de las vistas.
- Diccionario de datos SGBD o SO modernos.
  - Dan soporte automático para definiciones de datos, validar su consistencia, producir algunos informes.
- Procesador de textos convencional.
- Totalmente manual.

Ejemplo:

Dato Elemental

```
Nombre     : Estado_Civil
Descripción: Código de una letra para indicar el estado civil de cada
             empleado.
Long y tipo: 1 carácter alfabético.
Sinónimos  : ESTADO (Personal)
             CIVIL (Nóminas)
Valores    : S Soltero    D Divorciado
```

```
            C Casado     S Separado
            V Viudo      O Otros
```

Estructura de dato

```
Nombre     : Empleado
Descripción: Datos necesarios de un empleado
Componentes: Nombre_empleado +
             Num_empleado +
             Datos_personales =
                              Fecha_nacimiento +
                              Estado_Civil +
                              Num_hijos [ 0 - ] +
                             (Num_tfno)
             Dirección =
                         Calle +
                         Número +
                         (Población) +
                         Codigo_Postal +
                         Provincia
```

Flujo de datos

```
Nombre           : Pago_aceptado
Ref              : 11.1 - 11.2
Fuente           : 11.1 Aceptar pago
Destino          : 11.2 Validar pago
Descripción      : Pago recibido y sellado pero no validado
Estruct. de datos: Cheque +
                   Recibo_Caja +
                   (Letra_Pago) +
                   Metodo_Pago
Volumen          : 5000 por día
Comentarios      : La letra de pago está omitida en el 10% de los casos
```

Almacenamiento de datos

```
Nombre          : Historia_Pedidos
Ref             : P4
Flujo de Entrada: 9 - D4   Pedido
Flujo de Salida : D4 - 10    Detalles pedido
                  D4 - 11    Detalle ventas
                  D4 - 9     Demanda anterior
Descripción     : Todos los pedidos aceptados en los últimos 6 meses
Contenido       : Pedido =
                           Id_pedido +
                           Detalle_cliente +
                           Detalle_libro
```

Descripción lógica de un proceso

Para el proceso V*erificar_Crédito* la plantilla correspondiente sería la siguiente:

Procesos

```
Nombre     : Verificar_crédito
Ref        : 3
Definición : Decidir donde van los pedidos sin pago previo,
             o si debe pedirle el pago al cliente.
```

```
Entradas   : 1 - 3    Pedidos
             D3 - 3   Historia de pagos
Salidas    : 3 - C    Pedido de pago previo
             3 - D3   Nuevo balance de orden
             3 - 6    Pedidos con crédito ok
Descripción: Recuperar historia de pago.
             Si el cliente es nuevo, enviar pedido de pago previo.
             Si el cliente corriente (promedio de dos pedidos mensuales)
             , OK con el pedido, a menos que el balance esté vencido con
             más de dos meses. Para clientes anteriores (no corrientes)
             , OK, a menos que tengan cualquier balance vencido.
```

Hemos visto que para describir la lógica de un proceso, utilizaremos varias alternativas como son: narrativa, árboles de decisión, tablas de decisión y lenguaje estructurado.

Cuando utilizamos narrativa podemos encontrarnos con:

- frases oscuras (no solo, pero no obstante, sin embargo, …)
- rangos con huecos indefinidos (‘ hasta 20 unidades sin descuento, más de 20 unidades al 50%’).
- Frases con y/o (‘los clientes que nos compran más de 1 millón al año y tienen una buena historia de pago o que han tenido tratos con nosotros por más de 20 años deberán recibir trato preferencial’).
- Adjetivos indefinidos (‘buena historia de pagos’, ‘trato preferencial’).

Estas razones obligan a pensar en otras alternativas:

- **Árbol de decisión**: Pueden resultar una técnica no válida en situaciones complejas con gran número de condiciones e implicaciones ya que no asegura que se hayan considerados todas.
  Se debe utilizar cuando el número de acciones sea pequeño y no sean posibles todas las combinaciones.
- **Tablas de decisión**: Son más precisas dado que permiten reflejar todas la combinaciones posibles. Pero son más difíciles de entender para el usuario. Deben simplificarse una vez construidas y se convertirán en árboles de decisión.
  Se debe utilizar siempre que se dude que el árbol muestra toda la lógica.

  ```
  * Primera orden > 12 días ---------- Hacer pedido
  * Total ordenes < menor que X
  * Primera orden <= 12 días ---------- Esperar
  * Total ordenes < mayor o igual que X ---------- Calcular
  * Hacer pedido
  * No descuento ---------- Hacer pedido
  ```

Descripción lógica de una entidad externa

Para la entidad *Proveedores* la plantilla correspondiente sería la siguiente:

Entidad Externa

```
Nombre         : Proveedores
Ref            : p
Definición     : Proveedores actuales de la empresa
Flujos de Datos: 7 - p     Pedidos
                 p - 3     Albarán
                 p - 11    Facturas
Volumen        : Actualmente 25. Se espera llegar a 40.
```

Bibliografia

- dlsi (Jaime)
- PAe

Modelización conceptual. El modelo Entidad/Relación extendido (E/R): elementos. Reglas de modelización. Validación y construcción de modelos de datos.

Modelo Conceptual de Datos

A la hora de determinar una BD debemos establecer un proceso partiendo del acotamiento de una parcela del mundo exterior (micromundo o universo del discurso), aquél que nos interesa representar en los datos. En este proceso se debe aprender, comprender y conceptualizar dicho mundo exterior transformándolo en un conjunto de ideas y definiciones que supongan una imagen fiel del comportamiento del mundo real. A esta imagen del mundo exterior la llamaremos **Modelo Conceptual**.

Una vez definido el modelo conceptual, éste se ha de transformar en una descripción de datos, atributos y relaciones que denominaremos Esquema Conceptual de datos. Por último, este esquema conceptual habrá que traducirlo a estructuras almacenables en soportes físicos. Por tanto es necesario distinguir entre Bases de Datos, que será el banco, el almacén de los valores (ocurrencias) de los datos. Los modelos de Datos, que son las herramientas para diseñar los datos y sus relaciones de forma que puedan soportar los valores correspondientes. Y finalmente los Sistemas Gestores de Bases de Datos (SGBD), que serán los encargados de las acciones que llevemos a cabo con las bases de datos, permitiendo también cumplimentar a los usuarios, mostrándoles los datos de acuerdo a sus necesidades. Con todo ello, se puede definir un Modelo de Datos como: Un grupo de herramientas conceptuales para describir los datos, sus relaciones, su semántica y sus limitaciones; de tal forma, que facilita la interpretación de nuestro mundo real y su representación en forma de datos, en nuestro sistema informático.

Definido el modelo de datos, pasamos a analizarlo. Para ello, partiremos de las propiedades que podemos diferenciar en dos tipos:

- **Estáticas**: Son las propiedades invariantes en el tiempo. Quedan especificadas en el Modelo de Datos por ESTRUCTURAS. Esta se define mediante el ESQUEMA, con el lenguaje de definición de datos (DDL). El esquema, a su vez está constituido por Estructura y Restricciones. La Estructura queda definida por los Objetos del Modelo y las Restricciones inherentes, conformando un conjunto de reglas de definición de dichas Estructuras. Los objetos y Restricciones de la Estructura dependen de cada Modelo, pero en general son:
  - Entidades.
  - Atributos.
  - Dominio.
  - Relaciones.
  - Representación.

    - Restricciones: Hay tres tipos de restricciones:
        - *Restricciones inherentes*: vienen impuestas por la propia naturaleza del Modelo introduciendo rigideces en la modelización.
        - *Restricciones opcionales o de usuario*: restricciones propiamente dichas en el Esquema, son definidas por el usuario, pero el Modelo de Datos las reconoce y suministra herramientas para manejarlas.
        - *Restricciones libres de usuarios*: son responsabilidad del usuario y el Modelo de Datos ni las reconoce, ni las maneja.
- **Dinámicas**: Son las propiedades que varían con el tiempo. En el modelo de datos son las OPERACIONES. Se define como un conjunto de Operaciones con el Lenguaje de Manipulación de Datos (DML). Las operaciones sobre un Modelo de Datos pueden ser de:
    - Selección. Localización de los datos deseados.
    - Acción. Realización de una acción sobre los datos seleccionados. Dicha acción puede ser:
        - Recuperación (obtención de los datos seleccionados).
        - Actualización, que a su vez pueden ser:
            - Modificación.
            - Inserción.
            - Borrado.

Generalmente, toda operación de Actualización va precedida de una Recuperación, aunque no necesariamente.

Modelo Entidad/Relación Extendido

Se trata de una técnica cuyo objetivo es la representación y definición de todos los datos que se introducen, almacenan, transforman y producen dentro de un sistema de información, sin tener en cuenta las necesidades de la tecnología existente, ni otras restricciones.

Dado que el modelo de datos es un medio para comunicar el significado de los datos, las relaciones entre ellos y las reglas de negocio de un sistema de información, una organización puede obtener numerosos beneficios de la aplicación de esta técnica, pues la definición de los datos y la manera en que éstos operan son compartidos por todos los usuarios.

Las ventajas de realizar un modelo de datos son, entre otras:

- Comprensión de los datos de una organización y del funcionamiento de la organización.
- Obtención de estructuras de datos independientes del entorno físico.
- Control de los posibles errores desde el principio, o al menos, darse cuenta de las deficiencias lo antes posible.
- Mejora del mantenimiento.

Aunque la estructura de datos puede ser cambiante y dinámica, normalmente es mucho más estable que la estructura de procesos. Como resultado, una estructura de datos estable e integrada proporciona datos consistentes que puedan ser fácilmente accesibles según las necesidades de los usuarios, de manera que, aunque se produzcan cambios organizativos, los datos permanecerán estables.

Este diagrama se centra en los datos, independientemente del procesamiento que los transforma y sin entrar en consideraciones de eficiencia. Por ello, es independiente del entorno físico y debe ser una fiel representación del sistema de información objeto del estudio, proporcionando a los usuarios toda la información que necesiten y en la forma en que la necesiten.

Descripción

El modelo entidad/relación extendido describe con un alto nivel de abstracción la distribución de datos almacenados en un sistema. Existen dos elementos principales: las entidades y las relaciones. Las extensiones al modelo básico añaden además los atributos de las entidades y la jerarquía entre éstas. Estas extensiones tienen como finalidad aportar al modelo una mayor capacidad expresiva.

Los elementos fundamentales del modelo son los siguientes:

Entidad

Es aquel objeto, real o abstracto, acerca del cual se desea almacenar información en la base de datos. La estructura genérica de un conjunto de entidades con las mismas características se denomina tipo de entidad.

Existen dos clases de entidades:

- Regulares: tienen existencia por sí mismas.
- Débiles: cuya existencia depende de otra entidad.

Las entidades deben cumplir las siguientes tres reglas:

- Tienen que tener existencia propia.
- Cada ocurrencia de un tipo de entidad debe poder distinguirse de las demás.
- Todas las ocurrencias de un tipo de entidad deben tener los mismos atributos.

Relación

Es una asociación o correspondencia existente entre una o varias entidades. La relación puede ser regular, si asocia tipos de entidad regulares, o débil, si asocia un tipo de entidad débil con un tipo de entidad regular. Dentro de las relaciones débiles se distinguen la **dependencia en existencia** y la **dependencia en identificación**.

Se dice que la dependencia es en existencia cuando las ocurrencias de un tipo de entidad débil no pueden existir sin la ocurrencia de la entidad regular de la que dependen. Se dice que la dependencia es en identificación cuando, además de lo anterior, las ocurrencias del tipo de entidad débil no se pueden identificar sólo mediante sus propios atributos, sino que se les tiene que añadir el identificador de la ocurrencia de la entidad regular de la cual dependen.

Además, se dice que una relación es **exclusiva** cuando la existencia de una relación entre dos tipos de entidades implica la no existencia de las otras relaciones.

Una relación se caracteriza por:

- **Nombre**: que lo distingue unívocamente del resto de relaciones del modelo.
- **Tipo de correspondencia**: es el número máximo de ocurrencias de cada tipo de entidad que pueden intervenir en una ocurrencia de la relación que se está tratando. Conceptualmente se pueden identificar tres clases de relaciones:
  - *Relaciones 1:1*: Cada ocurrencia de una entidad se relaciona con una y sólo una ocurrencia de la otra entidad.
  - *Relaciones 1:N*: Cada ocurrencia de una entidad puede estar relacionada con cero, una o varias ocurrencias de la otra entidad.
  - *Relaciones M:N*: Cada ocurrencia de una entidad puede estar relacionada con cero, una o varias ocurrencias de la otra entidad y cada ocurrencia de la otra entidad puede corresponder a cero, una o varias ocurrencias de la primera.

- **Cardinalidad**: representa la participación en la relación de cada una de las entidades afectadas, es decir, el número máximo y mínimo de ocurrencias de un tipo de entidad que pueden estar interrelacionadas con una ocurrencia de otro tipo de entidad. La cardinalidad máxima coincide con el tipo de correspondencia.
  Según la cardinalidad, una relación es obligatoria, cuando para toda ocurrencia de un tipo de entidad existe al menos una ocurrencia del tipo de entidad asociado, y es opcional cuando, para toda ocurrencia de un tipo de entidad, puede existir o no una o varias ocurrencias del tipo de entidad asociado.

Dominio

Es un conjunto nominado de valores homogéneos. El dominio tiene existencia propia con independencia de cualquier entidad, relación o atributo.

Atributo

Es una propiedad o característica de un tipo de entidad. Se trata de la unidad básica de información que sirve para identificar o describir la entidad. Un atributo se define sobre un dominio. Cada tipo de entidad ha de tener un conjunto mínimo de atributos que identifiquen unívocamente cada ocurrencia del tipo de entidad. Este atributo o atributos se denomina identificador principal. Se pueden definir restricciones sobre os atributos, según las cuales un atributo puede ser:

- Univaluado, atributo que sólo puede tomar el valor para todas y cada una de las ocurrencias del tipo de entidad al que pertenece.
- Obligatorio, atributo que tiene que tomar al menos un valor para todas y cada una de las ocurrencias del tipo de entidad al que pertenece.

Además de estos elementos, existen extensiones del modelo entidad/relación que incorporan determinados conceptos o mecanismos de abstracción para facilitar la representación de ciertas estructuras del mundo real:

- La **generalización**, permite abstraer un tipo de entidad de nivel superior (supertipo) a partir de varios tipos de entidad (subtipos); en estos casos los atributos comunes y relaciones de los subtipos se asignan al supertipo. Se pueden generalizar por ejemplo los tipos *profesor* y *estudiante* obteniendo el *supertipo* persona.
- La **especialización** es la operación inversa a la generalización, en ella un supertipo se descompone en uno o varios subtipos, los cuales heredan todos los atributos y relaciones del supertipo, además de tener los suyos propios. Un ejemplo es el caso del tipo *empleado*, del que se pueden obtener los subtipos *secretaria*, *técnico* e *ingeniero*.
- **Categorías**. Se denomina categoría al subtipo que aparece como resultado de la unión de varios tipos de entidad. En este caso, hay varios supertipos y un sólo subtipo. Si por ejemplo se tienen los tipos *persona* y *compañía* y es necesario establecer una relación con *vehículo*, se puede crear *propietario* como un subtipo unión de los dos primeros.
- La **agregación**, consiste en construir un nuevo tipo de entidad como composición de otros y su tipo de relación y así poder manejarlo en un nivel de abstracción mayor. Por ejemplo, se tienen los tipos de entidad *empresa* y *solicitante de empleo* relacionados mediante el tipo de relación *entrevista*; pero es necesario que cada *entrevista* se corresponda con una determinada *oferta de empleo*. Como no se permite la relación entre tipos de relación, se puede crear un tipo de entidad compuesto por *empresa*, *entrevista* y *solicitante de empleo* y relacionarla con el tipo de entidad *oferta de empleo*. El proceso inverso se denomina desagregación.
- La **asociación**, consiste en relacionar dos tipos de entidades que normalmente son de dominios independientes, pero coyunturalmente se asocian.

La existencia de supertipos y subtipos, en uno o varios niveles, da lugar a una **jerarquía**, que permitirá representar una restricción del mundo real.

Una vez construido el modelo entidad/relación, hay que analizar si se presentan redundancias. Para poder asegurar su existencia se deben estudiar con mucho detenimiento las cardinalidades mínimas de las entidades, así como la semántica de las relaciones.

Los atributos redundantes, los que se derivan de otros elementos mediante algún cálculo, deben ser eliminados del modelo entidad/relación o marcarse como redundantes.

Igualmente, las relaciones redundantes deben eliminarse del modelo, comprobando que al eliminarlas sigue siendo posible el paso, tanto en un sentido como en el inverso, entre las dos entidades que unían.

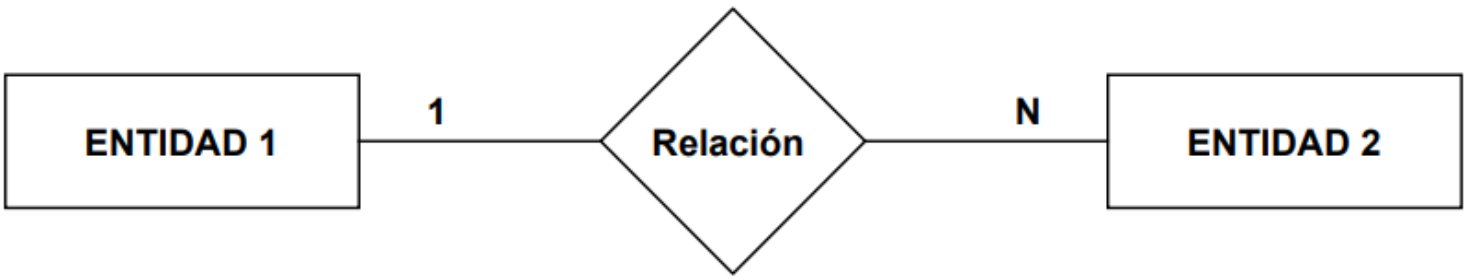
Notación

Entidad

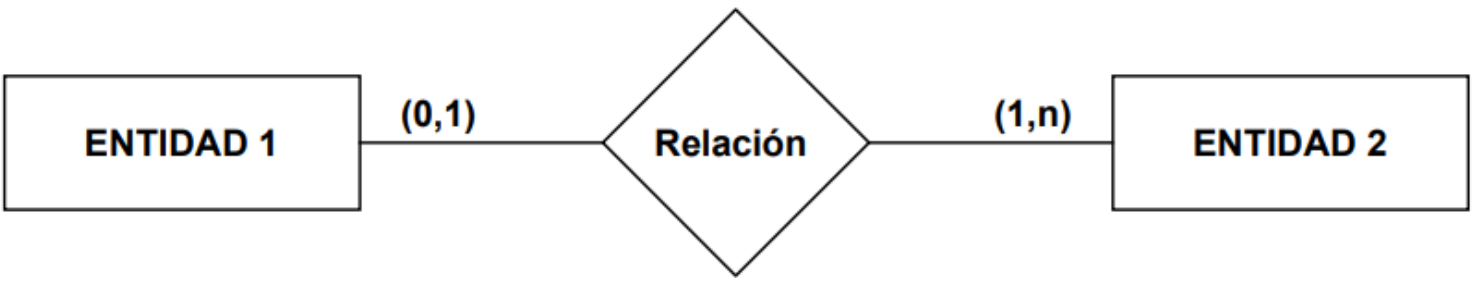
La representación gráfica de un tipo de entidad regular es un rectángulo con el nombre del tipo de entidad. Un tipo de entidad débil se representa con dos rectángulos concéntricos con su nombre en el interior.

Relación

Se representa por un rombo unido a las entidades relacionadas por dos líneas retas a los lados. El tipo de correspondencia se representa gráficamente con una etiqueta *1:1*, *1:N* o *M:N*, cerca de alguno de los vértices del rombo, o bien situando cada número o letra cerca de la entidad correspondiente, para mayor claridad.

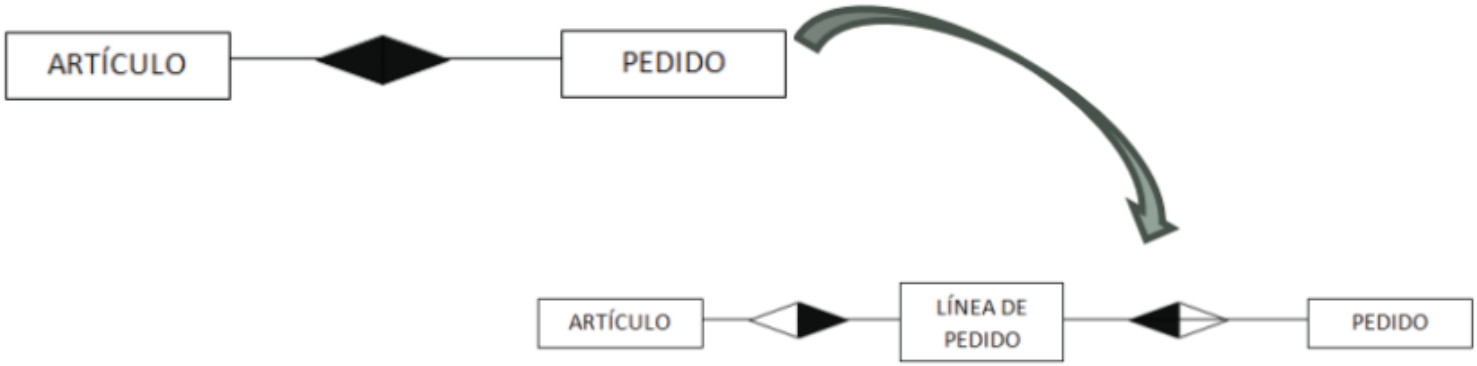


La representación gráfica de las cardinalidades se realiza mediante una etiqueta del tipo *(0,1)*, (*1,1*), *(0,n)* o *(1,n)*, que se coloca en el extremo de la entidad que corresponda. Si se representan las cardinalidades, la representación del tipo de correspondencia es redundante.



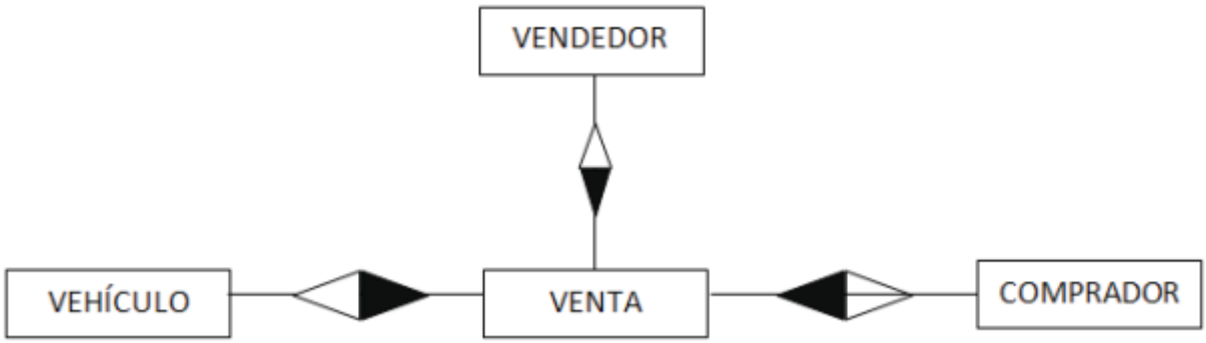
Relaciones del tipo M:N (Muchos a Muchos):

Si existe un concepto que puede sustituir la relación, tiene sentido como entidad y aporta una mejor comprensión al modelo (para usuarios y analistas) es conveniente deshacerlas mediante esta entidad y las relaciones uno a muchos adecuadas.



Relaciones entre Tres o Más Entidades

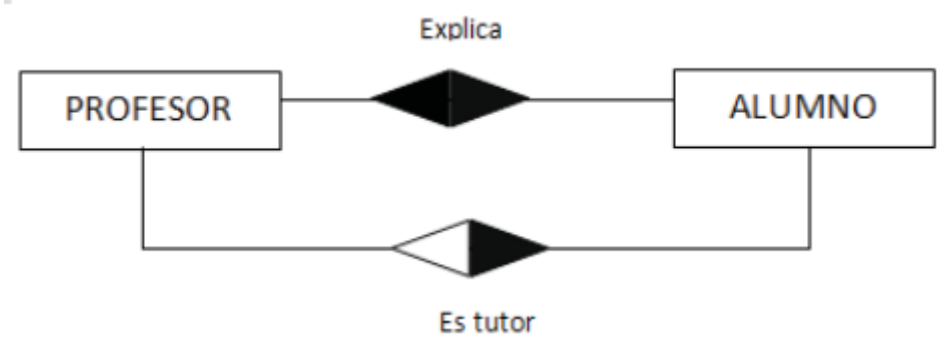
Las relaciones entre tres o más entidades se reclasificarán mediante una entidad relacionada con cada una de ellas, si existe un concepto que puede ser representado como una entidad, y aporta mayor comprensión al problema.



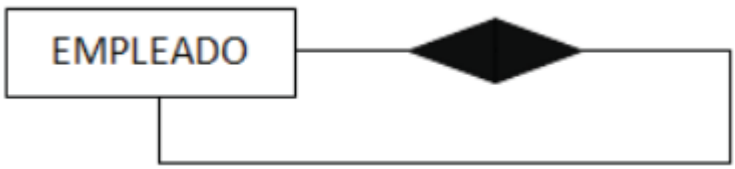
Relaciones potencialmente redundantes

Pueden serlo o no, depende del significado de las relaciones y de las cardinalidades.

Deben ser eliminadas.



Relaciones Recursivas o Autorrelaciones



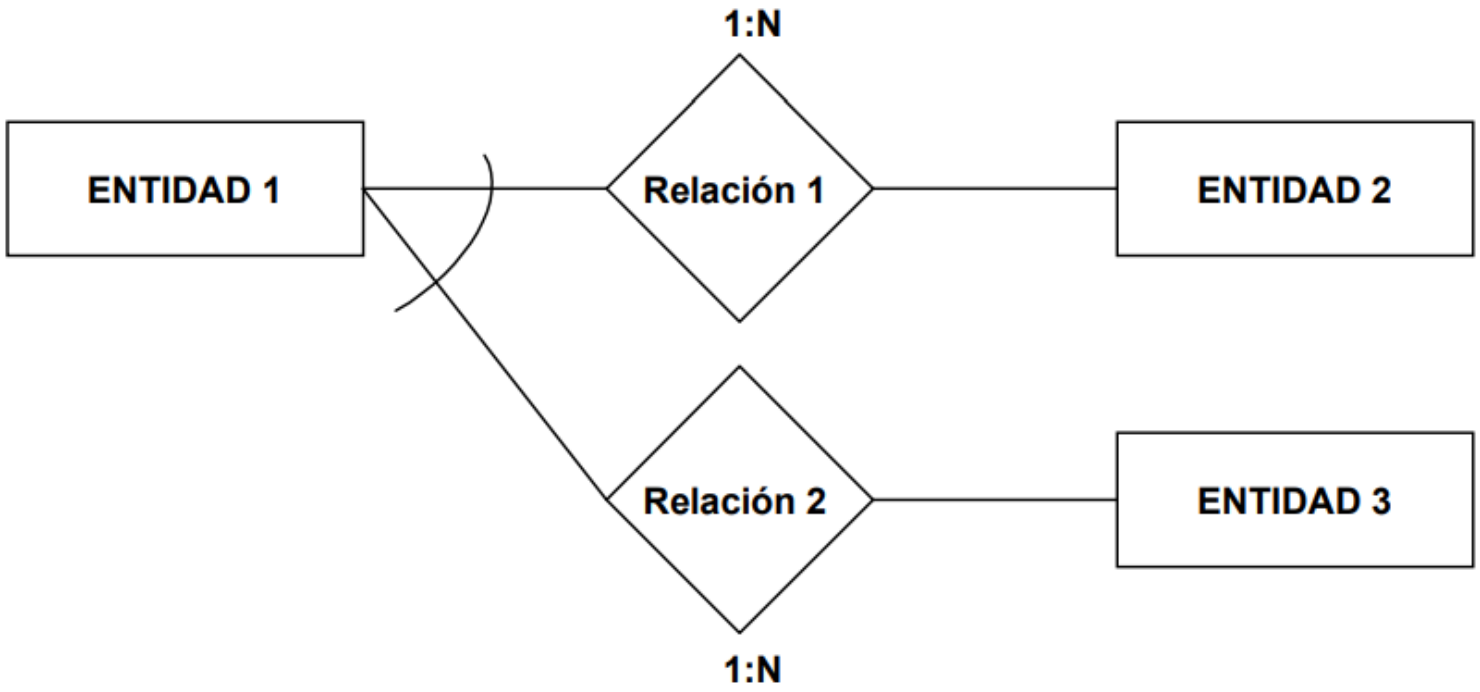
Atributo

Un atributo se representa mediante una elipse, con su nombre dentro, conectada por una línea al tipo de entidad o relación.

En lugar de una elipse puede utilizarse un círculo con el nombre dentro, o un círculo más pequeño con el nombre del atributo a un lado. También pueden representarse en una lista asociada a la entidad. El identificador aparece con el nombre marcada o subrayado, o bien con su círculo en negro.

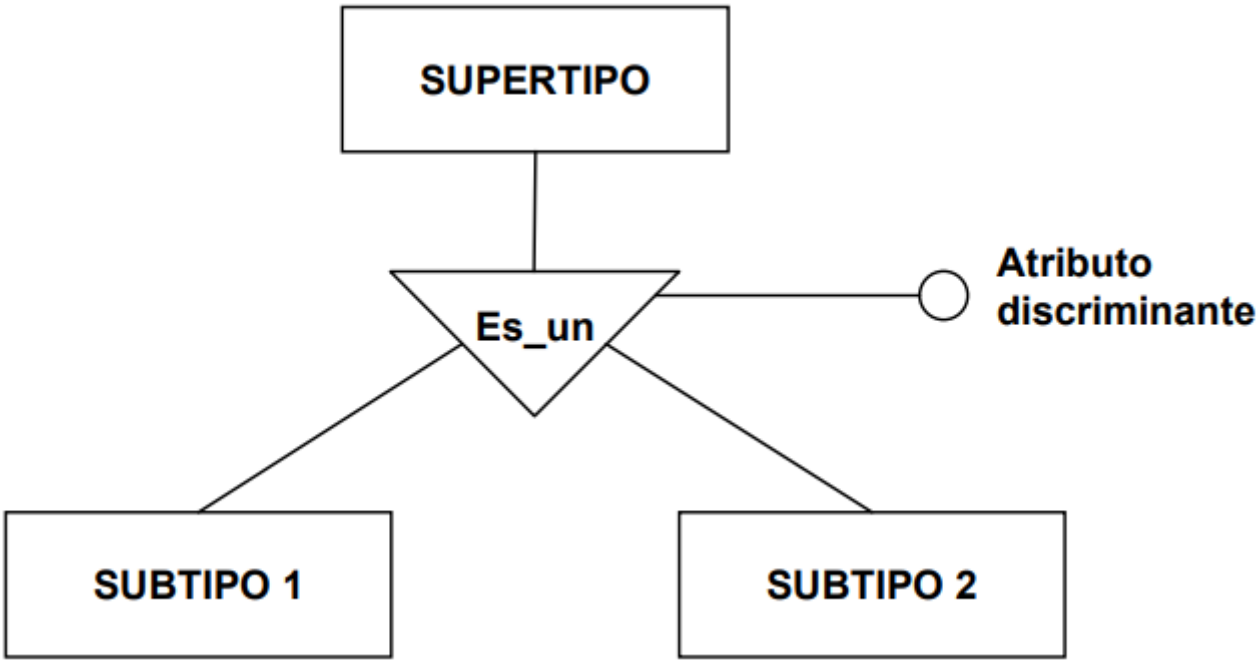
Exclusividad

En la representación de las relaciones exclusivas se incluye un arco sobre las líneas que conectan el tipo de entidad a los dos o más tipos de relación.

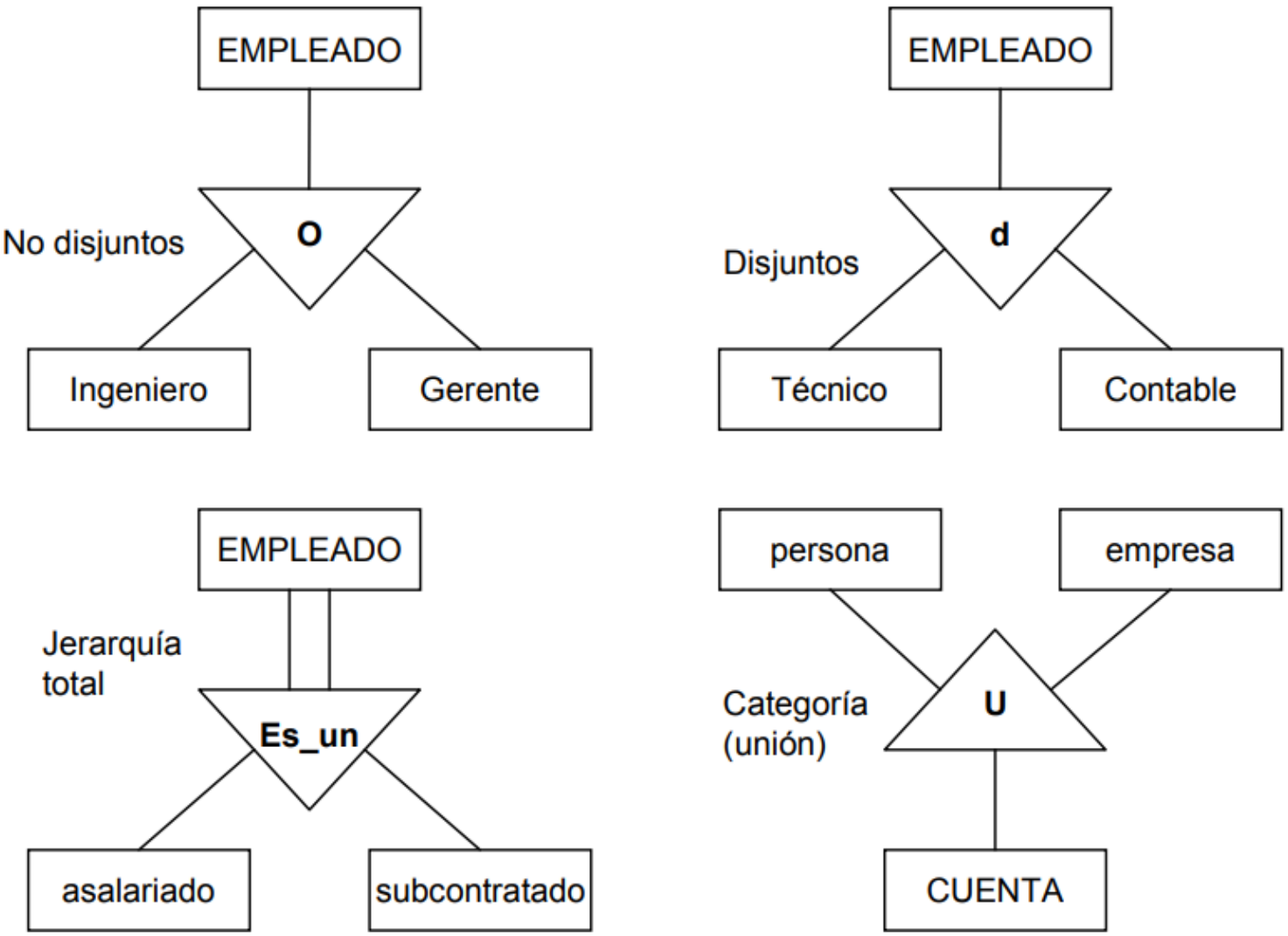


Jerarquía (tipos y subtipos)

La representación de las jerarquías se realiza mediante un triángulo invertido, con la base paralela al rectángulo que representa el supertipo y conectando a éste y a los subtipos. Si la división en subtipos viene determinada en función de los valores de un atributo discriminante, éste se representará asociado al triángulo que representa la relación.



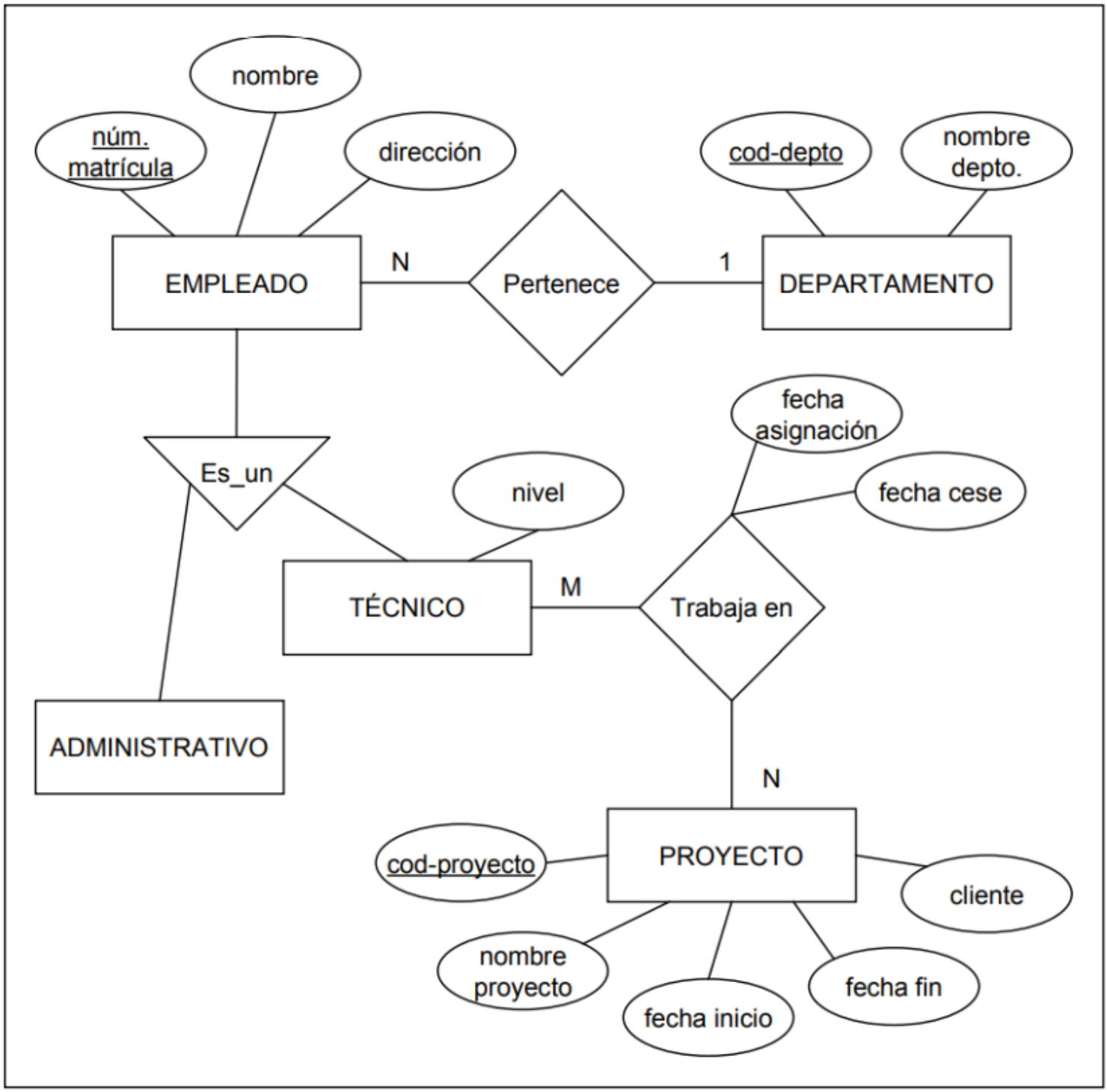
En el triángulo se representará: con una letra **d** el hecho de que los subtipos sean disjuntos, con un círculo o una **O** si los subtipos pueden solaparse y con un **U** el caso de uniones por categorías. La presencia de una jerarquía total se representa con una doble línea entre el supertipo y el triángulo.



Ejemplo

Modelo entidad-relación extendido para un sistema de gestión de técnicos y su asignación a proyectos dentro de una empresa u organización.

Como se aprecia en el diagrama, TÉCNICO es un subtipo de EMPLEADO, generado por especialización, pues era necesario para establecer la relación *Trabaja en* con PROYECTO, ya que no todos los empleados de la empresas, como los administrativos, son susceptibles de trabajar en un proyecto. La entidad TÉCNICO tendrá los atributos de EMPLEADO más el atributo *nivel*.



Los tipos de correspondencia son *1:N* entre DEPARTAMENTO y EMPLEADO, pues un departamento tienen 1 o varios empleados. Entre TÉCNICO y PROYECTO es *M:N*, pues un técnico puede trabajar en 1 o varios proyectos, y en un proyecto trabajan 1 o varios técnicos.

Por otro lado, se han incluido atributos que caracterizan la relación *Trabaja en*, como son *fecha de asignación* y *fecha de cese*, ya que un técnico no siempre estará trabajando en un

proyecto, sino en determinado periodo. (Nota.- Esta notación es la más habitual, pero MÉTRICA Versión 3 no exige su utilización).

EJEMPLO

1–1. Un hombre está casado con una y sólo una mujer

1-n. En una empresa pueden trabajar muchas personas, pero un trabajador sólo trabaja en una empresa.

n-n. Un alumno puede matricularse en muchas asignaturas y además esa asignatura hay otros muchos alumnos.

Esta clasificación de las relaciones se llama ***Cardinalidad***

Normalización

La teoría de la normalización tiene por objetivo la eliminación de dependencias entre atributos que originen anomalías en la actualización de los datos, y proporcionar una estructura más regular para la representación de las tablas, constituyendo el soporte para el diseño de bases de datos relacionales.

Como resultado de la aplicación de esta técnica se obtiene un modelo lógico de datos normalizado.

Descripción

La teoría de la normalización, como técnica formal para organizar los datos, ayuda a encontrar fallos y a corregirlos, evitando así introducir anomalías en las operaciones de manipulación de datos.

Se dice que una relación está en una determinada forma normal si satisface un cierto conjunto de restricciones sobre los atributos. Cuanto más restricciones existan, menor será el número de relaciones que las satisfagan, así, por ejemplo, una relación en tercera forma normal estará también en segunda y en primera forma normal.

Antes de definir las distintas formas normales se explican, muy brevemente, algunos conceptos necesarios para su comprensión.

Dependencia funcional

Un atributo Y se dice que depende funcionalmente de otro X si, y sólo si, a cada valor de X le corresponde un único valor de Y, lo que se expresa de la siguiente forma: $X \rightarrow Y$ (también se dice que X determina o implica a Y).

X se denomina implicante o determinante e Y es el implicado.

Dependencia funcional completa

Un atributo Y tiene dependencia funcional completa respecto de otro X, si depende funcionalmente de él en su totalidad, es decir, no depende de ninguno de los posibles atributos que formen parte de X.

Dependencia transitiva

Un atributo depende transitivamente de otro si, y sólo si, depende de él a través de otro atributo. Así, *Z* depende transitivamente de *X*, si:

```
X → Y
Y ---/→ X
Y → Z
```

Se dice que *X* implica a *Z* a través de *Y*.

Una vez definidas las anteriores dependencias, se pueden enunciar las siguientes formas normales:

Primera Forma Normal (1FN)

Una entidad está en 1FN si no tiene grupos repetitivos, es decir, un atributo sólo puede tomar un único valor de un dominio simple.

Una vez identificados los atributos que no dependen funcionalmente de la clave principal, se formará con ellos una nueva entidad y se eliminarán de la antigua. La clave principal de la nueva entidad estará formada por la concatenación de uno o varios de sus atributos más la clave principal de la antigua entidad.

Segunda Forma Normal (2FN)

Una entidad está en 2FN si está en 1FN y todos los atributos que no forman parte de las claves candidatas (atributos no principales) tienen dependencia funcional completa respecto de éstas, es decir, no hay dependencias funcionales de atributos no principales respecto de una parte de las claves. Cada uno de los atributos de una entidad depende de toda la clave.

Una vez identificados los atributos que no dependen funcionalmente de toda la clave, sino sólo de parte de la misma, se formará con ellos una nueva entidad y se eliminarán de la antigua. La clave principal de la nueva entidad estará formada por la parte de la antigua de la que dependen funcionalmente.

Tercera Forma Normal (3FN)

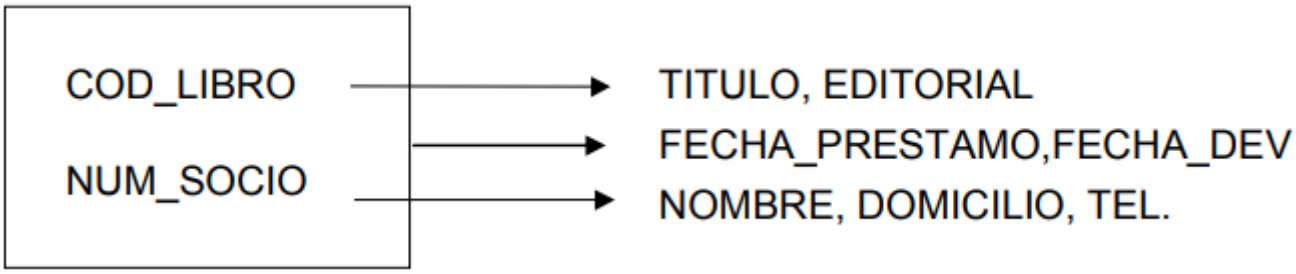
Una entidad está en 3FN si está en 2FN y todos sus atributos no principales dependen directamente de la clave primaria, es decir, no hay dependencias funcionales transitivas de atributos no principales respecto de las claves.

Una vez identificados los atributos que dependen de otro atributo distinto de la clave, se formará con ellos una nueva entidad y se eliminarán de la antigua. La clave principal de la nueva entidad será el atributo del cual dependen. Este atributo en la entidad antigua, pasará a ser una clave ajena.

Notación

Una herramienta muy útil para visualizar las dependencias funcionales es el grafo o diagrama de dependencias funcionales, mediante el cual se representa un conjunto de atributos y las dependencias funcionales existentes entre ellos.

En el grafo aparecen los nombre de los atributos unidos por flechas, las cuales indican las dependencias funcionales completas que existen entre ellos, partiendo del implicante hacia el implicado. Cuando el implicante de una dependencia no es un único atributo, es decir, se trata de un implicante compuesto, los atributos que lo componen se encierran en un recuadro y la flecha parte de éste, no de cada atributo.



En la figura se presenta un ejemplo de cómo se visualizan las dependencias. Se puede observar que COD_LIBRO determina funcionalmente el TITULO del libro y la EDITORIAL, como indica la correspondiente flecha; de forma análoga, NUM_SOCIO determina NOMBRE, DOMICILIO y TEL. del socio (suponiendo que sólo se proporciona un teléfono); mientras que ambos atributos en conjunto COD_LIBRO y NUM_SOCIO (lo que se indica mediante el recuadro que los incluye) determinan FECHA_PRESTAMO y FECHA_DEV.

Ejemplo

Sea una entidad TÉCNICOS de un grupo de empresas, con los siguientes atributos:

- cod_empresa
- cod_técnico
- nombre_técnico
- cod_categoría
- categoría
- nombre_empresa
- fecha_alta
- fecha_baja
- cod_conoc
- titulo_conoc
- área_conoc
- grado
- cod_proyecto
- nombre_proyecto
- f_inicio
- f_fin
- f_asignación
- f_cese
- cod_cliente
- nombre_cliente

La entidad TÉCNICOS tiene la clave principal compuesta por *cod_empresa* y *cod_técnico*, ya que, al ser varias empresas, el código de técnico no será único, sino que puede coincidir con otros de otras empresas.

Primera Forma Normal (1FN)

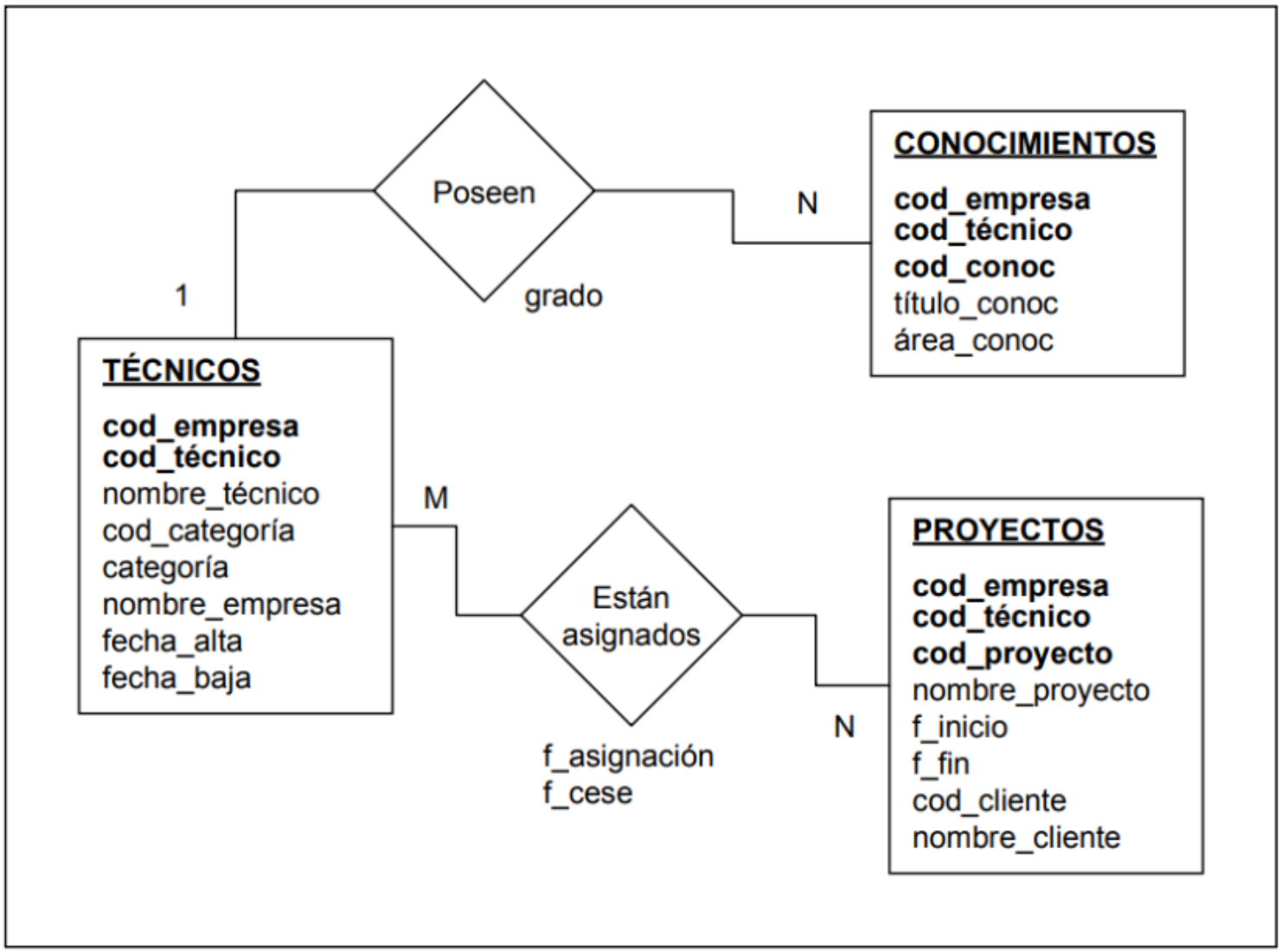
Evidentemente no se cumple la primera forma normal, ya que un técnico tendrá más o de un conocimiento (lenguajes, sistemas, operativos, bases de datos, etc), es decir habrá varios valores de *cod_conoc*, por lo que este atributo y los asociados a conocimientos no dependen funcionalmente de la clave principal.

Los atributos *cod_conoc*, *título_conoc*, *área_conoc* y *grado* identificados como no dependientes, formarán la nueva entidad CONOCIMIENTOS y desaparecerán de la entidad TÉCNICOS. La clave de la nueva entidad será *cod_conoc* concatenada con *cod_empresa* y *cod_técnico*.

Por otro lado, en este sistema un técnico puede trabajar en más de un proyecto a tiempo parcial, por lo que *cod_proyecto* tampoco depende funcionalmente de la clave principal de TÉCNICOS.

Se obtiene entonces la entidad PROYECTOS con los atributos de los proyectos, y su clave compuesta de *cod_proyecto* concatenada con *cod_empresa* y *cod_técnico* de la antigua entidad.

Esta situación se completará con dos tipos de relaciones: *Poseen*, cuyo tipo de correspondencia es *1:N* entre TÉCNICOS y CONOCIMIENTOS y *Están asignados*, también del tipo *M:N* entre TÉCNICOS y PROYECTOS, tal y como muestra el diagrama siguiente:



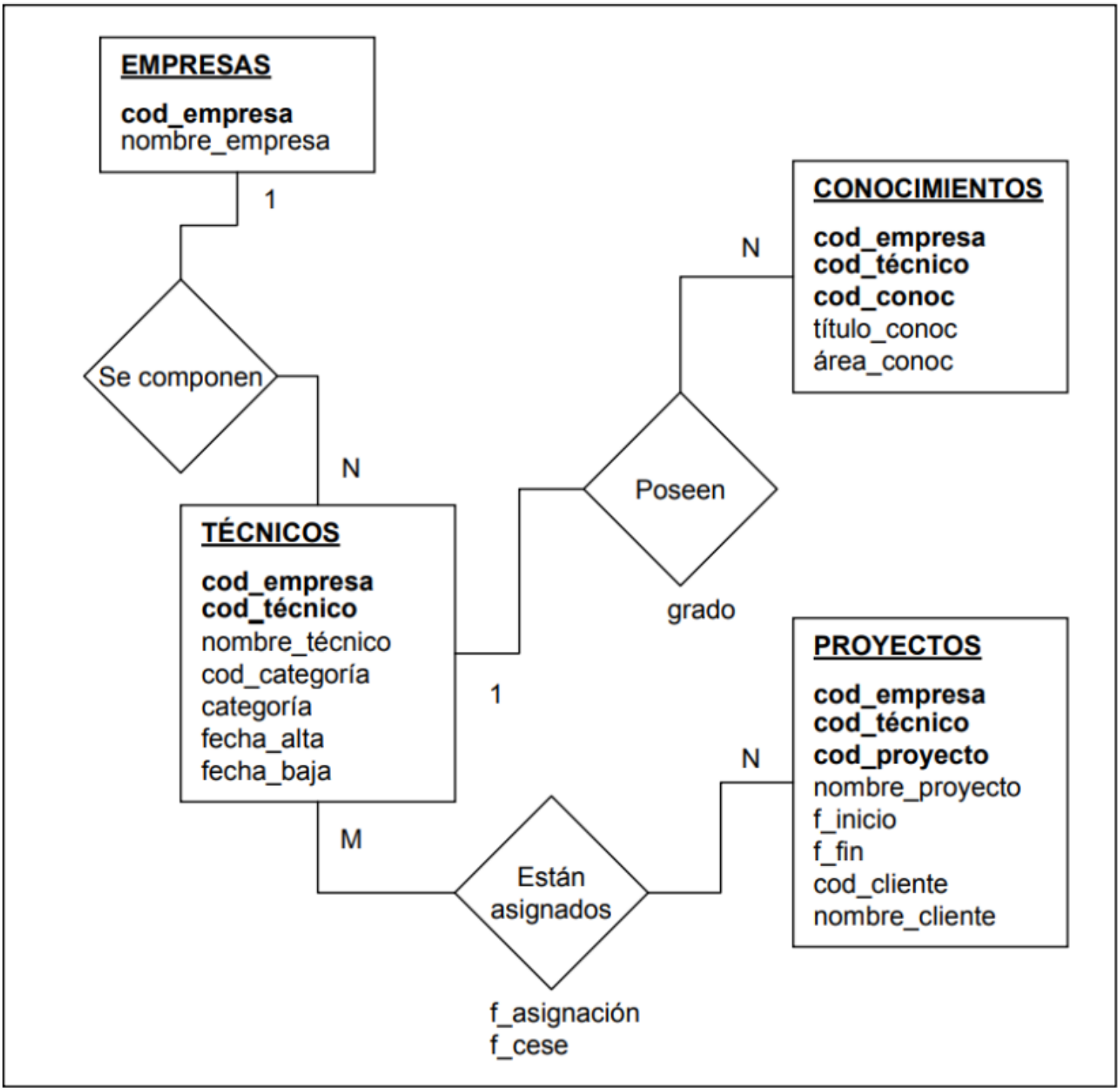
Como se aprecia en la figura, se ha trasladado el atributo *grado* de la entidad CONOCIMIENTOS a la relación *Poseen*, pues es un atributo que determina la relación entre las dos entidades. También han sido trasladado los atributos que caracterizan la relación *Están asignados*, como son *f_asignación* y *f_cese*, ya que un técnico no siempre estará trabajando en un proyecto, sino en determinado periodo.

Segunda Forma Normal (2FN)

En la entidad TÉCNICOS se observa que el atributo *nombre_empresa* no tiene una dependencia funcional completa con la clave, sino que la tiene sólo de una parte de la misma: *cod_empresa*.

El atributo identificado formará parte de una nueva entidad, EMPRESAS, siendo eliminado de la antigua. La clave principal de la nueva entidad será *cod_empresa*.

Para representar la segunda forma normal en el modelo de datos, deberá añadirse un tipo de relación, *Se componen*, y el tipo de correspondencia *1:N*.

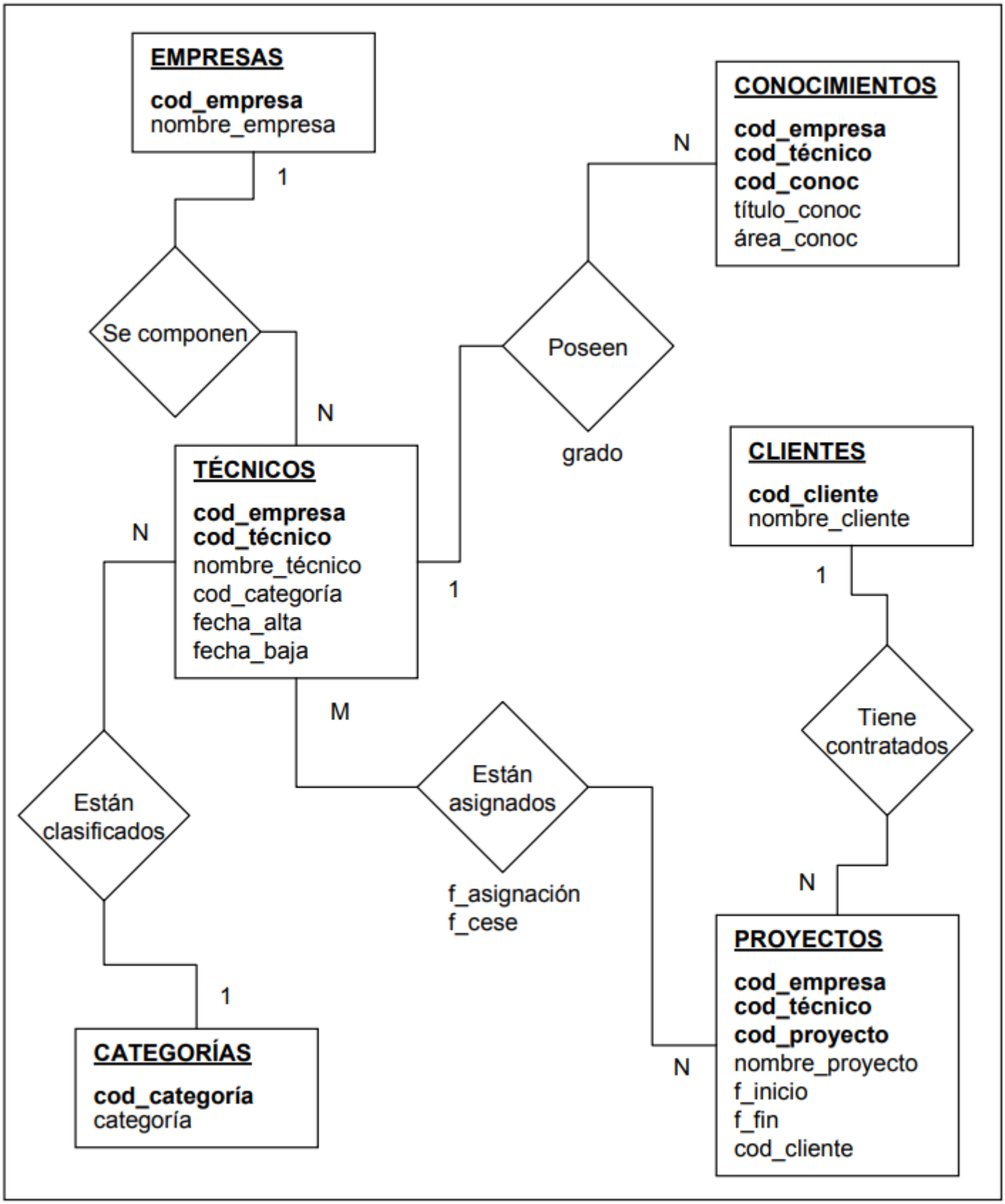


Tercera Forma Normal (3FN)

En la entidad TÉCNICOS de la figura se puede observar que para un *cod_técnico* hay un único *cod_categoría*, es decir, el segundo depende funcionalmente del primero; para un *cod_categoría* hay una única *categoría*, es decir, que este atributo depende funcionalmente del *cod_categoría*; y por último, para un *cod_categoría* hay varios valores de *cod_técnico*. Así pues, la categoría depende transitivamente del *cod_técnico*, por lo que la entidad TÉCNICOS no está en 3FN.

Una vez identificado el atributo categoría que depende de otro atributo distinto de la clave, *cod_categoría*, se formará con él una nueva entidad y se quitará de la antigua. La clave principal de la nueva entidad será el atributo del cual depende *cod_categoría* y en la entidad antigua pasará a ser una clave ajena.

Del mismo modo, puede observarse que la entidad PROYECTOS tampoco está en 3FN, pues el *nombre_cliente* depende de *cod_cliente*, que no forma parte de la clave de la entidad.



Así pues, aparecen dos entidades nuevas en el modelo: CATEGORÍAS y CLIENTES, y sus respectivas relaciones y tipos de correspondencias: *Están clasificados 1:N* y *Tiene contratados 1:N*.

Optimización

El objetivo de esta técnica es reestructurar el modelo físico de datos con el fin de asegurar que satisface los requisitos de rendimiento establecidos y conseguir una adecuada eficiencia del sistema.

Descripción

La optimización consiste en una desnormalización controlada del modelo físico de datos que se aplica para reducir o simplificar el número de accesos a la base de datos.

Para ello, se seguirán alguna de las recomendaciones que a continuación se indican:

- Introducir elementos redundantes.
- Dividir entidades.
- Combinar entidades si los accesos son frecuentes dentro de la misma transacción.
- Redefinir o añadir relaciones entre entidades para hacer más directo el acceso entre entidades.
- Definir claves secundarias o índices para permitir caminos de acceso alternativos.

Con el fin de analizar la conveniencia o no de la desnormalización, se han de considerar, entre otros, los siguientes aspectos:

- Los tiempos de respuesta requeridos.
- La tasa de actualizaciones respecto a la de recuperaciones.
- Las veces que se accede conjuntamente a los atributos.
- La longitud de los mismos.
- El tipo de aplicaciones (en línea / por lotes).
- La frecuencia y tipo de acceso.
- La prioridad de los accesos.
- El tamaño de las tablas.
- Los requisitos de seguridad: accesibilidad, confidencialidad, integridad y disponibilidad que se consideren relevantes.

Reglas de Obtención del Modelo Físico a partir del Lógico

El objetivo de esta técnica es obtener un modelo físico de datos a paritr del modelo lógico de datos normalizado. Para ello es necesario aplicar un conjunto de reglas que conserven la semántica del modelo lógico.

Descripción

Cada uno de los elementos del modelo lógico se tiene que transformar en un elemento del modelo físico. En algunos casos la transformación es directa porque el concepto se soporta igual en ambos modelos, pero otras veces no existe esta correspondencia, por lo que es necesario buscar una transformación que conserve lo mejor posible la semántica, teniendo en cuenta los aspectos de eficiencia que sean necesarios en cada caso.

Transformación de entidades

Una entidad se transforma en una tabla.

Transformación de atributos de entidades

Cada atributo se transforma en una columna de la tabla en la que se transformó la entidad a la que pertenece. El identificador único se convierte en clave primaria.

Si existen restricciones asociadas a los atributos, éstas pueden recogerse con algunas cláusulas del lenguaje lógico, que se convertirán en disparadores cuando éstos sean soportados por el sistema gestor de base de datos.

Transformación de relaciones

Según el tipo de correspondencia:

- **Relaciones 1:N**: se propaga el identificador de la entidad de cardinalidad máxima *1* a la que es *N*, teniendo en cuenta que:
  - Si la relación es de asociación, la clave propagada es clave ajena en la tabla a la que se ha propagado.
  - Si la relación es de dependencia, la clave primaria de la tabla correspondiente a la entidad débil está formada por la concatenación de los identificadores de ambas entidades.
- **Relaciones 1:1**: es un caso particular de las *1:N* y por tanto se propaga la clave en las dos direcciones. Se debe analizar la situación, intentando recoger la mayor semántica posible, y evitar valores nulos.

Las relaciones de agregación se transforman del mismo modo que las *1:N*.

Transformación de relaciones exclusivas

Después de haber realizado la transformación según las relaciones *1:N*, se debe tener en cuenta que si los identificadores propagados se han convertido en claves ajenas de la tabla originada por la entidad común a las relaciones, hay que comprobar que una y sólo una de esas claves es nula en cada ocurrencia. En otro caso, estas comprobaciones se deben hacer en las tablas resultantes de transformar las relaciones.

Transformación de la jerarquía

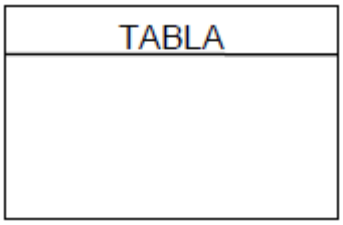
Existen varias posibilidades que deben ser evaluadas por el diseñador a fin de elegir la que mejor se ajuste a los requisitos. Las opciones para tratar la transformación de la jerarquía son:

- **Opción a**: Consiste en crear una tabla para el supertipo que tenga de clave primaria el identificador y una tabla para cada uno de los subtipos que tengan el identificador del supertipo como clave ajena.
  Esta solución es apropiada cuando los subtipos tienen muchos atributos distintos y se quieren conservar los atributos comunes en una tabla. También se deben implantar las restricciones y aserciones adecuadas. Es la solución que mejor conserva la semántica.
- **Opción b**: Se crea una tabla para cada subtipo, los atributos comunes aparecen en todos los subtipos y la clave primaria para cada tabla es el identificador del supertipo.
  Esta opción mejora la eficiencia en los accesos a todos los atributos de un subtipo, sean los comunes al supertipo o los específicos.
- **Opción c**: Agrupar en una tabla todos los atributos de la entidad supertipo y de los subtipos. La clave primaria de esta tabla es el identificador de la entidad. Se añade un atributo que indique a qué subtipo pertenece cada ocurrencia (el atributo discriminante de la jerarquía). Esta solución puede aplicarse cuando los subtipos se diferencien en pocos atributos y las relaciones entre los subtipos y otras entidades sean las mismas. Para el caso de que la jerarquía sea total, el atributo discriminante no podrá tomar valor nulo (ya que toda ocurrencia pertenece a alguna de las entidades subtipo).

Notación

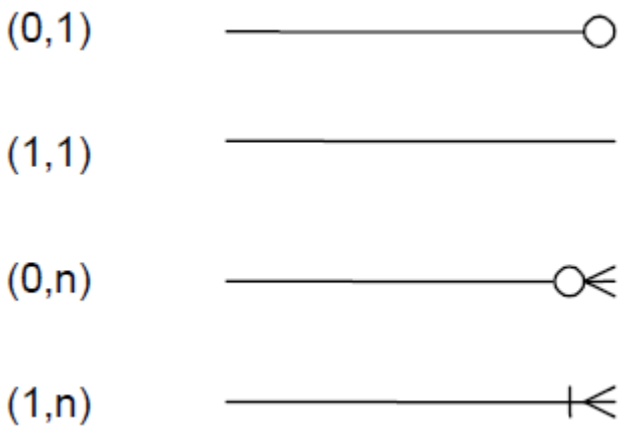
Tabla

La representación gráfica de una tabla es un rectángulo con una línea horizontal que lo divide en dos. La parte superior, de ancho menor, se etiqueta con el nombre de la tabla.



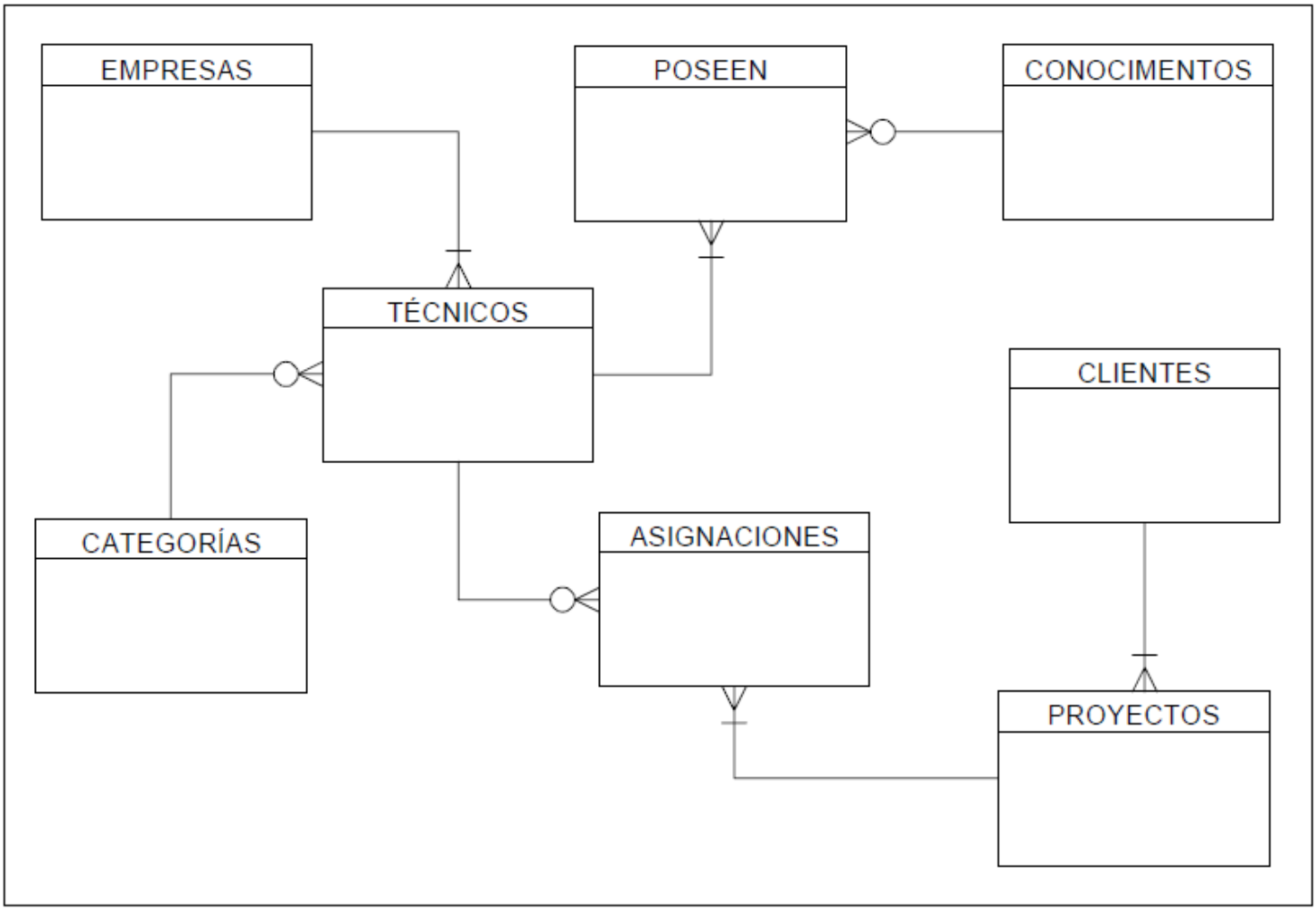
Relación

La relación entre tablas se representa gráficamente mediante una línea que las une. En ella pueden aparecer en sus extremos diversos símbolos para indicar la cardinalidad de la relación, como se muestra a continuación:



Ejemplo.

Sea el diagrama entidad-relación del ejemplo realizado para la Normalización sobre conocimientos de técnicos informáticos y su asignación a proyectos.



El modelo físico de la figura muestra que cada una de las entidades se ha convertido en una tabla, cuyo contenido coincide con los atributos de la entidad. Pero hay dos tablas más: POSEEN, que surge de la relación del mismo nombre y ASIGNACIONES, que se origina a partir de la relación *Están asignados*.

La tabla POSEEN está formada por su atributo *grado*, más *cod_empresa*, *cod_tecnico* y *cod_conoc*. La tabla ASIGNACIONES se forma con los atributos clave *cod_empres*a, *cod_tecnico* y *cod_proyecto* y los propios *f_asignación* y *f_cese*.

La relación entre EMPRESAS y TÉCNICOS era *1:N*, y la cardinalidad de la figura así lo muestra, pues la empresa siempre estará compuesta de uno o varios técnicos. Lo mismo sucede entre CLIENTES y PROYECTOS: un cliente siempre tendrá 1 o varios proyectos contratados.

El caso de CATEGORÍAS y TÉCNICOS es *(0,n)*. Cada técnico es de una categoría y una categoría corresponde, por regla general, a varios técnicos, pero puede existir alguna en la que no encaje ningún técnico (contable, secretaria de dirección, etc.).

La situación del subconjunto TÉCNICOS-POSEEN-CONOCIMIENTOS tienen algo más de complejidad. Un técnico posee normalmente varios conocimientos, pero debe poseer al menos uno para que tenga sentido su situación. La cardinalidad es pues *(1,n)* entre TÉCNICOS y POSEEN. En el otro lado, lo natural es que un conocimiento sea poseído por varios técnicos, sin embargo puede existir algún conocimiento que no sea poseído por ningún técnico, por lo que la cardinalidad es *(0,n)* y dibujada desde la tabla CONOCIMIENTOS a POSEEN.

Por último, en el subconjunto TÉCNICOS-ASIGNACIONES-PROYECTOS, se dispone de: una cardinalidad *(0,n)*, pues a un proyecto estarán asignados uno o más técnicos, pero puede haber algún técnico que, en un momento dado, no esté asignado aún a ningún

proyecto y una cardinalidad *(1,n)*, pues un proyecto siempre tendrá asignado al menos a un técnico, o varios.

(Nota.- La notación utilizada para el ejemplo es la más habitual, pero MÉTRICA Versión 3 no exige su utilización).

Reglas de Transformación

El objetivo de esta técnica es obtener un modelo físico de datos a partir del modelo de clases. Para ello es necesario aplicar un conjunto de reglas de transformación que conserven la semántica del modelo de clases.

Descripción

Cada uno de los elementos del modelo de clases se tiene que transformar en un elemento del modelo físico. En algunos casos la transformación es directa porque el concepto se soporta igual en ambos modelos, pero otras veces no existe esta correspondencia, por lo que es necesario buscar una transformación que conserve lo mejor posible la semántica, teniendo en cuenta los aspectos de eficiencia que sean necesarios en cada caso.

Transformación de clases

Una clase se transforma en una tabla. Lo habitual es que en los modelos con herencia pueden surgir excepciones cuando se apliquen las reglas de transformación propias de la herencia. Además, es posible que dos clases se transformen en una sola tabla cuando el comportamiento de una de ellas sea irrelevante en la base de datos.

Transformación de atributos de clases

Cada atributo se transforma en una columna de la tabla en la que se transformó la clase a la que pertenece. El identificador único se convierte en clave primaria. Además, se deben tener en cuenta las reglas de transformación que se aplican a la herencia de clases.

Si existen restricciones asociadas a los atributos, éstas pueden recogerse con algunas cláusulas del lenguaje lógico, que se convertirán en disparadores cuando éstos sean soportados por el sistema gestor de base de datos.

Transformación de relaciones

Según el tipo de correspondencia:

- **Relaciones M:N**: se transforman en una tabla, cuya clave primaria es la concatenación de los identificadores de las clases asociadas, siendo cada uno de ellos clave ajena de la propia tabla. Si la relación tienen atributos, éstos se transforman en columnas de la tabla.
- **Relaciones 1:N**: existen varias posibilidades:
  - Propagar el identificador de la clase de cardinalidad máxima *1* a la que es *N*, teniendo en cuenta que:
    - Si la relación es de asociación, la clave propagada es clave ajena en la tabla a la que se ha propagado.
    - Si la relación es de dependencia, la clave primaria de la tabla correspondiente a la clase débil está formada por la concatenación de los identificadores de ambas clases.
  - La relación se transforma en una tabla de clave primaria sólo el identificador de la clase de cardinalidad máxima *N* si:
    - La relación tiene atributos propios y se desea que aparezcan como tales.
    - Se piensa que en un futuro la relación puede convertirse en *M:N*.

        - El número de ocurrencias relacionadas de la clase que propaga su clave es muy pequeño (y por tanto pueden existir muchos valores nulos).
    - Al igual que en el caso de relaciones *M:N*, las claves propagadas son claves ajenas de la nueva tabla creada.
- **Relaciones 1:1**: es un caso particular de las *1:N* y se puede tanto crear una tabla o propagar la clave, si bien, en este último caso, la clave se propaga en las dos direcciones. Para decidir qué solución adoptar, se debe analizar la situación, intentando recoger la mayor semántica posible, y evitar valores nulos.
  Las relaciones de agregación se transforman del mismo modo que las *1:N*.

Transformación de relaciones exclusivas

Después de haber realizado la transformación según las relaciones *1:N*, se debe tener en cuenta que si se han propagado los atributos de las clases, convirtiéndose en claves ajenas de la tabla que provenía de la clase común a las relaciones, hay que comprobar que una y sólo una de esas claves es nula en cada ocurrencia. En caso de no propagarse las claves, estas comprobaciones se deben hacer en las tablas resultantes de transformar las relaciones.

Transformación de la herencia

Existen varias posibilidades que deben ser evaluadas por el diseñador a fin de elegir la que mejor se ajuste a los requisitos. Las opciones para tratar la transformación de la herencia son:

- **Opción a**: Consiste en crear una tabla para la superclase que tenga de clave primaria el identificador y una tabla para cada una de las subclases que tengan el identificador de la superclase como clave ajena.
  Esta solución es apropiada cuando las subclases tienen muchos atributos distintos, y se quieren conservar los atributos comunes en una tabla. También se deben implantar las restricciones y/o aserciones adecuadas. Es la solución que mejor conserva la semántica.
- **Opción b**: Se crea una tabla para cada subclase, los atributos comunes aparecen en todas las subclases y la clave primaria para cada tabla es el identificador de la superclase.
  Esta opción mejora la eficiencia en los accesos a todos los atributos de una subclase (los heredados y los específicos).
- **Opción c**: Agrupar en una tabla todos los atributos de la clase y sus subclases. La clave primaria de esta tabla es el identificador de la clase. Se añade un atributo que indique a qué subclase pertenece cada ocurrencia (el atributo discriminante de la jerarquía).
  Esta solución puede aplicarse cuando las subclases se diferencien en pocos atributos y las relaciones que asocian a las subclases con otras clases, sean las mismas. Para el caso de que la jerarquía sea total, el atributo discriminante no podrá tomar valor nulo (ya que toda ocurrencia pertenece a alguna subclase).

Técnicas Matriciales

Las técnicas matriciales tienen como objetivo representar las relaciones existentes entre distintos tipos de entidades, objetos o cualquier otro elemento del sistema.

Se utilizan, principalmente, para analizar la consistencia entre los modelos generados durante el desarrollo, comprobar la trazabilidad con los requisitos especificados por el usuario, etc.

Descripción

Las técnicas matriciales son útiles para representar las relaciones entre elementos comunes de los distintos modelos, tales como entidades/procesos, procesos/diálogos, datos/localización geográfica, y asegurar que los modelos son coherentes entre sí.

Las siguientes son algunas de las matrices empleadas en MÉTRICA Versión 3:

- Procesos/localización geográfica: permite representar la localización geográfica de los procesos de una organización.
- Almacenes de datos/entidades del modelo lógico de datos normalizado: establece las relaciones existentes entre los almacenes de datos y las entidades, y permite verificar que cada almacén de datos definido en el modelo de procesos se corresponde con una o varias entidades del modelo lógico de datos normalizado.
- Atributos de interfaz/atributos de entidades del modelo lógico de datos normalizado: permite verificar que los atributos que aparecen en cada diálogo de la interfaz de usuario forman parte del modelo lógico de datos normalizado.
- Entidades/procesos: permite representar el tratamiento lógico de los procesos sobre los datos del sistema y verificar que cada entidad del modelo lógico de datos normalizado es accedida por algún proceso primitivo representado en el DFD.
- Diálogos/procesos: permite representar los diálogos asociados a un proceso interactivo y verificar que cada proceso interactivo tiene asociado al menos un diálogo.
- Objetos Diagrama de interacción / clases, atributos al modelo de clases: permite verificar que cada mensaje entre objetos se corresponde con un método de una clase.
- Mensajes Diagrama de interacción / métodos, atributos del modelo de clases: permite verificar que una clase tiene capacidad para proporcionar los datos que se soliciten en los mensajes que recibe.
- Evento, acción, actividad de clases / métodos de clases: permite verificar que todo evento, actividad o acción de una clase se corresponde con un método de esa clase.
- Clases/elementos del modelo físico de datos: permite verificar que cada elemento del modelo físico de datos se corresponde con un elemento del modelo de clases.
- Dependencias entre subsistemas/subsistemas: permite representar para cada subsistema, los subsistemas que dependen de él.
- Esquemas físicos de datos / nodos: permite representar la localización física de los datos en los nodos de la arquitectura del sistema, así como verificar que cada esquema del modelo físico de datos está asociado con un nodo del particionamiento físico del sistema de información.

Notación

Dados dos tipos de elementos A y B, su representación será una matriz bidimensional NxM, siendo N el número de elementos de A, y M el número de elementos de B.

En el cruce de una fila y una columna (C), se tendrá el modo en que se relacionan un elemento concreto de A y uno de B.

	B1	B2	...	Bm
A1	C11	C12	...	C1m
A2	C21	C22	...	C2m
...	...	...	...	...
An	Cn1	Cn2	...	Cnm

Bibliografia

- Junta de Castilla y León (SOP_INF_T09_FINAL)
- PAe

Diseño de bases de datos.La arquitectura ANSI/SPARC. El modelo lógico relacional. Normalización. Diseño lógico. Diseño físico. Problemas de concurrencia de acceso. Mecanismos de resolución de conflictos.

Introducción

El diseño lógico es la etapa del proceso de diseño de una BD en la que se obtiene la representación de la estructura de la BD en términos de almacenamiento (tablas). La obtención de esta estructura implica la aplicación de unas reglas de transformación de los elementos previamente existentes en el modelo conceptual de la BD, reglas que se van a describir en este tema.

Por su parte, el diseño físico es la etapa que incluye las acciones de configuración y ajuste del almacenamiento físico y de la seguridad de la BD. El diseño físico es una tarea compleja y dependiente del SGBD utilizados y del uso concreto que se pretenda hacer de la BD diseñada, por lo que en este tema se van a describir la problemática general que se aborda en esta etapa y los criterios de toma de decisiones en esas etapas para resolver un problema.

Finalmente, el tema describirá la necesidad del uso concurrente de las BD y los conflictos que ese uso plantea, así como los mecanismos que los SGBD utilizan para resolverlos.

Diseño Lógico

El diseño lógico es la etapa de creación de la BD en la que se va a traducir el modelo conceptual obtenido en la etapa de diseño conceptual en modelo lógico y un esquema lógico expresado de un modo comprensible para un SGBD. Si consideramos que el SGBD es relacional, es esquema lógico estará expresado en tablas y columnas (o relaciones y atributos).

El diseño lógico se puede dividir en dos etapas:

- **Diseño lógico estándar**: En esta etapa se obtiene un modelo lógico estándar y un esquema lógico estándar, independientes del SGBD comercial en el que se vaya a implementar la BD. El modelo lógico estándar puede expresarse empleando varias técnicas, entre las que cabe citar el Diagrama de Estructura de Datos (DED) y el modelo relacional. El esquema lógico estándar se obtiene utilizando un Lenguaje de Definición de Datos (DDL) estándar, habitualmente el SQL-92 (que es el estándar ISO).
- **Diseño lógico específico**: Utilizando el esquema lógico estándar que se ha obtenido, se estudia su implementación en un SGBD comercial (Oracle, DB2, Sybase, etc). Para ello, habrá de analizarse la compatibilidad del modelo lógico estándr con el modelo lógico específico del SGBD elegido, y proponer un modo de solucionar aquellos aspectos del modelo lógico estándar que no recoge el modelo lógico específico. Una vez realizada esa tarea, se obtiene un esquema lógico específico usando el Lenguaje de Definición de Datos propio del SGBD (normalmente, será un lenguaje SQL con algunas peculiaridades propias de cada SGBD). A la etapa del diseño lógico específico también se la conoce como etapa de implementación de la BD.

Diseño lógico estándar

Tal y como se ha comentado en el punto anterior, el diseño lógico estándar consiste en convertir el modelo conceptual de BD en un esquema lógico estándar, independiente del SGBD que se vaya a utilizar. El esquema lógico estándar será la expresión del modelo lógico estándar utilizando un Lenguaje de Definición de Datos independiente del SGBD (SQL habitualmente).

Las técnicas de modelado conceptual son diferentes de las técnicas de modelado lógico, por lo que habrá que convertir cada elemento presente en el modelo conceptual en elementos expresables en la técnica de modelado lógico que hayamos seleccionado.

En las BD relacionales, es habitual usar el modelo E/R para el modelado conceptual y el modelo relacional para el modelado lógico. En este apartado vamos a estudiar la conversión de los elementos del E/R al modelo relacional.

Transformación de dominios

Un dominio del modelo E/R se transforma en un dominio en el modelo relacional utilizando la sentencia SQL:

```
CREATE DOMAIN
```

Transformación de entidades

Cada entidad del modelo E/R se transformará en una tabla (relación) en el modelo relacional. La tabla tendrá el mismo nombre que la entidad de la que proviene. Vemos un ejemplo a continuación y una entidad de un modelo E/R (conceptual).

Transformando la entidad al modelo relacional, obtendremos lo siguiente:

```
EMPRESA (CIF, Nombre, Dirección)
```

Para obtener el esquema lógico estándar, la tabla se definirá usando la sentencia SQL CREATE TABLE.

```
CREATE TABLE EMPRESA (
     CIF         Código_CIF,
     Nombre      Nombres,
     Dirección   Direccines,
     PRIMARY KEY (CIF)
)
```

Transformación de atributos

Cada atributo de una entidad se transformará en una columna de la tabla (relación) del modelo relacional. Los atributos de las entidades pueden ser de cuatro tipos:

- **Identificadores**: Se transforman en una columna que es la clave primaria de la tabla. En SQL, la condición de clave primaria se representará colocando la cláusula PRIMARY KEY al lado del nombre de la columna en la que se ha convertido el atributo (dentro de la sentencia CREATE TABLE con la que se crea la tabla en la que está incluida la columna).
- **Identificadores alternativos**: Se transforman en una columna a la que se le añade la restricción UNIQUE, lo cual significa que no puede haber valores repetidos en esa columna.
- **Atributos no identificadores**: Se transforman en una columna de la tabla.
- **Atributos multivaluados**: En el modelo relacional una instancia de una relación sólo toma un valor para cada atributo. Por ello, será obligatorio crear una nueva tabla que contenga a la clave primaria de la tabla anterior y al atributo multivaluado, siendo la clave primaria de la nueva tabla la concatenación de los dos atributos y marcándose la clave primaria de la tabla anterior como clave foránea en la nueva tabla.

Transformación de interrelaciones

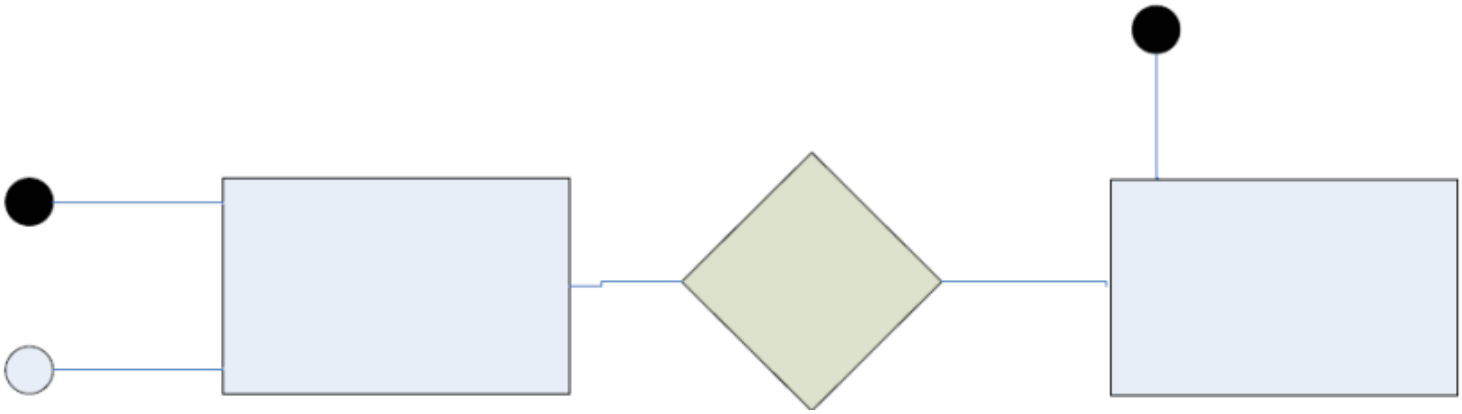
En el modelo relacional, la única unidad de modelado son las tablas, por lo que las interrelaciones del modelo E/R deben transformarse en tablas del modelo relacional (se produce una cierta pérdida de semántica).

El modo de transformar las interrelaciones en tablas depende del tipo de interrelación considerada. Los tipos de transformaciones existentes son:

Transformación de interrelaciones N:M

Una interrelación N:M se va a convertir en una tabla del modelo relacional. La nueva tabla tendrá como columnas a la concatenación de los atributos identificadores de las entidades que estaban unidas por la interrelación.

Una interrelación en el modelo E/R:



```
EMPRESA (CIF, Dirección, ...)
VENDE (CIF, Codigo_producto)
PRODUCTO (Codigo_producto, ...)
```

Si la interrelación tiene atributos en el modelo E/R, cada atributo de la interrelación pasará a ser una columna de la nueva tabla.

Los atributos identificadores de las entidades unidas mediante la interrelación serán claves primarias en las tablas del modelo relacional que representen a sus entidades. Por tanto, esos mismos atributos serán considerados claves foráneas en la tabla que representa a la interrelación.

La condición de calve ajena de una columna se expresará mediante la cláusula FOREIGN KEY en SQL.

```
CREATE TABLE VENDE (
     CIF         Codigo_CIF,
     Producto    Codigo_Producto,
     ...,
     PRIMARY KEY (Codigo_CIF, Codigo_Producto),
     FOREIGN KEY (Codigo_CIF) REFERENCES Empresa,
     FOREIGN KEY (Codigo_Producto) REFERENCES Producto
)
```

Las cardinalidades mínimas de cada entidad participante en la interrelación se van a expresas en el esquema lógico estándar usando la sentencia de SQL.

```
CREATE ASSERTION
```

Transformación de interrelaciones 1:N

Las relaciones 1:N se pueden transformar de dos maneras:

- No crear ninguna tabla que represente a la interrelación y añadir a la tabla que representa a la entidad con cardinalidad n el conjunto de atributos que son clave primaria de la entidad con cardinalidad 1. Este es el modo habitual de realizar la transformación.
- Convirtiendo la interrelación en una tabla, siendo la clave primaria de la tabla el conjunto de atributos identificadores del lado n de la relación. Esta opción se utiliza en los siguientes casos:
    - Cuando se cree que en un futuro la relación se va a transformar en una de tipo N:M (y por tanto, será necesario tener una tabla que represente a esa relación).
    - Cuando la interrelación tiene atributos en el modelo relacional.
    - Cuando la interrelación es optativa para las ocurrencias de las entidades situadas en el lado 1 de la relación (cardinalidad 0:1) y el porcentaje de

ocurrencias interrelacionadas es bajo, lo cual va a significar que en las columnas absorbidas en la tabla n van a existir muchos valores nulos.

Transformación de interrelaciones 1:1

Se realizan del mismo modo que las interrelaciones 1:N, pero teniendo en cuenta que si se decide no crear una tabla que represente a la interrelación, la elección de la tabla a la que se le añaden los atributos del otro extremo es optativa, si bien se suele seguir el siguiente criterio:

- Si una de las dos entidades de la interrelación tiene cardinalidad (0,1) y la otra entidad tiene cardinalidad (1,1), entonces se propagan los atributos identificadores de la tabla (1,1) a la tabla (0,1), evitándose los valores nulos.
- Si las dos tablas tienen cardinalidad (1,1), se puede escoger cualquiera de los dos extremos para propagar la clave. En este caso, la elección puede depender de criterios como las frecuencias de acceso a las tablas.
- Si los dos extremos participan con una cardinalidad (0,1), crear una tabla que represente a la interrelación. El identificador de la tabla podrá ser el identificador de cualquiera de los dos extremos, y los atributos que sean clave primaria en uno de los dos extremos, serán clave foránea en la nueva tabla.

Transformación de interrelaciones con un grado superior a 2

El mecanismo de transformación es igual al de las tablas N:M, se crea una tabla que representa a la relación, y su clave primaria será la concatenación de los atributos identificadores de todas las entidades a las que interrelaciona (lógicamente, tres o más).

Transformación de interrelaciones de dependencia y existencia

El modelo relacional no distingue tipos de relaciones, por lo que las interrelaciones de dependencia y existencia se han de convertir en relaciones del mismo modo que las interrelaciones 1:N. Habitualmente, se propaga la clave de la tabla que representa a la entidad débil a la tabla que representa al entidad fuerte.

Transformación de restricciones de entidades o atributos

En el modelo E/R pueden estar expresadas restricciones de usuario. Estas restricciones se recogen en el esquema lógico estándar del siguiente modo:

- Si la restricción indica un rango de valores, se usa la cláusula BETWEEN.
- Si la restricción implica que un determinado atributo o conjunto de atributos sólo puede tomar un valor de entre los pertenecientes a una lista, se emplea la cláusula IN.
- Si la restricción es de otro tipo, se puede utilizar la sentencia CHECK para comprobar el cumplimiento de la condición fijada, o la sentencia CREATE ASSERTION si la restricción afecta a más de una tabla.

```
CREATE TABLE Decreto
(
     Num_Decreto         Numeros_Decreto,
     Fecha_Aprobacion    Fecha,
     Fecha_Publicacion   Fecha,
     ...,
     CHECK (Fecha_Aprobacion < Fecha_Publicacion)
)
```

Transformación de dependencias de identificación y existencia

La transformación de estas dependencias se realizarán del mismo modo que el de las relaciones 1:M, es decir, no escribiendo ninguna relación que las represente y propagando la clave de la tabla que representa a la entidad fuerte de la tabla que representa a la entidad débil, en la cual jugará el papel de clave foránea.

Para dicha clave foránea, no se admitirán los valores nulos, y se añadirá la condición de borrado y modificación en cascada (la eliminación o modificación de una ocurrencia de la entidad fuerte obligará a la eliminación o modificación de las ocurrencias de las entidades débiles que tiene asociadas).

Si la dependencia es en identificación, entonces la clave primaria de la tabla que representa a la entidad débil se formará mediante la concatenación de la clave primaria de la entidad débil y de la clave propagada que proviene de la entidad fuerte.

Transformación de restricciones en las interrelaciones

Se utilizarán los mismos mecanismos que se han comentado para la transformación de restricciones de las entidades o de sus atributos (usando las condiciones CHECK o CREATE ASSERTION si la restricción afecta una interrelación o a varias).

Transformación de generalizaciones (Relaciones ISA)

Hay tres estrategias para llevar a cabo esta transformación:

- Transformar la entidad y sus subtipos en una sola tabla, la cual tendrá como atributos la concatenación de los atributos de la entidad y de los subtipos. En el ejemplo:

```
EMPLEADO_PUBLICO (DNI, Nombre, Relacion_laboral, Inicio_contrato, Toma_posesion)
```

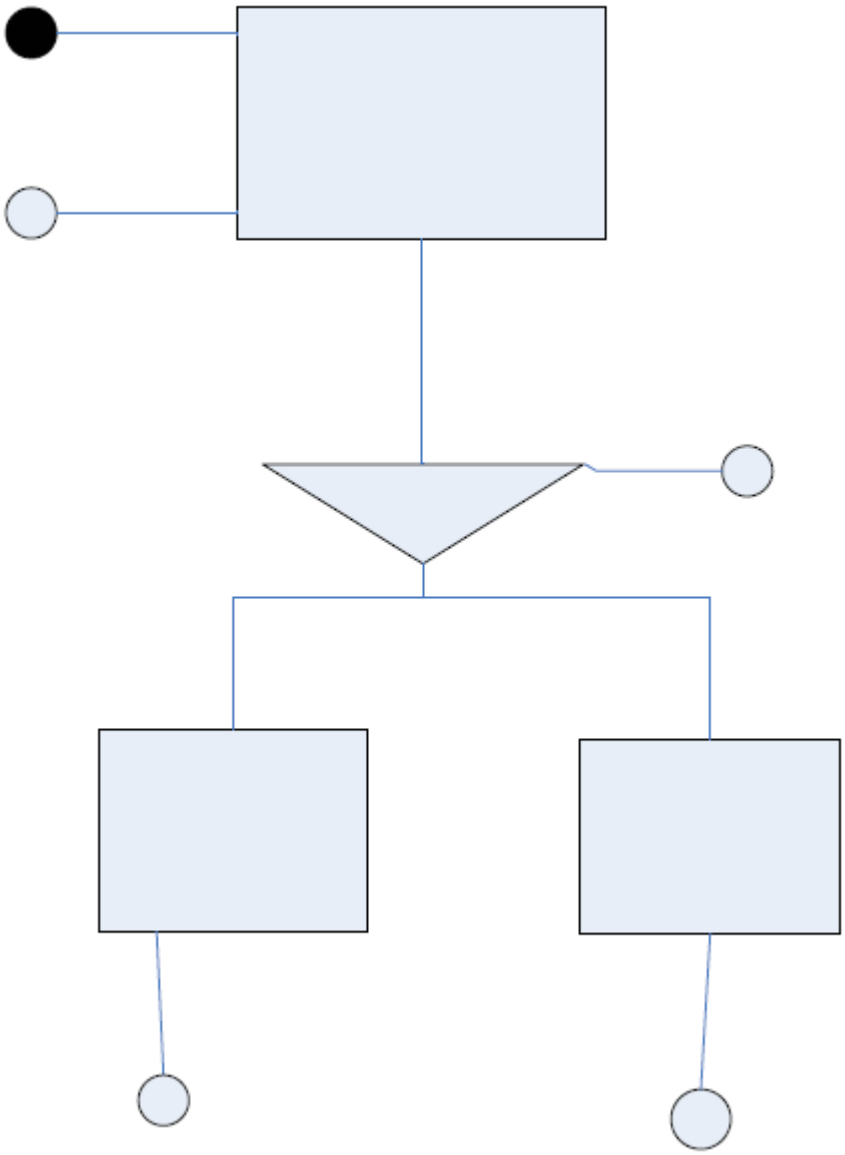
- Crear una tabla para la entidad generalizadora y una tabla por cada subtipo. Cada tabla tendrá como atributos los de su entidad o subtipo correspondiente. Ésta es la opción que mejor mantiene la semántica del modelo E/R. En el ejemplo:

```
EMPLEADO_PUBLICO (DNI, Nombre)
INTERINO (DNI, Inicio_Contrato)
FUNCIONARIO (DNI, Toma_Posesion)
```

- Crear una tabla para cada subtipo. Cada tabla tendrá como columnas los atributos del subtipo al que representa y los atributos comunes (los que posee la entidad generalizadora). En el ejemplo:

```
INTERINO (DNI, Nombre, Inicio_Contrato)
FUNCIONARIO (DNI, Nombre, Toma_Posesion)
```

Mostramos la figura de una relación ISA en el modelo E/R:



Transformación de la dimensión temporal

Se distinguen dos casos:

- Si la dimensión temporal aparece en el modelo E/R como una entidad, se transformará del mismo modo que el resto de las entidades.
- Si la dimensión temporal aparece en forma de atributos de una interrelación, estos atributos se ubicarán en la tabla que les corresponda al transformar la interrelación (bien en la tabla de la interrelación o bien en la tabla hacia la que se hayan propagado claves). Ahora bien, debe considerarse que estos atributos de tipo fecha pueden tener que formar parte de la clave primaria de la tabla en la que se ubiquen en función de la semántica de la situación que se representa.

Transformación de atributos derivados

Los atributos derivados se transformarán en columnas de la entidad a la que pertenezcan (como el resto de los atributos). Además, se establecerá un disparador o un procedimiento almacenado que calcule el valor del atributo cada vez que se inserta una nueva fila en la tabla o cada vez que se modifique en una fila el valor de alguno de los atributos a partir de los cuales se calcula el atributo derivado.

Normalización del esquema obtenido

Finalizada la transformación del esquema conceptual en un esquema lógico, debe aplicarse a dicho esquema lógico un proceso de normalización, evitándose así las anomalías de inserción, modificación y borrado que provoca la redundancia.

Diseño lógico específico

Partiendo del esquema lógico obtenido en el apartado anterior, se elabora un esquema adaptado al sistema gestor de BD que se va a utilizar, creándose las tablas del esquema utilizando el Lenguaje de Definición de Datos propio del cada sistema. En las BDR, el Lenguaje de Definición de Datos (LMD) es el SQL, si bien existen pequeñas variaciones entre el SQL usado en cada sistema, que normalmente incluye pequeñas modificaciones o extensiones del lenguaje SQL (que es el estándar ISO).

En el paso del modelo lógico estándar al modelo lógico específico de cada SGBD puede encontrarse que el modelo lógico específico soporta todos los conceptos del modelo lógico estándar (del modelo relacional) o, por el contrario, que existen determinados aspectos del modelo lógico estándar que el modelo lógico específico no soporta. En este último caso, habrá que realizar un trabajo complementario de adaptación (bien añadiendo programación complementaria en el diccionario de datos del SGBD o bien haciendo que la puesta en práctica de esas restricciones no soportadas por el modelo lógico del SGBD, la lleve a cabo el código de los programas que utilicen los datos de la BD).

Algunos de los aspectos del modelo lógico estándar que pueden tener que adaptarse son los siguientes:

- **Dominios**: El DDL del SGBD puede no incluir ninguna sentencia que nos permita crear dominios (los únicos dominios que reconoce automáticamente son los asociados a los tipos de datos predefinidos en el propio SGBD). En este caso, habrá que elegir una de las dos opciones siguientes:
  - Cuando se especifique la columna (dentro de la sentencia CREATE TABLE), escoger el tipo de datos predefinido que mejor se ajuste, fijar la longitud y añadir alguna restricción CHECK.
  - Crear una tabla de dominio, que contendrá una sola columna, y en la que cada fila será uno de los valores posibles del dominio. Una vez creada esta tabla, crear un procedimiento almacenado que comprueba que los valores que se intentan insertar en la columna son compatibles con el dominio que queremos establecer. La tabla de dominio será estática, es decir, sólo podrá ser modificada por el administrador de la BD. Lógicamente, la opción de construir una tabla de dominio sólo será validad si el dominio a construir es finito.
- **Clave Primaria**: Si el SGBD no incluye una cláusula PRIMARY KEY, debe conservarse la semántica dando los siguientes pasos:
  - Añadir la restricción NOT NULL en los atributos que formen parte de la clave primaria (debe recordarse que una clave primaria no admite valores nulos).
  - Añadir la restricción UNIQUE al conjunto de atributos de la clave primaria (ya que una clave primaria no admite valores repetidos).
  - Añadir a la tabla un índice construido sobre las columnas que forman parte de la clave primaria. Este índice se debe crear al crear la tabla y se debe destruir cuando la tabla sea eliminada.
  - Documentar el esquema con un comentario que indique cuál es la clave primaria.
- **Clave ajena**: Si el SGBD no incluye una cláusula FOREIGN KEY, debe conservarse la semántica dando los siguientes pasos:
  - Añadir la restricción NOT NULL en los atributos de la clave ajena que no admitan nulos (cuando la cardinalidad mínima de la interrelación original fuera de al menos uno).
  - Hacer que los programas que trabajen con la BD implementen las restricciones de clave ajena (integridad referencial).

    - Documentar el esquema con un comentario que indique que una columna o conjunto de columnas son clave ajena.

El resto de los aspectos del modelo lógico estándar no recogidos por el modelo lógico específico del SGBD suelen modelarse empleando procedimientos almacenados o triggers.

Finalizada la etapa del diseño lógico específico, habremos creado un esquema lógico específico en el SGBD. Esto quiere decir que ya tendremos una BD operativa en la que se podrán insertar, eliminar o modificar datos y sobre la cual podremos realizar consultas.

Diseño Físico

El diseño físico es la última etapa del proceso de creación de una BD. El objetivo de esta fase es obtener un esquema interno de la BD que cumpla lo mejor posible los objetivos de funcionamiento de la BD que los usuarios esperan. Más concretamente, se trata de:

- Disminuir el tiempo de respuesta de la BD (tanto el tiempo medio como la respuesta ante los picos de carga)
- Disminuir el espacio de almacenamiento utilizado
- Incrementar la Seguridad de la BD

Estos objetivos del diseño físico no siempre son compatibles entre sí. Por ejemplo, para reducir el tiempo de respuesta de las consultas a una BD, puede ser necesario incrementar la redundancia de los datos (tener los mismos datos almacenados en varias tablas a la vez). Obviamente, esto incrementará el espacio de almacenamiento utilizado. Un buen diseño físico debe tener en cuenta para cada BD las necesidades de uso, establecer unos objetivos concretos, y, cuando estos objetivos sean contradictorios, priorizarlos y alcanzar un nivel de compromiso aceptable entre ellos.

Para llevar a cabo esta etapa, es preciso contar con información precisa sobre muchos aspectos de la BD que se va a crear y de la plataforma en la que se va a trabajar. El diseño físico comienza a realizarse cuando se ha recopilado información suficiente sobre:

- Los recursos software de los que se dispone
- Los recursos hardware de los que se dispone
- El esquema lógico específico de la BD
- Políticas de seguridad de los datos
- Estudio detallado de las aplicaciones que va a utilizar la BD y de las transacciones que van a generar.

El nivel físico de las BD no está estandarizado, por lo que la realización del diseño físico es dependiente del SGBD que se esté utilizando. Cada SGBD relacional definirá su propia estructura de archivos, índices, buffers de memoria, roles de seguridad y objetos de gestión del nivel físico, manipulándose este nivel a través de una extensión del lenguaje SQL estándar específica de cada sistema gestor.

De los dicho en los párrafos anteriores se pueden extraer las siguientes conclusiones:

Al contrario que en el diseño lógico general, el enfoque del diseño físico no puede ser formal, sino casuístico, adaptado a cada SGBD y a cada BD utilizados. No hay recetas universalmente válidas, sino “buenas ideas” (heurísticas) asentadas en conceptos de almacenamiento, arquitectura de computadores, redes o algoritmia. Al no haber una estandarización del nivel físico, esas ideas deben ser puestas en práctica en cada caso concreto, probadas, evaluadas y, si es necesario, refinadas hasta alcanzar la situación final deseada. A este proceso de mejora del diseño físico se le conoce como ajuste de la BD o tunning.

Por último, es preciso comentar que no todos los SGBD tienen el mismo grado de flexibilidad en su nivel físico. En función del grado de manejo que permitan para el diseño físico, podemos distinguir tres tipos de SGBD:

- **Rígidos**: El SGBD fija una estructura interna que apenas admite configuración. Esto asegura la independencia físico/lógica de la BD, pero es poco adaptable a cada situación concreta, lo que puede suponer una pérdida de eficiencia.
- **Flexibles**: El SGBD permite que sea el Administrador de BD el que diseñe toda la estructura interna. El diseño de toda la estructura interna es un trabajo extenso y complejo, y la toma de decisiones del administrador puede afectar a la independencia físico/lógica de los datos. Sin embargo, también es el enfoque más adaptable a cada necesidad concreta, con lo que es la alternativa con la que se podría obtener un mayor grado de eficiencia en el uso de la BD.
- **Semiflexibles**: El SGBD proporciona una estructura inicial configurable a través de un conjunto de parámetros. La modificación de estos parámetros por parte del Administrador de BD permite ir mejorando esa estructura interna, y por ende, el rendimiento de la BD. Ésta opción ofrece un buen compromiso entre eficiencia e independencia físico/lógica, siendo habitual en los SGBD.

Metodología de trabajo para la obtención del diseño físico

Podemos dividir la etapa del diseño físico en tres fases:

- Diseño de la representación física.
  - Análisis de las transacciones.
  - Selección del modo de almacenamiento en memoria secundaria.
  - Creación de índices secundarios.
  - Realización de Agrupamientos de tablas.
  - Realización de procesos de desnormalización.
  - Estimación de la necesidad de espacio en disco.
- Diseñar los mecanismos de seguridad.
  - Diseñar las vistas de los usuarios.
  - Diseñar las reglas de acceso.
- Monitorizar y ajustar el sistema.

Análisis de las transacciones

Para realizar un buen diseño físico es necesario conocer las consultas y las transacciones que se van a ejecutar sobre la BD. Esto incluye tanto información cualitativa, como cuantitativa. Para cada transacción, hay que especificar:

- La frecuencia con que se va a ejecutar.
- Las relaciones y los atributos a los que accede la transacción, y el tipo de acceso: consulta, inserción, modificación o eliminación. Los atributos que se modifican no son buenos candidatos para construir estructuras de acero.
- Los atributos que se utilizan en los predicados WHERE de las sentencias SQL. Estos atributos pueden ser candidatos para construir estructuras de acceso dependiendo del tipo de predicado que se utilice.
- Si es una consulta, los atributos involucrados en el join de dos o más relaciones. Estos atributos pueden ser candidatos para construir estructuras de acceso.
- Las restricciones temporales impuestas sobre la transacción. Los atributos utilizados en los predicados de la transacción pueden ser candidatos para construir estructuras de acceso.

Selección de la organización del almacenamiento en memoria secundaria

Las BD van a almacenar la información en dispositivos de almacenamiento secundarios (discos o cintas), los cuales se caracterizan por tener mayor capacidad que la memoria

principal y por la no volatilidad de los datos. Sin embargo, son mucho más lentos que la memoria principal a la hora de recuperar información, por lo que es preciso realizar un estudio detallado sobre el modo de organizar la información en ellos, de modo que consigamos un rendimiento en tiempo de acceso ajustado a cada necesidad de uso de la BD.

Las alternativas de organización consisten básicamente en la elección del tipo de fichero o estructura de datos más adecuado para cada caso. En este apartado nos limitaremos a estudiar las ventajas y las desventajas de las más habituales, pero sin entrar en una descripción detallada de esas estructuras.

Ficheros secuenciales

Organizados de tal manera que cada registro es adyacente al siguiente registro. Esta relación de adyacencia puede ser física (direcciones físicas consecutivas) o lógica (haciendo que cada registro contenga un puntero al siguiente registro).

Los ficheros secuenciales no permiten el acceso directo a los datos, por lo que el acceso a los registros se realiza en el mismo orden en el que fueron introducidos en el fichero.

Ficheros secuenciales indexados ISAM

Es una estructura de fichero indexado en el que los registros se agrupan en bloques, y en el interior de dichos bloques están organizados secuencialmente.

El índice que se crea sobre el fichero contiene apuntadores a las direcciones de inicio de cada bloque, y el acceso a datos a través del índice se realiza de la siguiente manera:

1. Se localiza el índice de la clave que cumpla la condición de búsqueda.
2. Se accede al bloque apuntado por el nodo que contenía a la clave anterior.
3. Una vez dentro del bloque, el registro deseado se busca secuencialmente.

La organización de ficheros ISAM mantiene el equilibrio entre el tamaño de los índices y el tiempo de acceso a los registros.

Como el tamaño de los bloques está limitado, en estos ficheros hay una zona de desbordamiento en que se van a almacenar los registros que no se pueden guardar en el bloque que les corresponde cuando éste ya está lleno.

El uso de la zona de desbordamiento disminuye el rendimiento del sistema, puesto que se accede a ella tras buscar al registro en el bloque en el que le correspondería estar, y porque la búsqueda en la zona de desbordamiento es secuencial. Cuando el área de desbordamiento es muy grande, se reorganiza el fichero, realizándose una nueva división de bloques, ubicando a todos los registros en los bloques y reorganizando el índice de apuntadores a bloques.

Árboles-B

Estructura de indexación en forma de árbol equilibrado. El hecho de ser equilibrados (misma altura en todas sus ramas) permite minimizar el número de accesos a disco cuando se realiza una búsqueda: las búsquedas rápidas son una característica que distingue a los árboles-B.

En cuanto al almacenamiento, los árboles-B consiguen una gestión del espacio razonablemente buena, ya que si el árbol es de orden n, cada nodo debe tener al menos n/2 claves (es decir, como mucho se desaprovecha la mitad del espacio de almacenamiento del índice, lo cual es fácilmente asumible con los recursos de almacenamiento de que se dispone hoy en día).

Ficheros de acceso aleatorio empleando técnicas de Hashing

En estos ficheros se accede directamente a los registros mediante el valor de su clave (siendo la clave uno o más de los campos del registro). Para ello, se dispone de una función “hash” o de mapeado que permite calcular la dirección del registro a partir del valor de la clave. Este sistema es el que más rápido permite realizar una búsqueda de un registro concreto, pero sólo funciona para resolver consultas exactas (con todo el valor de la clave). Si la búsqueda es por rango o por patrón (por ejemplo, LIKE ‘%a’), no se puede aplicar la función hash.

Criterios de elección entre las estructuras

La elección de una estructura de organización dependerá del uso que se realice de los datos almacenados. La siguiente table recoge las situaciones en las que se suele preferir cada estructura:

Estructura	Situación
Ficheros secuenciales	Archivos pequeños (es más costoso gestionar el índice que realizar las búsquedas secuenciales). Recuperaciones masivas de los datos (habrá que recorrer todos los datos). Acceso muy infrecuente a los datos y existencia de una sobrecarga de almacenamiento en el sistema (Manteniendo el fichero secuencial se ahorra el espacio de almacenamiento del índice. Como el acceso a los datos es muy infrecuente, el rendimiento de la BD no se ve muy afectado).
HASH	Búsquedas exactas (con todo el valor de la clave).
Ficheros indexados (ISAM, Árboles-B o variantes)	En todos aquellos casos en los que no convenga utilizar ficheros secuenciales ni técnicas de HASH.

La elección entre un fichero ISAM y un árbol-B se realiza considerando los siguientes factores:

- Frecuencia de las actualizaciones de los datos: Si la frecuencia de las actualizaciones es alta, debe elegirse un árbol-B, ya que los ficheros ISAM se irán degradando al irse añadiendo registros a la zona de desbordamiento.
- Elevado número de consultas recurrentes: Si se realizan muchas consultas simultáneas sobre los datos indexados, el fichero ISAM debe ser la estructura elegida, ya que al ser su índice estático, no se bloquea (facilita el acceso concurrente).
- Si se deben tener en cuenta los dos factores o no se conoce bien el entorno de explotación de la BD (existen dudas sobre el número de usuarios, la frecuencia de acceso a datos, etc), la estructura elegida será el árbol-B, ya que es la más adaptable de las dos y sus aspectos desfavorables respecto a los ficheros ISAM tienen poca repercusión en el funcionamiento del sistema.

Creación de los índices secundarios

Las BD relacionales indexan a cada tabla por su clave primaria, creándose automáticamente el índice de clave primaria en el mismo momento en el que se crea la tabla. Sin embargo, es frecuente añadir índices adicionales a las tablas, para que hagan más rápidas las consultas que se realizan sobre determinados campos de esas tablas.

La introducción de un índice secundario en una tabla repercute negativamente en los tiempos de ejecución de la inserción o eliminación de registros, y supone un incremento del espacio de almacenamiento necesario para la tabla. El administrador de BD debe ponderar en cada caso las pérdidas y ganancias de introducción de un nuevo índice, para

así determinar si su creación es interesante. Algunas situaciones que hacen interesante la indexación son:

- Atributos de la relación a los que se accede con mucha frecuencia.
- Claves foráneas de la relación sobre las que es habitual realizar joins.

Algunas situaciones que desaconsejan la introducción de índices secundarios son:

- Tablas con pocas filas: el recorrido secuencial de las tablas sería muy breve, y la reducción del tiempo de acceso obtenida de la introducción del índice es muy escasa.
- Atributos cuyo valor se modifica muy a menudo: Los índices utilizan como clave de búsqueda los valores de los atributos indexados. Si la modificación de esos valores es muy frecuente, habrá que reconstruir el índice cada poco tiempo, lo cual puede suponer un coste elevado si la tabla tiene muchas filas.
- Atributos con valores poco selectivos: Los atributos en los que es muy habitual la repetición de su valor en distintas filas de la tabla no son buenos candidatos para la indexación. Después de recorrer el índice y localizar su clave de búsqueda, tendríamos todavía muchos registros con el mismo valor de clave, y habría que recorrer secuencialmente esos registros para encontrar el que se busca; en este caso, la introducción del índice no supondría ningún beneficio importante desde el punto de vista del tiempo de acceso. Por ejemplo, si tenemos una tabla de personas y deseamos buscar una persona en concreto, no tiene sentido indexar por un campo ‘Sexo’, ya que aproximadamente la mitad de los registros de la tabla tendrían el valor ‘hombre’ y la otra mitad el valor ‘mujer’.

Realización de agrupamientos de tablas (clustering)

El clustering o agrupación de tablas es una técnica consistente en almacenar un grupo de tablas en una misma área de memoria secundaria. De este modo, los accesos simultáneos a las tablas agrupadas no obligan al SGBD a la búsqueda de los datos en zonas lejanas del almacenamiento secundario, lo que reduce el tiempo de resolución de esos accesos.

El clustering será una técnica a considerar si esos accesos simultáneos son frecuentes.

Realización de procesos de desnormalización

El proceso de desnormalización consiste en introducir redundancias en un esquema lógico previamente normalizado (típicamente en 3FN o en FN de Boyce-Codd). Aunque ésta es una decisión de nivel lógico, se suele tomar por motivos de eficiencia de la BD (repetir datos en varias tablas evita navegar entre ellas para encontrarlos), es decir, la decisión viene motivada por cuestiones de implementación.

La desnormalización ralentiza las actualizaciones de datos, hace perder flexibilidad al esquema lógico (puede afectarnos si decidimos añadir nuevas tablas, por ejemplo) y complica la implementación de la BD y su documentación. Por tanto, es una alternativa que sólo se debe utilizar cuando el resto de las técnicas de diseño físico no nos proporcionan un rendimiento de las consultas de la BD que sea aceptable.

Entre las acciones de desnormalización que se pueden realizar, se pueden destacar:

- Introducir atributos derivados: Los atributos derivados son aquellos cuyo valor se puede obtener realizando un conjunto de operaciones sobre atributos ya existentes en la BD. La introducción de un atributo derivado ayuda a obtener ese valor rápidamente en una consulta, puesto que no hay que realizar las acciones de cálculo del mismo, pero incrementa los costes de almacenamiento y de actualización de las tuplas de la BD (cada vez que se modifique el valor de un atributo que interviene en su cálculo, habrá que recalcular el valor del atributo derivado).
- Fusionar tablas involucradas en relaciones uno a uno: Si el acceso conjunto a ambas tablas es muy habitual, puede ser interesante crear una nueva tabla cuyos atributos

sean la concatenación de los atributos de las dos tablas, lo cual puede reducir su tiempo de acceso. Sin embargo, este cambio perjudicará a las operaciones join que se realicen entre cualquiera de esas dos tablas y una tercera, ya que las filas de la nueva tabla que se han de leer ocuparán más en memoria (la tabla fusionada tiene más campos), por lo que se recuperarán menos registros en cada lectura, lo cual obligará a un número de accesos a disco superior.
- Introducir atributos duplicados en relaciones 1:N: Consiste en añadir atributos no clave de la tabla con cardinalidad 1 en la tabla con cardinalidad N. Así, cuando se realiza una operación de join entre las dos tablas que sólo involucra a los atributos duplicados, no será necesario recorrer la tabla con cardinalidad 1.

Determinación del espacio de almacenamiento necesario

Una vez obtenido el esquema de implementación definitivo, el administrador de la BD debe determinar el espacio de almacenamiento necesario para la BD. La cantidad de espacio que se determine será una estimación basada en el estudio de la cantidad de datos ya existente y que hay que cargar en la BD y en las estimaciones de crecimiento futuro de la BD.

La estimación del espacio puede verse afectada también por el nivel de seguridad que se desee para el almacenamiento de datos y por el grado de disponibilidad de la BD. Por ejemplo, si uno de los requisitos de la BD es que su disponibilidad sea 24×7 (24 horas al día, 7 días a la semana), es posible que se necesite usar una estructura de almacenamiento secundario en forma de RAID 1+0 o 0+1, lo que duplicaría el espacio de almacenamiento necesario para la BD.

Diseño de los mecanismos de seguridad de los datos

Creación de las vistas de los usuarios

Las vistas son un elemento de la BD que permiten a los usuarios ver los datos de una determinada forma (corresponden al nivel externo en la arquitectura ANSI/SPARC). El uso de las vistas permite:

- Reducir la complejidad del esquema lógico global, manteniendo un esquema único para todos los usuarios y adaptando ese esquema a las necesidades que tenga cada tipo de usuario.
- Mejorar el nivel de seguridad de la BD, ya que los usuarios no verán más datos de las tablas que aquellos que están incluidos en la vista.

Fijar las reglas de acceso de los usuarios

En esta fase, se establecen los perfiles de usuario, donde cada perfil es el conjunto de acciones que cada tipo de usuario va a poder hacer sobre cada elemento o conjunto de elementos de la BD (tablas, filas, unidades lógicas de almacenamiento, etc.)

Monitorización y ajuste del sistema

El diseño físico no se debe concebir como un proceso secuencial que finaliza cuando se ha encontrado una configuración válida. Las BD pueden sufrir variaciones a lo largo del tiempo, tanto en su tamaño, como en su esquema lógico global o en el uso que se desee hacer de ellas, lo que obligará a una monitorización continua de su rendimiento (para comprobar si se va degradando) y a la introducción de ajustes que permitan adaptarse a los cambios sufridos por la BD o solventar las pérdidas de eficiencia que se hayan producido.

Cada proceso de ajuste supone en cierta medida una reconstrucción del diseño físico que se haya realizado, por lo cual esta etapa de diseño volvería a comenzar de nuevo.

La gestión de la concurrencia

Veamos la gestión de la concurrencia en los SGBD. Para ello, empezaremos introduciendo la necesidad de dicha gestión y el concepto de transacción, pasando posteriormente a abordar los problemas que plantea la concurrencia (Lectura Fantasma, Lectura Sucia, etc.) y los mecanismos de resolución existentes.

Necesidad de gestión de la concurrencia

Los SGBD deben poder mantener la integridad de los datos que almacenan. Durante la vida de una BD, existen secuencias de acciones de escritura y lectura que pueden originar que la BD quede en un estado inconsistente si no se ejecutan todas las acciones de esa secuencia.

Para evitar este tipo de problemas, los SGBD utilizan las transacciones, que son unidades lógicas de proceso que se componen de una secuencia de acciones cuya ejecución es atómica, o se ejecutan todas o no se ejecuta ninguna. En un punto intermedio de una transacción, el estado del BD puede ser inconsistente, siendo siempre consistente al principio y al final de la transacción.

El elemento del SGBD encargado de conseguir este objetivo es el Gestor de Transacciones (transaction manager), el cual gestiona las peticiones de los usuarios en forma de transacciones.

Además, los SGBD que existen en la actualidad ofrecen la posibilidad de ser usados en modo multiusuario, lo que implica que múltiples usuarios pueden trabajar a la vez con el sistema como si fuera un recurso dedicado, sin apercibirse de la presencia de otros usuarios. Para lograr este efecto, los tiempos de respuesta del SGBD a todos los usuarios deben ser reducidos, lo que obliga a realizar una ejecución concurrente (simultánea) de las transacciones de cada usuario.

La ejecución atómica que caracteriza a las transacciones no garantiza que las pertenecientes a un programa de un usuario no puedan interferir en las transacciones que pertenecen a otros programas y que se están ejecutando al mismo tiempo. Conseguir que las transacciones no interfieran con el funcionamiento de otras transacciones se conoce como aislamiento de la transacción (isolation), y el conjunto de problemas que plantea conseguir ese aislamiento en un entorno multiusuario es conocido como el problema de la gestión de la concurrencia.

Además, durante el transcurso de una transacción, pueden producirse problemas en el SGBD que no permitan que la transacción concluya con éxito, lo que, en virtud del principio de “todo o nada” que acompaña a las transacciones, obliga al SGBD a estar preparado para deshacer las acciones que una transacción no concluida ha realizado sobre una BD. Es el problema de la recuperación y se gestiona añadiendo una nueva propiedad a las transacciones, que es la persistencia, y que consiste en que las modificaciones de datos que se realizan durante una transacción no se almacenan en la BD hasta que la transacción ha finalizado con éxito.

Resumiendo, una transacción ha de tener las siguientes propiedades:

- **Atomicidad**: Las acciones de una transacción se ejecutan todas o ninguna.
- **Consistencia**: La BD se encuentra en un estado consistente antes de la ejecución de la transacción y debe estar en un estado consistente cuando la transacción termine.
- **Aislamiento**: La ejecución de una transacción no debe interferir en la ejecución de otras transacciones, la transacción debe ejecutarse como si estuviera aislada.
- **Persistencia**: Los efectos de una transacción no son permanentes en la BD hasta que la transacción ha finalizado con éxito.

Manejo de transacciones en el SGBD

El gestor de transacciones es el componente funcional del SGBD que planifica y controla la ejecución de transacciones concurrentes en un BD.

Un gestor de transacciones se puede dividir conceptualmente en los siguientes elementos:

- Un contenedor de entrada: Al que llegan las transacciones que se deben ejecutar.
- Un planificador (scheduler): Que determina el orden en el que las transacciones de la cola van a ser ejecutadas. Cuando el planificador da paso a una transacción, ésta es enviada al gestor de datos, desde el cual se realizarán las operaciones de la transacción sobre la BD.
- Un contenedor de salida: Al que llega la transacción cuando ha terminado de ejecutarse. Si la transacción ha finalizado correctamente, el gestor de transacciones realizará una operación commit, haciendo las modificaciones de la transacción persistentes en la BD.

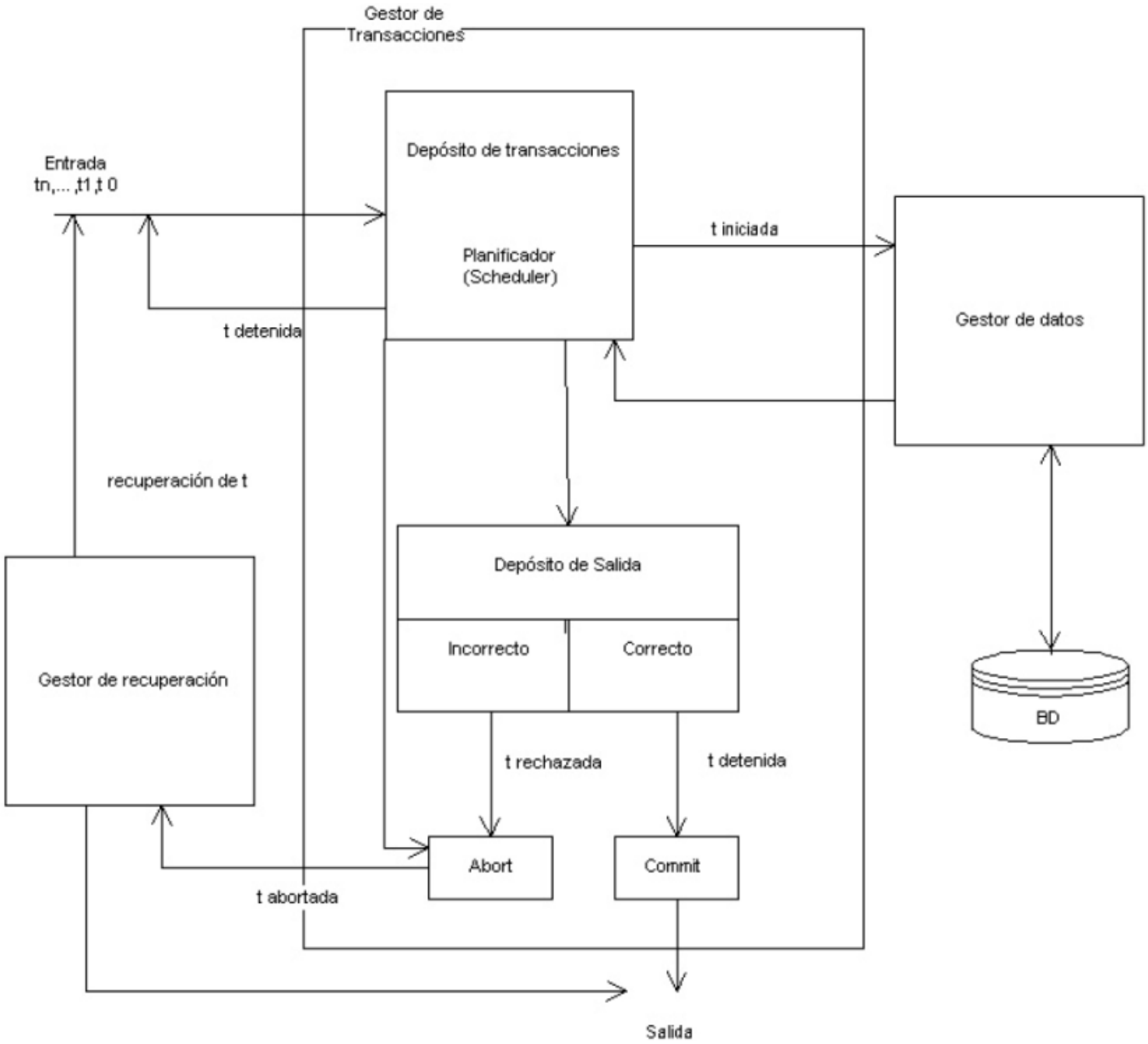
Si la transacción no ha finalizado correctamente debido a algún problema, el gestor de transacciones ejecuta una operación abort y envía la transacción a un gestor de recuperación, el cual deshace las operaciones de la transacción y devuelve los datos al estado en que estaban antes de iniciarse la transacción. La transacción abortada será devuelta al contenedor de salida para volver a ser ejecutada.

Con el fin de incrementar el rendimiento de las BD en un entorno multiusuario, las transacciones no se ejecutan secuencialmente una detrás de otra. El modelo de ejecución secuencial de las transacciones podría originar una situación en la que un programa corto se quedase bloqueado a la espera de la finalización de un programa largo.

En este caso, el usuario del programa debería sufrir un gran tiempo de espera para la realización de un conjunto reducido de operaciones, con lo que no percibiría a la BD como un recurso dedicado.

Para evitar estas situaciones, las transacciones se despachan concurrentemente, es decir, sus acciones se van a llevar a cabo de forma entrelazada. Sin embargo, el entrelazado de acciones de las transacciones puede originar conflictos, algunos de los cuales van a ser comentados en el punto siguiente.

La siguiente figura representa a un gestor de transacciones:



Problemas de la concurrencia

El problema de la actualización perdida

Problema de concurrencia que ocurre cuando:

Dos transacciones T1 y T2 trabajan en paralelo e intentan modificar el valor del mismo objeto de la BD. Ambas transacciones leen el valor del objeto antes de que la otra lo actualice.

En este caso, cada una de las transacciones modificará el valor del objeto en memoria y tratará posteriormente de escribirlo en la BD. El valor que se almacenará en la BD será el que tiene el objeto en la transacción que escriba más tarde, ya que el valor que ha escrito la primera transacción se sobrescribirá. Así, la actualización realizada por la transacción que ha escrito primero quedará sin efecto.

Transacción 1	Tiempo	Transacción 2
Inicio	1	
r(x)	2	
	3	Inicio
	4	r(x)
u(x)	5	
	6	u(x)
w(x)	7	
Fin	8	
	9	w(x)
	10	Fin

NOTA: u(x) significa UPDATE, actualización de x.

El problema de la lectura sucia

Se produce cuando:

Una transacción T1 lee un valor de un objeto que ha sido modificado por otra transacción T2 y que todavía no se ha hecho persistente en la BD.

La transacción T2, que había modificado los datos no termina correctamente. En este caso, la transacción T1 está trabajando con un valor que es inconsistente.

Debe recordarse que los datos pueden encontrarse en un estado inconsistente en el transcurso de una transacción, por lo que la transacción primera puede haber leído dichos datos inconsistentes, utilizándolos para completar su secuencia de acciones.

Transacción 1	Tiempo	Transacción 2
Inicio	1	
r(x)	2	
u(x)	3	
w(x)	4	
	5	Inicio
	6	r(x)
	7	u(x)
Abort	8	
	9	w(x)

El problema de la lectura fantasma o lectura no repetible

Puede aparecer cuando:

Una transacción T1 está leyendo un conjunto de datos al tiempo que una transacción T2 los está modificando.

La transacción T1 lee algunos de los datos antes de que sean modificados por T2 y otros después de ser modificados por T2.

El resultado de la operación que realiza T1 se ve afectado por el hecho de combinar datos no actualizados por T2 con datos actualizados por T2.

Transacción 1	Tiempo	Transacción 2
Inicio	1	
s:=0	2	
r(x)	3	
r(y)	4	
s:= s + x	5	
s:= s+y	6	
	7	Inicio
	8	r(z)
	9	z:= z-10
	10	w(z)
	11	r(x)
	12	x:= x + 10
	13	w(x)
	14	commit
	15	Fin
r(z)	16	
s:=s + z	17	
Fin	18	

En este caso, el valor ‘s’ es la agregación de los valores x, y, z. La acción de la transacción T2 no modifica el valor de esa agregación, sin embargo, la lectura de valores con diferentes versiones ha provocado que T1 ofrezca un valor de s que es equivalente a x+y+x-10.

Mecanismos de resolución de conflictos

Como ya hemos comentado en el punto anterior, la labor fundamental de un gestor de transacciones es manejar las transacciones que generan los programas de los usuarios de tal modo que se mantenga la consistencia de la BD. En un entorno multiusuario, el gestor de transacciones deberá conseguir mantener la consistencia cuando se ejecutan entrelazadamente las acciones que componen las transacciones de los distintos programas.

Este orden de ejecución lo establece el planificador (scheduler) del gestor de transacciones, llamándose plan de ejecución a cada posible secuencia de ejecución de un conjunto de acciones. Lógicamente, existen múltiples planes de ejecución posibles, siendo interesantes los planes que cumplen la condición de serializabilidad; un plan de ejecución A es serializable para un conjunto de transacciones T si existe un plan de ejecución secuencial A’ que es equivalente a A.

O dicho de un modo menos formal, un plan es serializable si el resultado de la ejecución entrelazada de las acciones de esas transacciones es equivalente al resultado de ejecutar las transacciones en serie.

Sin embargo, la comprobación “a priori” de la serializabilidad de un plan es muy costosa (problema NP-completo), por lo que el gestor de transacción no trata de determinar si un plan de ejecución es serializable; en lugar de eso, el gestor de transacciones emplea diversos mecanismos de ejecución de las acciones que o eliminan los conflictos debidos a la concurrencia o permiten recuperar el estado consistente de la BD cuando se producen.

Podemos considerar los métodos de resolución de conflictos divididos en dos tipos:

- **Métodos pesimistas**: Basados en la suposición de que los conflictos entre transacciones concurrentes ocurren con alta frecuencia, lo que obliga a realizar una serie de acciones para manejar esos conflictos.
- **Métodos optimistas**: Basados en la suposición de que los conflictos entre transacciones se producen con escasa frecuencia.

Métodos pesimistas

Concepto de bloqueo y técnicas basadas en bloqueos

Este mecanismo de resolución de conflictos se basa en el concepto de bloqueo, que consiste en que cuando una transacción T1 necesita realizar alguna acción sobre un objeto de la BD, debe solicitar al SGBD una reserva de ese objeto, no pudiendo ejecutar la acción hasta que la reserva no se ha producido. El SGBD sólo concederá esa reserva si otra transacción T2 no mantiene el mismo objeto reservado (en otras palabras, mientras una transacción tenga a un objeto de la BD reservado, el acceso al resto de las transacciones puede considerarse bloqueado).

En un momento dado, bien durante su transcurso o bien a su finalización, la transacción terminará las acciones que necesite hacer sobre el objeto de la BD, deshaciendo la reserva y quedando el objeto disponible para otras transacciones.

Las acciones de bloqueo o desbloqueo no tienen porqué ejecutarse inmediatamente antes y después de la acción durante la cual queremos que el objeto esté bloqueado. En el transcurso de una transacción se hacen múltiples lecturas y escrituras, pudiendo realizarse varias de ellas sobre el mismo objeto. Si se bloquease y desbloquease antes y después de cada acción, se incurrirá en un coste de gestión de los bloqueos muy alto, lo que repercutirá en el rendimiento de la BD (en sus tiempos de respuesta). Por tanto, una política de gestión de bloqueos debe llegar a un buen compromiso entre dos aspectos de signo contrario:

- Tiempo que el objeto (los datos) están bloqueados: Cuanto mayor sea, más largo es el tiempo durante el cual otras transacciones no pueden realizar sus acciones sobre el conjunto de datos, lo que supone que han de estar a la espera del desbloqueo, retardándose su finalización.
- Tiempo empleado en gestionar los bloqueos: Realizar muchas operaciones de bloqueo/desbloqueo repercute en el rendimiento de la BD, ya que hay que almacenar esos bloqueos, comprobar su compatibilidad, etc.

Así pues, la solución consisten en encontrar la manera de ejecutar un número elevado de acciones entre dos bloqueos, consiguiendo a la vez que la distancia entre bloqueos sea razonablemente pequeña.

Considerando el tipo de acción que se impide, podemos distinguir dos clases de bloqueo:

- Bloqueo de escritura (write-lock o wlock): Impide que otras transacciones modifiquen un dato.
- Bloqueo de lectura (read-lock o rlock): Impide que otras transacciones lean un dato.

La distinción entre bloqueos de escritura y bloqueos de lectura se establece porque interesa que el gestor de transacciones pueda hacer un manejo diferenciado de cada tipo:

- Un bloqueo de lectura realizado por una transacción es compatible con un bloqueo de lectura de otra transacción. Dicho de otro modo, no existe ningún problema porque dos transacciones lean un objeto a la vez (y evitar esperas innecesarias supone incrementar el rendimiento de la BD).
- Un bloqueo de escritura es incompatible con cualquier otro tipo de bloqueo, ya que no se puede permitir que una transacción lea un dato cuando otra transacción lo está modificando (como ya vimos en los problemas de la concurrencia) y, desde luego, un objeto no puede ser sobrescrito a la vez.

Así pues, se puede establecer la siguiente tabla de compatibilidades entre bloqueos de lectura y escritura:

	read_lock	write_lock
read_lock	**SI**	NO
write_lock	NO	NO

Los SGBD también permiten manejar diferentes granularidades de los bloqueos: Un bloqueo sobre distintas clases de objetos de la BD, como por ejemplo, una fila, un conjunto de filas o una tabla. De este modo las transacciones pueden efectuar reservas del tamaño estrictamente necesario, no bloqueando innecesariamente partes de los objetos que pueden ser utilizadas simultáneamente por otras transacciones (ejemplo, una transacción puede necesitar hacer un bloqueo de escritura sobre las primeras n filas de una tabla, lo cual no debe impedir que otras transacciones realicen acciones de lectura o escritura sobre las n+1 restantes).

Cuanto menor sea el tamaño de los objetos bloqueados (granularidad más fina), menor será el número de esperas que los bloqueos originarán. Sin embargo, una granularidad fina genera más situaciones de interbloqueo que una granularidad gruesa, por lo que será necesario hacer una selección cuidados del nivel de granularidad empleado.

EL PROBLEMA DEL INTERBLOQUEO

El uso de técnicas basada en bloqueo para la gestión de transacciones concurrentes pueden conducir al problema denominado interbloqueo (o deadlock o abrazo mortal). El interbloqueo es una situación que se produce cuando dos transacciones se quedan esperando indefinidamente por un recurso que la otra transacción tiene bloqueada, de tal manera que ninguna de las dos accede nunca al recurso que la otra ha reservado.

Por tanto, el gestor de transacciones deberá implantar mecanismos que le permitan detectar esas situaciones de interbloqueo y resolverlas, existiendo tres tipos de estrategias de resolución:

- La **predicción**: Consiste en la no generación de transacciones que puedan producir interbloqueo. Esto obligaría a analizar todas las transacciones a priori, lo cual produciría una gran sobrecarga en el sistema (todos los algoritmos de predicción son exponenciales, y por tanto no utilizables en un sistema con altas exigencias de rendimiento como es una BD).

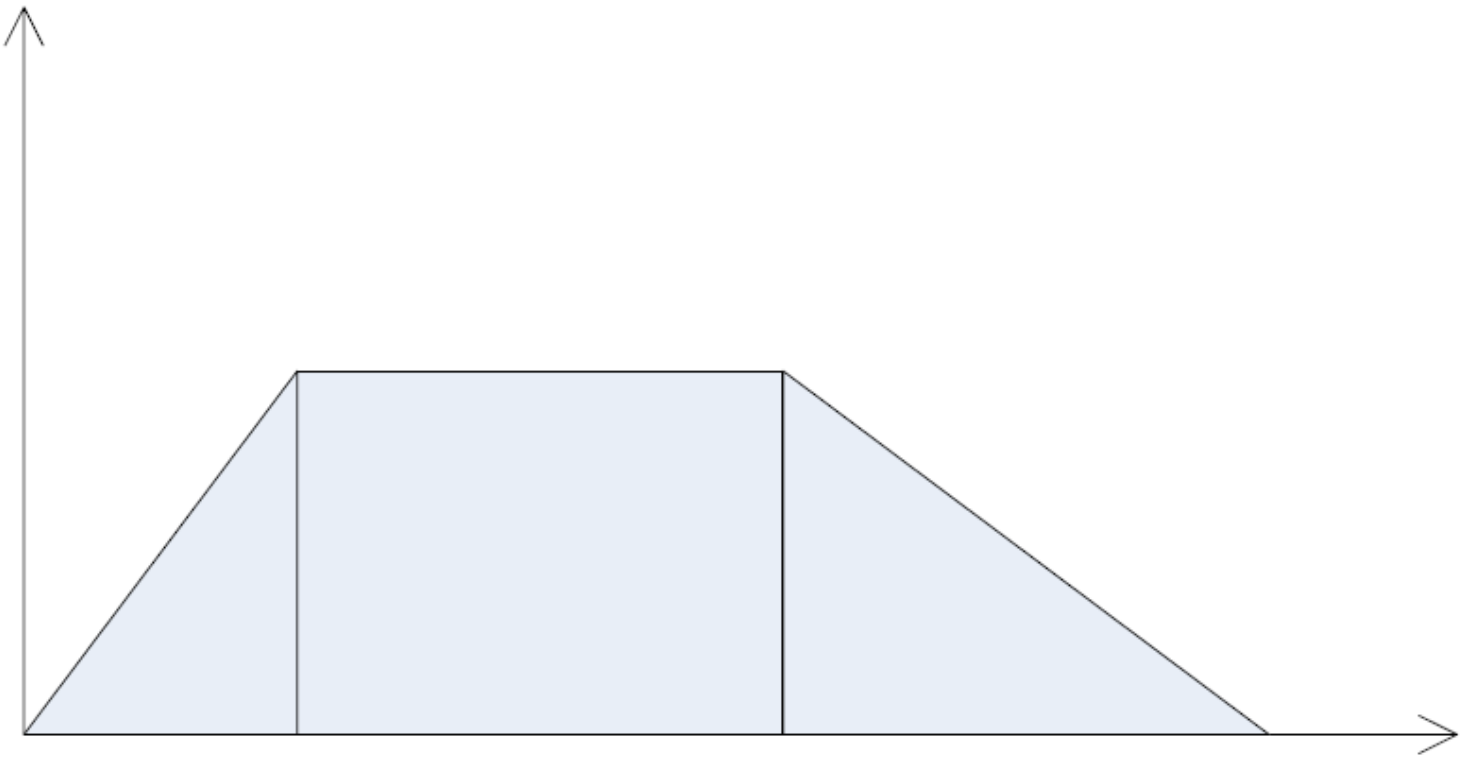
- La **prevención**: Basada en que las transacciones puedan renunciar a un bloqueo que hayan realizado previamente. Para ello, se detectan los casos en los que una transacción T1 tenga reservado un ítem y una transacción T2 intenta hacer un tipo de bloqueo sobre ese ítem que es incompatible con el bloqueo de T1, aplicando técnicas de prevención, de entre las que se pueden destacar:
  - Wait-Die: Si T2 es más antigua, se le hace esperar. En caso contrario, T2 se cancela.
  - Kill-Wait: Si T2 es más antigua, se cancela. En caso contrario, se hace esperar a T2.
- La **eliminación**: Es el modo más sencillo de los tres. Consiste en eliminar una de las transacciones bloqueadas, continuando la otra transacción su camino. El gestor de transacciones volvería entonces a generar la transacción eliminada y la reintroduciría en la cola de transacciones, siendo lo más probable que la otra transacción haya terminado o haya avanzado hasta un punto en el que no se repita la situación de interbloqueo.

EL PROTOCOLO DE BLOQUEO EN DOS FASES

Es una implementación muy extendida de uno de los principios de la resolución de conflictos basada en bloqueos (técnica pesimista). El protocolo de bloque en dos fases define un conjunto de reglas sobre la aplicación de bloqueos que pueden ser utilizadas por el gestor de transacciones para gestionar las transacciones concurrentes.

Estas reglas son:

1. En una transacción sólo se va a escribir sobre un objeto una vez.
2. Si una transacción necesita leer y escribir, realizará en primer lugar el bloqueo de escritura. De esta manera, basta con realizar una única operación de desbloqueo en la transacción (un desbloqueo genérico que termina con el bloqueo de lectura y escritura).
3. Antes de realizarse las acciones de lectura o escritura, deben hacerse los bloqueos correspondientes, los cuales deben permanecer hasta que las acciones hayan finalizado.
4. Los bloqueos del mismo tipo sólo se realizan una vez por transacción.
5. En una transacción, una vez que se hace el primer desbloqueo, no se vuelve a hacer ningún bloqueo sobre ningún objeto. Esta norma caracteriza al bloqueo en dos fases, en la que se distingue una fase de crecimiento, donde van apareciendo nuevos bloqueos, y una fase de decrecimiento, que se inicia a partir del primer desbloqueo y dura hasta el final de la transacción, y en cuyo transcurso el número de bloqueos siempre decrece. Gráficamente, el comportamiento de este protocolo puede representarse así (evolución del número de bloqueos de una transacción con el algoritmo 2PL):



Entre las ventajas del protocolo de bloqueo en dos fases o 2PL, podemos citar do:

- Es un protocolo seguro, es decir, nunca se van a producir las situaciones anómalas de concurrencia estudiadas en el punto anterior.
- Es un protocolo sencillo, fácil de implantar.

Entre sus desventajas, podemos citar:

- Los bloqueos tienden a ser grandes, lo que va a provocar incompatibilidades habituales con otras transacciones, derivándose de ello una pérdida de rendimiento.
- Este protocolo permite la aparición de interbloqueo.

Time-Stamping

El time-stamping trata de conseguir la serializabilidad de los planes de ejecución haciendo que, cuando varias transacciones tengan que acceder a objetos comunes, lo hagan en distintos momentos. Este objetivo se consigue asignando una marca temporal distinta (time-stamp) a cada transacción.

Cuando una transacción intenta acceder a un objeto para leerlo o modificarlo, se comprueba previamente si ya ha sido accedido por otra transacción más joven. Existen tres modalidades de time-stamping:

TIME-STAMPING BÁSICO

1. Se almacena en cada objeto el time-stamp de la última transacción que lo ha leído (TSR) y el de la última que lo ha grabado (TSW).
2. Cuando una transacción T intenta leer un objeto, si su time-stamp es mayor o igual que el TSR del objeto, podrá leerlo, debiendo ser cancelada en caso contrario.
3. Si una transacción T intenta escribir un objeto, se pueden producir las siguientes situaciones:

```
TS (T) >= TSW(Objeto):
     Si TS(T) >= TSR(Objeto), podrá escribir.
                              Si no, T tendrá que se cancelada
Si TS(T) <= TSW(Objeto):
```

```
Si TS(T) >= TSR(Objeto), T puede seguir adelante saltándose la
                         grabación, ya que la versión que habría
                         grabado no sería la última y no habrá
                         sido leída por ninguna transacción
                         (regla de escritura de Thomas).
                         En caso contrario, T deberá ser cancelada.
```

El time-stamping básico no asegura que se eviten las anomalías de concurrencia, pero si que impide los efectos de las anomalías de la Actualización Perdida y los de la Lectura Sucia (ya que si la transacción no puede hacer un commit, tampoco podrán hacerlo las que hayan leído o escrito los objetos que ella ha actualizado).

TIME-STAMPING DINÁMICO

Consiste en asignar un time-stamp a las transacciones cuando tratan de acceder a un objeto que ha sido leído o actualizado por otra transacción que no ha terminado todavía. De este modo, se evitan algunas de las cancelaciones de transacciones que se producen en el time-stamp básico. Su funcionamiento es el siguiente:

Cuanto T1 intenta leer o actualizar un objeto actualizado por T2 o intenta actualizar un objeto que T2 ya ha leído:

1. Si T1 y T2 no tienen todavía time-stamp, se les asigna a cada una un time-stamp, cumpliéndose que TS(T1) > TS(T2).
2. Si una de las dos no tiene time-stamp, se le asigna, respetándose de nuevo que TS(T1) > TS(T2).
3. Si T1 y T2 tiene ya time-stamp y TS(T1) > TS(T2), entonces continúa normalmente la ejecución de las dos transacciones. En caso contrario, se cancelará la transacción T1.

Al igual que el time-stamping básico, el time-stamping dinámico no asegura que se eviten las anomalías de concurrencia, pero si que impide los efectos de las anomalías de la Actualización Perdida y la Lectura Sucia. Además, es más eficiente que el time-stamping estático, ya que el número de cancelaciones que produce es menor.

TIME-STAMPING MULTIVERSIÓN

Basada en almacenar versiones de los ítems de la BD, permite el acceso simultáneo a un ítem por parte de transacciones diferentes, de forma que los accesos de cada transacción a los ítems que necesita se hace siempre usando versiones consistentes de esos ítems.

Métodos optimistas

Basados en la suposición de que los conflictos entre transacciones se producen con escasa frecuencia. Por tanto, se acepta como válido cualquier plan de ejecución de transacciones, y es tras finalizar la ejecución del plan y antes de realizar el commit (los datos todavía no son persistentes en la BD) cuando se comprueba si se ha producido algún conflicto entre transacciones o se ha incumplido alguna condición de consistencia. Si estos problemas han aparecido, las transacciones afectadas se deshacen (abort) y vuelven a situarse en la entrada del gestor de transacciones para su ejecución.

En los métodos optimistas se puede considerar la ejecución de una transacción dividida en tres fases:

- **La fase de lectura**: En ella se produce la ejecución de las acciones de la transacción. Todos los objetos de la BD que se necesitan son leídos y escritos de un buffer de memoria local al programa, y por tanto las acciones no afectan al resto de las transacciones en ejecución.

- **La fase de validación**: Se comprueba si las modificaciones introducidas por la transacción pueden ser hechas persistentes en la BD sin incumplir la condición de serializabilidad de la transacción.
- **La fase de escritura**: Si la comprobación realizada durante la fase de validación es positiva, las acciones de la transacción se ejecutan sobre la BD. Si la comprobación es negativa, la transacción es abortada.

TÉCNICA OPTIMISTA BÁSICA

La técnica optimista básica determina el conjunto de transacciones que se han validado utilizando una técnica de time-stamping. Durante la fase de validación se analiza si hay algún ítem común y del mismo nivel de granularidad entre el conjunto de ítems leídos por una transacción T1 y el conjunto de ítems que han escrito las transacciones cuyo time-stamp se ha asignado entre el instante de inicio y el instante de finalización de la fase de lectura de la transacción. Si no existe esa coincidencia, se añade un time-stamp a la transacción T1.

Esta técnica exige que haya como mucho una transacción en la fase de validación en cada instante de tiempo, por lo que la fase de validación actúa como cuello de botella del sistema de gestión de transacciones.

Métodos de control de concurrencia basados en la semántica

Son métodos que, además de la sintaxis, utilizan la semántica de las acciones a realizar para determinar si un plan de ejecución es o no serializable. Es decir, no se limitan a comprobar si los accesos de las transacciones son de lectura o escritura, sino que analizan que van a escribir concretamente las acciones de las transacciones y determinan si esas acciones pueden generar un conflicto o no.

Bibliografía

- Scribd (Ibiza Ales)

Tipos abstractos de datos y estructuras de datos. Grafos. Tipos de algoritmos: ordenación y búsqueda. Estrategias de diseño de algoritmos. Organizaciones de ficheros.

Tablas, listas y árboles

Tipos abstractos de datos

Al escribir un programa para resolver un problema, el enfoque tradicional consistía en pasar de la realidad a una implantación en el lenguaje de programación utilizado, lo que conducía, inevitablemente, a que la forma de “ver” los datos estuviera muy influida por la máquina donde se ejecutaba el programa. Con el tiempo y la experiencia acumulada surgió otro enfoque mejor para afrontar esta situación, que consistía en establecer un nivel intermedio, donde se modela lo esencial de la realidad, mediante técnicas de abstracción, sin comprometerse con detalles de implantación. Estas técnicas de abstracción han evolucionado en el sentido de alejar los elementos que aparecen en los sistemas de software de las nociones propias de las máquinas sobre las que se implantan

dichos sistemas, aproximándolos a las nociones propias de los dominios en que se presentan las situaciones modeladas.

La principal de estas técnicas está basada en los **Tipos Abstractos de Datos (TAD)**.

Esta técnica está centrada en las abstracciones de datos, consiste en:

- Identificar los distintos tipos de datos que intervienen en el sistema y la función principal de dicho sistema.
- Caracterizar cada tipo de datos en función de las operaciones que se puedan realizar con los objetos de los distintos tipos (haciendo abstracción de sus representaciones concreta).
- Componer el sistema utilizando objetos de los tipos definidos junto con sus operaciones características.
- Implantar cada uno de los tipos utilizados.

Estas características son la base conceptual de los TAD. De forma sintetizada, puede decirse que:

Un TAD es una estructura algebraica, a saber, un conjunto de objetos estructurados de alguna forma y con ciertas operaciones definidas sobre ellos.

Piénsese, por ejemplo, en una calculadora; los elementos que maneja son cantidades numéricas y las operaciones definidas son operaciones aritméticas. Otro ejemplo posible es el TAD matriz matemática; los elementos que maneja son las matrices y las operaciones son las que nos permiten crearlas, sumarlas, invertirlas, etc. Otro tipo de TAD es cualquier tipo de cola de espera: en el autobús, cine, la compra, etc. Los elementos de las colas son las personas y las operaciones más usuales son entrar o salir de la cola, pero no son las únicas. Desde un punto de vista abstracto podemos pensar en operaciones como crear un elemento de la cola, comprobar si la cola está vacía, si está llena, etc.

Es conveniente observar que las operaciones de un TAD son de diferentes clases. Algunos nos deben permitir crear objetos nuevos, otras determinar su estado, construir otros objetos a partir de algunos ya existentes, etc.

Puesto que los TAD deben ser lo más independientes posibles de los detalles de implantación, es obvio que no deben tener ninguna dependencia con respecto al lenguaje de programación elegido para implantarlos. Aun así, ciertos lenguajes no incorporan ciertas estructuras de datos necesarias para implantar algunos tipos de TAD.

El concepto de algoritmo

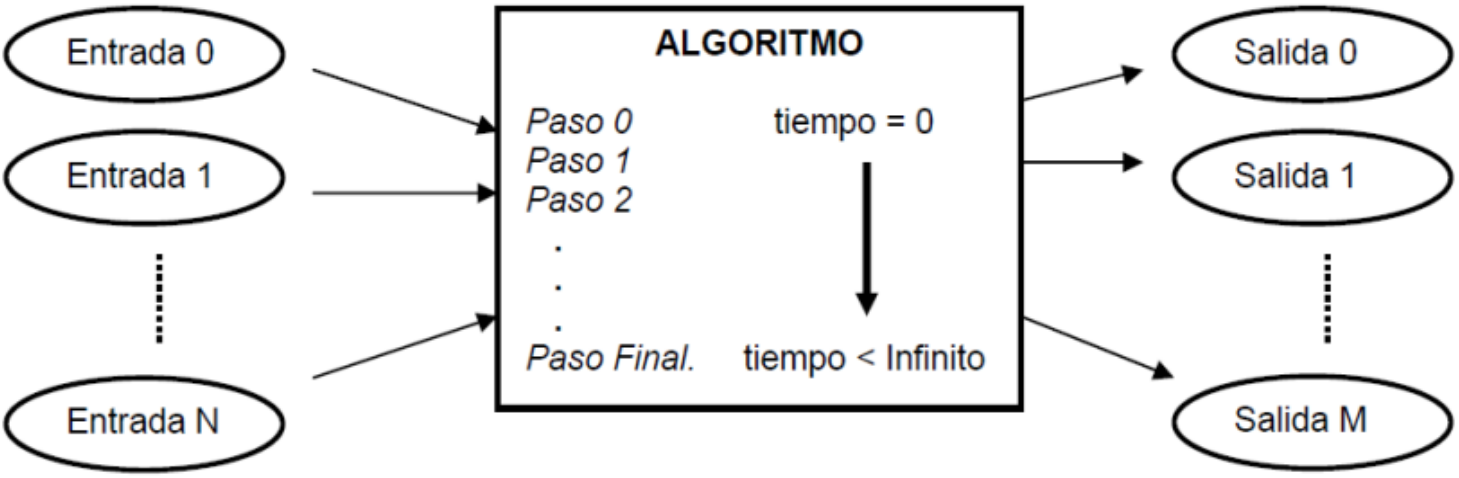
Un **algoritmo** es *un conjunto de pasos o instrucciones que se deben seguir para realizar una determinada tarea*.

Estas instrucciones deben cumplir las siguientes características:

- **FINITUD**: Debe ser *un conjunto finito de instrucciones* y que se realicen en un *tiempo finito*.
- **PRECISIÓN**: Debe indicar el *orden de realización de cada instrucción o paso de forma inequívoca*.
- **DEFINICIÓN**: Debe tener un *número finito (0 … N) de datos de entrada* y un *número finito (0 … M) de datos de salida (resultados)*. Frente a un mismo conjunto de datos de partida se debe llegar siempre a un mismo conjunto de resultados.

Otra cualidad deseable en un buen algoritmo, aunque no imprescindible para ser considerado como tal, es que sea *óptimo*, es decir, que sea la forma más fácil y rápida de hacer una determinada tarea, si bien es cierto que en muchas ocasiones facilidad y rapidez son dos cualidades contrapuestas.

De forma gráfica, la siguiente figura engloba las principales características de un algoritmo.



Coste de un algoritmo

Normalmente, si se escribe un programa para resolver un problema, se hace para que éste sea utilizado muchas veces, por lo que resulta conveniente caracterizarlos según su tiempo de ejecución y la calidad de la respuesta. Cuando estudiamos algoritmos es muy importante caracterizar la solución obtenida de cada algoritmo antes de estar seguros de que dos algoritmos son equivalentes (para un mismo valor de entrada dan exactamente el mismo resultado) o son similares (pueden dar resultados diferentes, pero desde el punto de vista del problema que estamos resolviendo somos indiferentes a cualquiera de los resultados).

Por esta razón, uno de los criterios más importantes para seleccionar un algoritmo es evaluar el tiempo que tarda en ejecutarse, que llamaremos **coste del algoritmo** o **tiempo de ejecución**. Para analizar el coste de un algoritmo adoptamos un modelo que nos dice que recursos usados por la implantación del algoritmo son importantes para su desempeño. La complejidad de un algoritmo bajo un modelo computacional es la cantidad de recursos, en tiempo o espacio, que el algoritmo usa para resolver el problema.

Desafortunadamente, por lo general es imposible predecir el comportamiento exacto de un algoritmo, ya que existe la influencia de muchos factores, de aquí que se trate de extraer las características principales de un algoritmo. Así, se definen ciertos parámetros y ciertas medidas que son las más importantes en el análisis y se ignoran detalles relativos a la implantación.

Así, el análisis es una aproximación, no es perfecto, pero lo importante es que se pueden comparar diferentes algoritmos para determinar cuál es mejor para un propósito determinado.

La metodología que generalmente se utiliza para predecir el tiempo de ejecución de algoritmos se basa en el **comportamiento asintótico**, en este método se ignoran los factores constantes y se concentra en el comportamiento del algoritmo cuando el tamaño de la entrada tiende a infinito. Es totalmente análogo al estudio de los límites de una función cuando su variable independiente tienen a infinito, por ejemplo:

$F(x) = 6x^2 + x + 3$. Haciendo $x \to \infty$ (infinito) esta función es equivalente a: $F(x) = 6x^2$

De hecho, es muy usual evaluar el coste de un algoritmo como una función matemática del número de entradas del algoritmo (N), siendo un algoritmo de mayor coste que otro cuando para un mismo N la función que expresa el coste da un resultado superior. Así

pues, los costes o tiempos de ejecución más usuales de los algoritmos, de menor a mayor, son:

1	Constante
Log N	Logarítmico
N	Lineal
N * (Log N)	Semi-Logarítmico
N^2	Cuadrático
N^3	Cúbico
N!	Factorial
2^n	Exponencial

Por otra parte, conviene considerar dos casos límite, el **mejor caso** y el **peor caso**. Puede ocurrir que al realizar una operación se parta de unos datos iniciales del TAD que hagan que al aplicar el algoritmo éste se ejecute de la manera más rápida posible, y puede ocurrir también lo contrario, encontrarnos con el caso en que se ejecute de la forma más lenta. Por ejemplo, si queremos ordenar unos datos, parece lógico pensar que el mejor caso será cuando los datos ya estén ordenados, y el peor caso cuando ningún dato inicial esté en el orden que tendrá cuando se haya ejecutado la ordenación.

De forma estadística se supone que se parte de un caso medio. Por otra parte, hay algoritmos cuya diferencia de coste entre los casos mejor y peor es muy grande, y otros en los que, debido a la naturaleza del algoritmo, la diferencia es muy pequeña o incluso inexistente.

Si un algoritmo crece de manera proporcional a N, se dice que es de orden N. En general, el tiempo de ejecución es proporcional, esto es, multiplica por una constante a alguno de los costos anteriormente propuestos, además de la suma de algunos términos más pequeños.

Implantación tradicional frente a los TAD

Según Nicklaus Wirth, un programa responde a la ecuación:

```
(Ec. 1)   Programa = Datos + Algoritmos
```

El enfoque tradicional se ciñe bastante bien a esta concepción. En la ecuación de Wirth la parte *Algoritmo* la podemos expresar como:

```
(Ec. 2)   Algoritmo = Algoritmo de datos + Algoritmo de control
```

Se entiende como *Algoritmo de datos* a la parte del algoritmo encargada de manipular las estructuras de datos del problema, y *Algoritmo de control* a la parte restante (la que representa en sí el método de solución del problema, también llamada *lógica de negocio*, independiente hasta cierto punto de las estructuras de datos seleccionadas).

Con los TAD se identifican ciertas operaciones o partes del algoritmo que manipulan los datos. Además de proporcionar una estructura de datos, por lo que podemos sustituir el sumando "*Datos*" de la ecuación anterior por el sumando "*Estructura de Datos*". De esta forma, podemos entonces escribir:

```
(Ec. 3)   Programa = Estructura de Datos + Algoritmo de Datos
                     + Algoritmos de Control
```

Definiendo:

`(Ec. 4)  Implantación del TAD = Estructura de Datos + Algoritmos de Datos`

Se establece la *ecuación fundamental* que describe el enfoque de desarrollo con Tipos Abstractos de Datos:

`(Ec. 5)  Programa = Implantación del TAD + Algoritmo de Control`

A la hora de crear un Tipo Abstracto de Datos, la ecuación a seguir es (Ec. 4), es decir, determinar como es la **estructura de datos** y cuales son los algoritmos de control para manera los datos, en otras palabras, las **operaciones sobre los datos** que se pueden hacer, o interesa hacer, sobre la mencionada estructura de datos.

Operaciones de los TAD

Las operaciones en los TAD pueden servir para crear nuevos objetos abstractos, para obtener información acerca de ellos, y algunas para construir nuevos elementos a partir de otros ya existentes. De esta forma las operaciones las podemos clasificar de esta manera:

- Operaciones de CREACIÓN: Se utilizan para definir y/o crear el TAD y sus elementos.
- Operaciones de TRANSFORMACIÓN: Se utilizan para realizar algún tipo de transformación en los componentes del TAD. Por ejemplo: asignación de valor, ordenaciones, permutaciones, balanceados, etc.
- Operaciones de ANÁLISIS: Se utilizan para obtener información concerniente a cualquiera de los componentes del TAD o de su estructura. Por ejemplo: búsquedas, recorridos, obtención de algún dato del TAD (valor de un componente, cantidad de elementos, profundidad, número de niveles, etc.)

Implantación y tipos de TAD

Cuando ya se tiene bien diseñado un Tipo Abstracto de Dato, el siguiente paso es decidir una implantación. Esto supone elegir algunas de las estructuras de datos que nos proporcione el lenguaje de programación utilizado para representar cada uno de los objetos abstractos y, por otro lado, escribir una rutina (Procedimiento o función) en tal lenguaje que simule el funcionamiento de cada una de las operaciones especificadas para el TAD.

La selección de las estructuras de datos determina la complejidad del algoritmo que implanta una operación y su elección es, por esta razón, de gran importancia. Existen estructuras de datos muy dependientes de un lenguaje de programación y debido a esto deben tratar de evitarse cuando el TAD se quiere que sea portable.

Existen muchos tipos de TAD, pero los más utilizados son:

- Tablas
- Listas
- Árboles

Tablas

Una **tabla** o *matriz*, es un TAD que representa una estructura homogénea de datos donde se cumple:

- Todos sus c*omponentes son del mismo tipo*.
- Tiene un *número predefinido* de componentes que no puede variarse en tiempo de ejecución.

- Los elementos de la tabla contienen *una clave* que los identifica de forma unívoca. El conjunto de claves forman un conjunto de *índices* para localizar los elementos.
- Se permite el *acceso directo* a cualquiera de los elementos de la tabla a través de los índices.

La operación fundamental en el uso de una tabla es ***localizar la posición*** de sus elementos con una clave conocida "a priori". Dicho de otra forma, lo más usual es que se quiera conocer el contenido del i-esimo (índice i) elemento de una tabla. Es menos frecuente la operación inversa, dado un valor saber los índices de los elementos cuyo contenido coincide con el valor dado; aunque también resulta útil en ocasiones esta operación.

Tipos

Recordando la (Ec. 4) de Wirth, desde el punto de vista de la *Estructura de Datos* de una tabla, la principal característica de su estructura es la *dimensión*.

Se habla entonces de:

- Tabla **Monodimensional**:
  - Se refiere a tablas de una dimensión con un determinado número (N) de elementos. La declaración de estas tablas responde a la sintaxis genérica:
    - `Nombre_Tabla = matriz[1..N] de Tipo_Elemento;`
- Tabla **Multidimensional**:
  - Se refiere a tablas de más de una dimensión (d), con un determinado número de elementos para cada dimensión (N1, N2, ..., Nd). Su declaración es:
    - `Nombre_Tabla = matriz[1..N1, ... 2..Nd] de Tipo_Elemento;`

Aunque cada dimensión puede tener diferente número de elementos, todas las dimensiones tienen el mismo tipo de elemento, no pudiéndose declarar diferentes tipos de elementos según las diferentes dimensiones de la matriz, pues supondría violar la primera característica de las tablas.

Operaciones

Desde el punto de las *operaciones* (Algoritmo de Datos), las operaciones básicas en una tabla son:

- **Definir/Crear** la tabla: Se refiere a la forma que cada lenguaje debe tener para definir la estructura de una tabla y crear una variable del tipo de la tabla definida.
- **Insertar/Eliminar** elementos de la tabla: Aunque cuando se define una tabla se indica "a priori" el número de elementos que tiene, eso no quiere decir que desde el principio esos elementos tengan un valor significativo para el uso que se les piensa dar. Es muy común al crear una tabla asignar a todos sus elementos un valor especial (nulo) que indique que en caso de que el elemento tenga ese valor, a todos los efectos, desde un punto de vista abstracto, es como si ese elemento no existiese.
- **Buscar** elementos de la tabla: Esta operación es fundamental en el uso de tablas. Existen distintas formas de hacer búsquedas en una tabla, y según los valores de los componentes y el tipo de búsqueda ésta será más o menos eficiente (rápida).
- **Ordenar** los elementos de la tabla: Esta operación resulta muy útil a la hora de realizar búsqueda. Veremos más adelante que resulta mucho más efectivo realizar búsquedas sobre tablas donde se ha establecido algún tipo de orden sobre otras donde no lo hay.
- **Contar** los elementos de la tabla: Calcular el número de elementos que hay en la tabla en un momento dado. La acción de preguntar si una tabla está llena o vacía (de elementos significativos) son casos particulares de la operación de contar. También

pueden considerarse recuentos más especializados, como contar el número de elementos con un determinado valor, o con valor superior a uno dado, etc.

Hay varios algoritmos que implantan las operaciones de búsqueda y ordenación en una tabla (las más complejas y utilizadas). Las operaciones de definición, creación y recuento resultan muy sencillas de implantar.

Representación

La forma más simple de representación abstracta de una tabla es:

$$\text{Nombre de Tabla} = \{ e_0, e_1, ..., e_i, ..., e_{N-1} \}$$

Para índices que van desde 0 a (N-1) y siendo ei el contenido del elemento i de la tabla.

Es muy común la siguiente representación, más intuitiva, de una tabla:

TABLA MONODIMENSIONAL DE N ELEMENTOS					
ÍNDICE del elemento	*0*	*1*	*2*	....	*N – 1*
VALOR del elemento	Juan	Pedro	Inés	....	Ana

En la figura anterior se representa una tabla de una dimensión con N elementos, cuyos índices van desde 0 hasta (N-1), y cuyo contenido se refiere a nombres de persona. Para representar tablas multidimensionales, basta con utilizar una representación como la indicada para cada dimensión de la tabla.

Implantación

Puesto que una de las características de las tablas es la invariabilidad del número de componentes, prácticamente todos los lenguajes de programación implantan las tablas usando memoria estática (en oposición a la memoria dinámica). Una vez que el programa ha sido preparado para ser ejecutado (compilado y enlazado), la posición y el espacio de memoria que utilizará la tabla en la ejecución del programa ya está fijado y no puede de ninguna forma cambiarse o utilizarse para otros fines que no sea almacenar los elementos de la tabla.

Hay otros tipos de TAD cuya implantación se realiza frecuentemente con memoria dinámica, cuyo significado fundamental es que, en oposición a la memoria estática, el tamaño y la posición del espacio de memoria utilizado si puede cambiar en tiempo de ejecución, pudiéndose utilizar para otros fines.

Listas

Una **lista** es una estructura de datos que cumple:

- Todos sus *componente son del mismo tipo*.
- Cada *elemento* o *nodo va seguido de otro* del mismo tipo o de ninguno.
- Sus componentes se almacenan según *cierto orden*.

Nótese las diferencias con respecto a las tablas:

- Los elementos de las tablas no se almacenan según un orden. En las listas siempre hay un orden.
- No existe unos conjuntos de índices asociados a cada posición, tal como ocurre con las tablas. Por tanto, en las listas será precio ir recorriendo los elementos hasta encontrar el buscado.

Tipos

Una manera de clasificar las listas es por la forma de acceder al siguiente elemento:

- **Lista densa**: La propia estructura determina cuál es el siguiente elemento de la lista. Por ejemplo, aplicando ciertos "trucos" se puede usar una tabla para implementar una lista, de forma que el siguiente elemento de la lista fuera dado por el orden del índice (no confundir con el orden de los elementos del array). Nótese sin embargo que, puesto que un array es estático, la memoria reservada para implementar esta "lista" también será estática.
- **Lista enlazada**: Cada elemento contiene la información necesaria para llegar al siguiente. La posición del siguiente elemento de la estructura la determina el elemento actual. Es necesario almacenar al menos la posición de memoria del primer elemento. Además es dinámica, es decir, su tamaño cambia durante la ejecución del programa.
- Dentro de las listas enlazadas podemos distinguir, según la cantidad de enlaces por nodo, entre:
  - **Lista simplemente enlazada**: Cada elemento conoce qué elemento es el que le sucede en el orden, pero no cual le precede.
  - **Lista doblemente enlazada**: Cada elemento conoce qué elemento le precede y cual le sucede en el orden.
  - **Lista con enlaces múltiples**: Son aquellas listas en donde cada elemento, aparte de conocer los elementos que le preceden o suceden, también tiene uno o varios enlaces al primer elemento de otra lista, siendo éstas sublistas de la anterior.

También podemos clasificar las listas en función de que los elementos se coloquen en la lista por orden de llegada y se acceda a ellos por su posición. Los ejemplos más usuales de este tipo de listas son:

- **PILAS** (Estructuras LIFO. Last In First Out): El último elemento en entrar es el primero en salir.
- **COLAS** (Estructuras FIFO. First In First Out): El primer elemento en entrar es el primero en salir.

Operaciones

Las operaciones básicas a realizar en una lista son las mismas que en las tablas, si bien, la forma en que se realizan en las listas difiere notablemente entre ambos TAD, sobre todo en lo concerniente a las operaciones de Inserción, Eliminación, Búsqueda y Ordenación.

A la hora de insertar o eliminar elementos de una lista, puede distinguirse entre insertar o eliminar el primer elemento de la lista, un elemento intermedio o el último elemento. En todos los casos resulta de capital importancia reordenar los diferentes enlaces entre los elementos de forma correcta, pues realizar mal un enlace o perderlo supone la imposibilidad definitiva de acceder a parte de la lista, o incluso a toda ella.

Representación

La representación más habitual de una lista se hace colocando sus elementos entre paréntesis, separados por comas, según muestra el siguiente ejemplo:

$$\text{Nombre de Lista} = (e_1, e_2, \ldots, e_n)$$

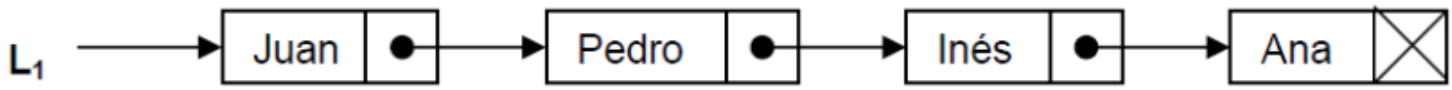
Siendo e1, e2, …, en los elementos de la lista.

Puesto que los componentes de una lista se almacenan según cierto orden, es muy importante notar que:

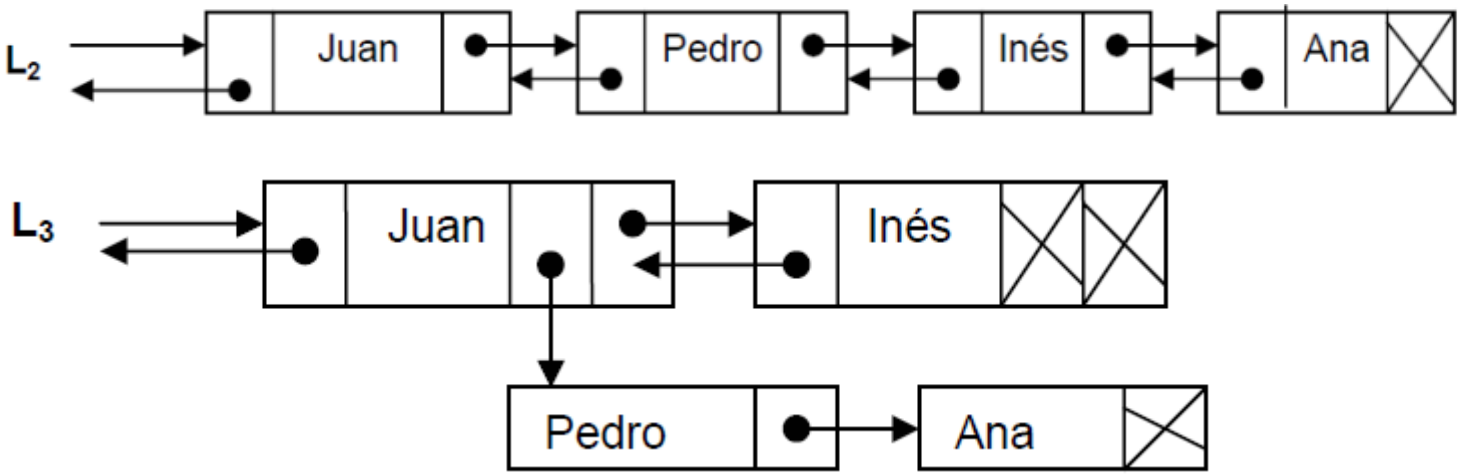
`L1 = (e1, e2, ..., en) es una lista distinta de L2 = (en, e1, ..., e2)`

De forma gráfica presentamos las siguientes listas:

Simplemente enlazada:



Doblemente enlazada:



En estas representaciones las flechas indican los enlaces entre los componentes de las listas. El punto negro de la flecha indica el elemento origen y la punta de flecha el elemento destino, de tal forma que el origen conoce el destino, pero el destino no conoce su elemento origen. Por esta razón es por lo que las listas doblemente enlazadas precisan “dos” enlaces, uno en cada sentido. En la lista **L1** el elemento de contenido “Juan” sabe que le sigue el elemento “Pedro”, y éste sabe que le sigue “Inés”, pero “Pedro” no sabe que le precede “Juan”.

El símbolo **X** o aspa que aparece en algunos elementos indican que no tienen enlace a ningún sitio, se entiende que es un enlace nulo. Obsérvese que todas las listas tienen que tener un “punto de entrada”, es decir, una referencia al primer elemento de la lista. Estas referencias, en nuestro ejemplo, son **L1, L2** y **L3**.

También conviene fijarse de forma especial en las listas con enlaces múltiples. Mientras que en las listas simples y dobles únicamente hay una forma de “recorrer” la lista, desde el primer elemento hasta el último”, en las listas con múltiples enlaces surgen varias formas de recorrerlas, según nos decidamos a avanzar por uno de los múltiples enlaces a otras *sublistas* que puede tener cada elemento.

Implantación

Las listas son un tipo de TAD cuya implantación puede realizarse de múltiples maneras. Según el tamaño, manejo o rendimiento que queramos que tengan ciertas operaciones, como búsquedas, ordenaciones, inserciones, etc, las listas pueden implementarse con arrays, ficheros secuenciales o punteros. Estructuras de datos todas ellas muy comunes en la mayoría de los lenguajes de programación.

También conviene observar que para implantar listas puede usarse tanto memoria estática (arrays) como memoria dinámica (punteros). Si bien es poco frecuente usar los arrays, pues en esencia son tablas. Para implementar una lista simple usando tablas hace falta recurrir a algún tipo de “truco”, por ejemplo, almacenar en el contenido de los elementos

del array dos datos, el contenido de los elementos de la lista y el “puntero” al siguiente elemento.

Veamos un ejemplo de lista implementada con array:

ARRAY (TABLA)				
ÍNDICE del array	***0***	***1***	***2***	***3***
VALOR del elemento	Juan#2	Inés#3	Pedro#1	Ana#(-1)

Nótese como junto al nombre de las personas, usamos el carácter especial (#) para separar los nombres de los valores numéricos que hacen de puntero, indicando qué índice de la tabla contiene el siguiente elemento de la lista. Para indicar el valor nulo usamos un valor de índice imposible, en este caso el negativo (-1). Hay que sobreentender que el puntero de comienzo de la lista es siempre el índice cero del array.

Si se eliminasen algunos elementos de la lsita, por ejemplo a Inés y a Pedro, supondría reorganizar los apuntadores, de forma que tendríamos: Juan#3, Inés#(-1); Pedro#(-1); Ana#(-1). Desde un punto de vista abstracto la lista sólo tiene ahora dos elementos Juan y Ana, sin embargo esto no quiere decir que el array haya disminuido su tamaño al dejar de apuntar a los elementos Inés y Pedro; simplemente estamos diciendo que hemos dejado de apuntarlos, pero el array sigue teniendo el mismo espacio reservado en memoria, y este espacio no está disponible para otro uso más que para almacenar los elementos del array. Esto es debido a que un array se implementa con memoria estática.

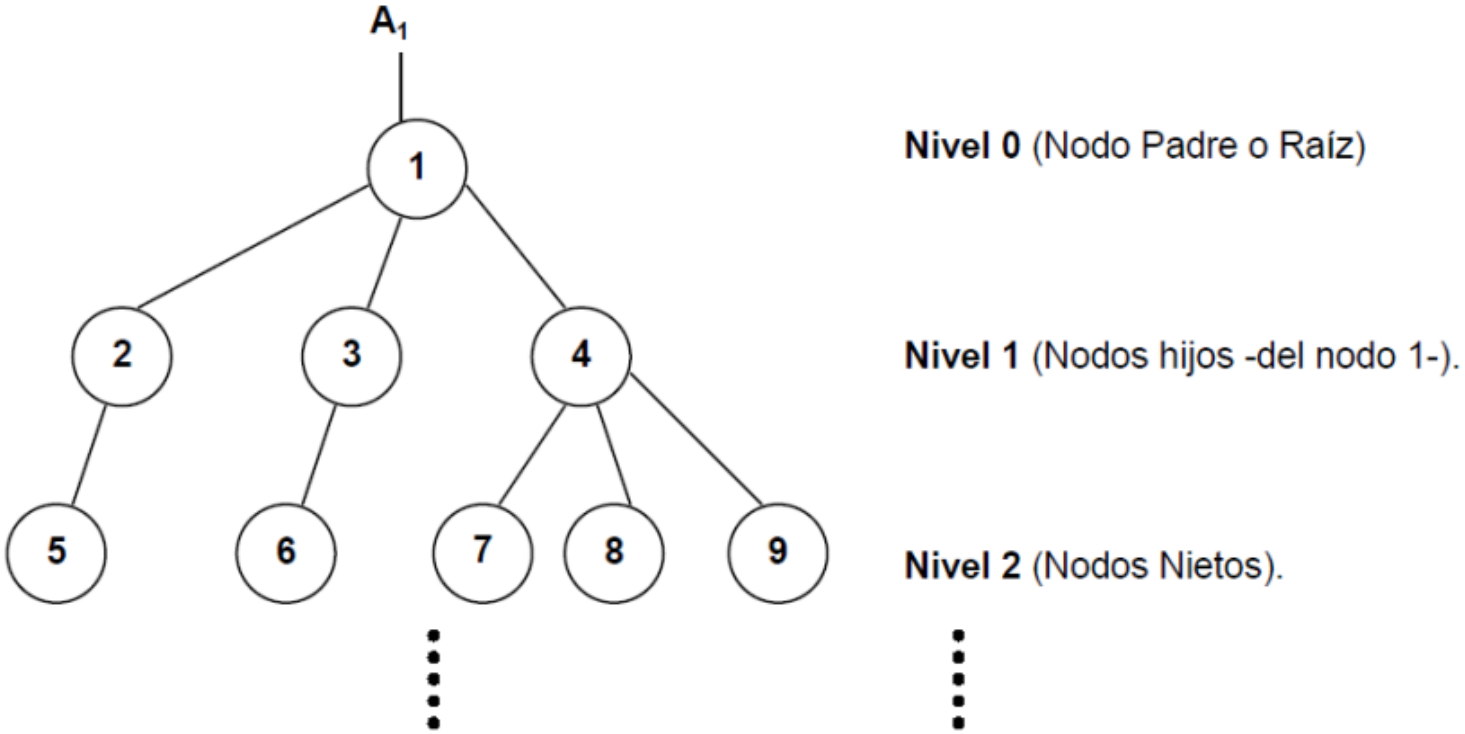
Usando memoria dinámica (punteros) no se tendría esta desventaja. Bajo esta circunstancia, si se elimina de la lista un elemento, el espacio de memoria que ocupaba queda disponible para ser usado para otras finalidades.

Árboles

Como dice David Harel en su libro “The Spirit of Computing”:

“Una de las estructuras de datos más importantes y prominentes que existen es el **árbol**. No es un árbol en el sentido botánico de la palabra, sino uno de naturaleza más abstracta. Todos hemos visto usar tales árboles para describir conexiones familiares. Los dos tipos más comunes de árboles familiares son el – árbol de antecesores – , que empieza en un individuo y va hacia atrás a través de padres, abuelos, etc, y el – árbol de descendientes – , que va hacia adelante a través de hijos, nietos, etc”.

Un árbol es un TAD cuya estructura corresponde con el siguiente gráfico:



En donde podemos ya notar las principales características de los árboles. Cada uno de las diferentes “circunferencias” representa un elemento o nodo del TAD tipo árbol. El nodo 1 suele ser llamada *nodo de nivel 0* (también nodo raíz por su similitud con un árbol real). Es importante percatarse de que sólo puede haber un nodo de nivel 0 en cada árbol.

Los nodos 2, 3 y 4 son llamados nodos de nivel 1, por estar todos ellos al mismo “nivel” al ser todos “hijos” del nodo 1. Este razonamiento se puede ir aplicando a los sucesivos niveles, hablándose de nodos nietos (o hijos si nos referimos al nivel inmediatamente superior).

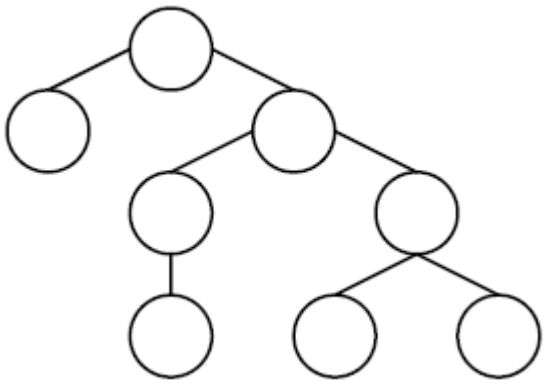
Por la similitud con un árbol en el sentido botánico, también se dice que los nodos que no tienen hijos son nodos hoja y los nodos que sí tienen hijos, salvo la raíz, son llamados nodos rama o nodos internos. Con el ejemplo del árbol anterior (**A1**), vamos a mostrar los principales conceptos usados al hablar de árboles:

- **Antecesor Directo** o **Padre**: El nodo 4 es el Antecesor Directo o Padre de los nodos 7, 8 y 9.
- **Antecesor** o **Ancestro**: El nodo 1 es el Antecesor de todos los otros nodos.
- **Sucesor Directo** o **Hijo**: El nodo 5 es el Sucesor Directo o Hijo del nodo 2.
- **Sucesor** o **Descendiente**: Todos los nodos son Sucesores o Descendientes del nodo 1.
- **Nodo de Nivel Cero** o **Raíz**: En nuestro ejemplo es el nodo 1.
- **Nodo Interno** o Rama: En nuestro ejemplo son los nodos 2, 3 y 4.
- **Nodo Hoja**: En nuestro ejemplo son los nodos 5, 6, 7, 8 y 9.
- **Nivel de un Nodo**: El nodo 1 tiene nivel 0. Los nodos 2, 3 y 4 nivel 1. Los nodos 5, 6, 7, 8 y 9 nivel 2.
- **Grado de un Nodo**: Es el número de descendientes directos que tiene un nodo. El grado del nodo 2 es 1.
- **Grado del Árbol**: Es el mayor de los grados de los nodos que los componen. En nuestro caso es 3.
- **Altura del Árbol**: Es el mayor de los niveles del árbol. En nuestro caso es 2.
- **Longitud de Camino de un Nodo**: Número de enlaces o arcos que hay que atravesar para ir de la raíz a un nodo. La longitud de camino del nodo raíz es la unidad. Por ejemplo, el nodo 5 tiene longitud de camino 3.

Tipos

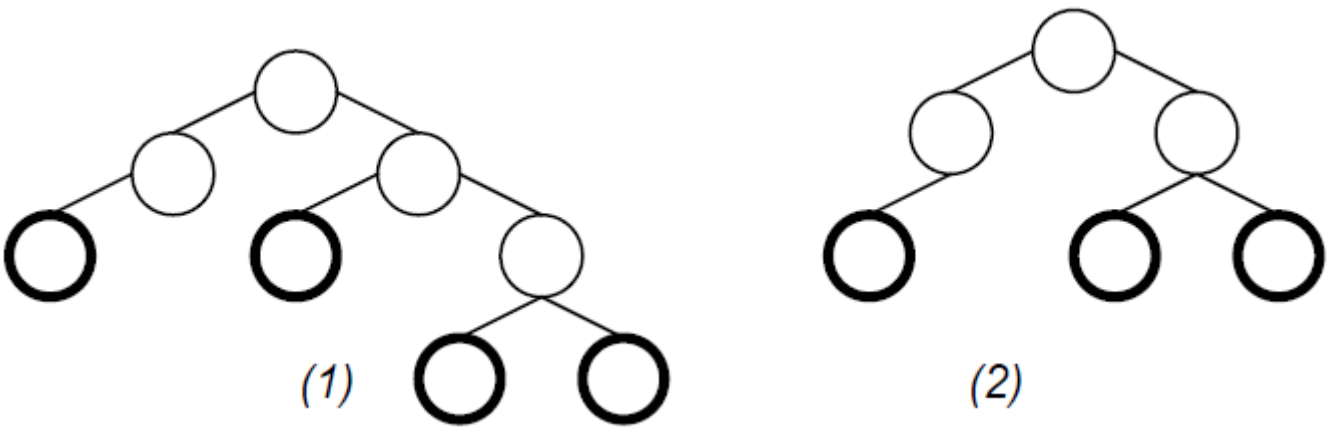
En función de la estructura que define un árbol, se pueden establecer distintos tipos de árboles atendiendo a ciertos aspectos en la *forma* de árbol. Según ésto es muy corriente hablar de los siguientes tipos de árboles:

- **Árbol BINARIO** o **Árbol B**: Es el árbol cuyo máximo número de nodos hijos que tiene cualquier nodo es dos.

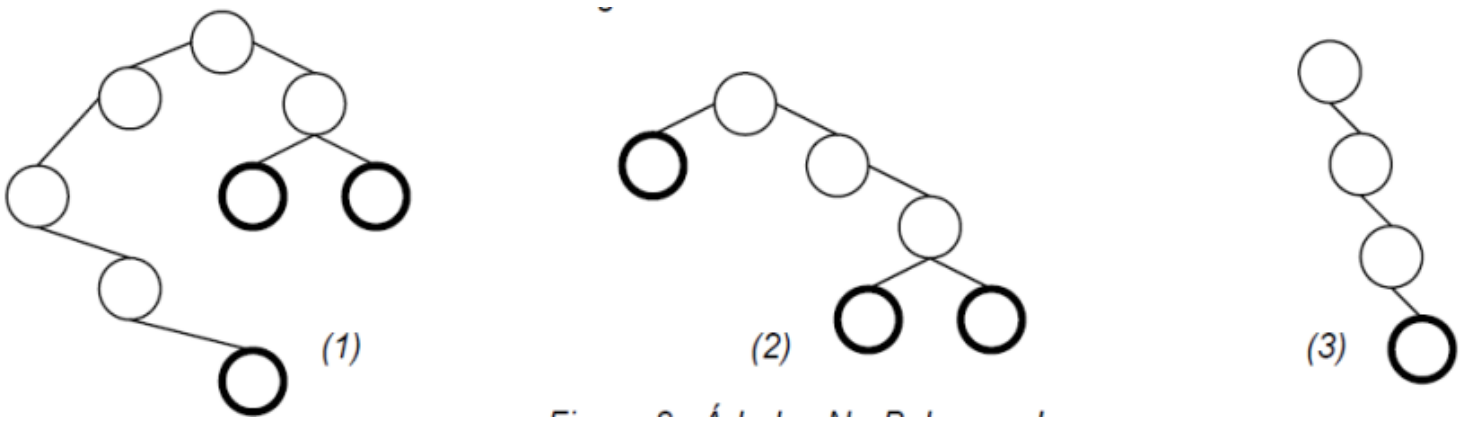


- **Árbol MULTIRAMA**: Son aquellos en los que no hoy límite para el número de nodos hijo. El número de niveles no crece tanto como en los binarios. Es el caso de nuestro árbol de ejemplo **A1**.
- **Árbol BALANCEADO**: Son aquellos árboles en los cuales se cumple que *entre todos sus nodos hoja no hay una diferencia de nivel superior a la unidad*. Veamos algunos ejemplos de árboles binarios balanceados (también denominados árboles AVL) y no-balanceados.

Balanceados:



No Balanceados:



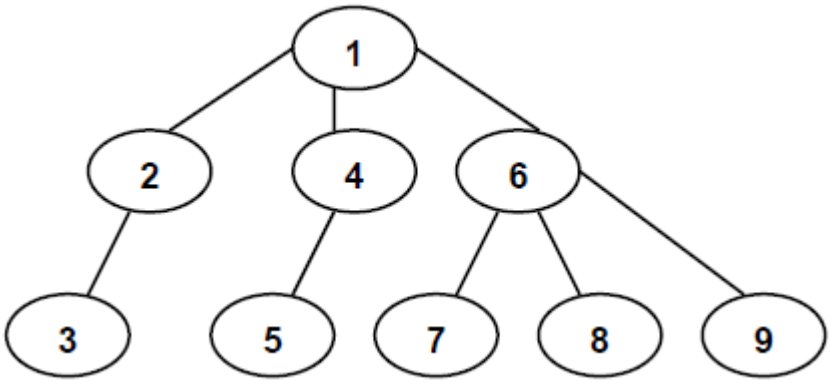
Es de observar que una lista realmente es un árbol en el que cada nodo únicamente tiene un hijo, y se considera al primer nodo de la lista el nodo raíz del árbol (tal como se muestra en el árbol (3) de la figura.

Una lista de más de dos elementos se corresponde a la estructura de un árbol con un grado de no-balanceo máximo (sólo hay un nodo hoja). Se suele decir en estos casos que la lista es un *árbol totalmente degenerado*.

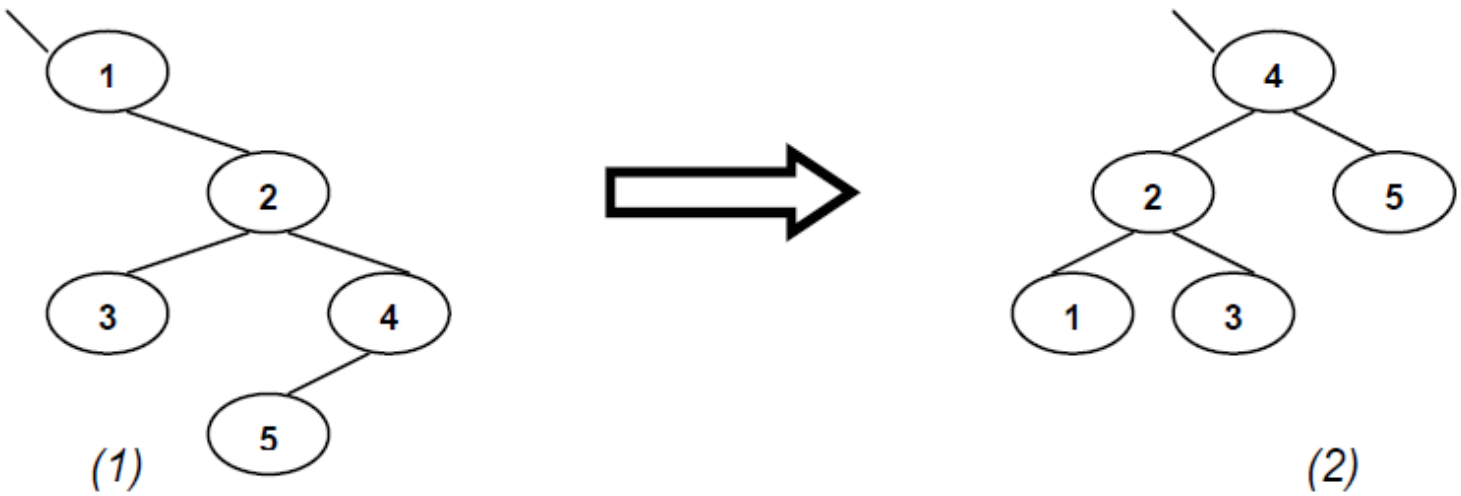
Operaciones

Las operaciones básicas que se pueden realizar sobre los árboles son:

- **Definir/Crear** el árbol: Se refiere a la forma que cada lenguaje debe tener para definir la estructura de tipo árbol y crear una variable de este tipo.
- **Recorrer** los diferentes caminos del árbol: Esta es una operación que resulta muy importante en los árboles, pues es sobre la que se apoyan las demás operaciones. Vamos a ver que existen dos formas de recorrer un árbol, en ambas, por convenio, se asume que siempre nos desplazaremos de *izquierda* a *derecha* en horizontal, al igual que la lectura de un libro. Respecto a la dimensión vertical, nos moveremos de arriba hacia abajo o viceversa. Existen dos forma de recorrer un árbol:
  - Recorrido en **AMPLITUD**: Se trata de recorrer consecutivamente los nodos que se encuentran en el mismo nivel (siempre de izquierda a derecha). La dimensión que prima en el recorrido es la horizontal. Tomando como ejemplo el árbol **A1** de nuestro ejemplo, el recorrido en amplitud nos daría la siguiente secuencia de nodos: (1, 2,3,4,5,6,7,8,9).
  - Recorrido en **PROFUNDIDAD**: La dimensión que prima al recorrer el árbol es la *vertical*. Siempre se tenderá a alejarse del nodo raíz mientras se pueda. Sin embargo, a diferencia del recorrido en amplitud, existen diferentes formas de recorrido en profundidad, estas formas son:
    - Profundidad en **PREORDEN**: Se empieza por el nodo raíz y se tiende a alejarse lo máximo posible de él. Moviéndose en vertical de arriba hacia abajo. En nuestro ejemplo obtendríamos la secuencia: *(1,2,5,3,6,4,7,8,9)*.
    - Profundidad en **POSTORDEN**: Se comienza por el nodo hoja más a la izquierda y se mantiene la máxima distancia que se pueda con el nodo raíz. En nuestro ejemplo se obtiene la secuencia: *(5,2,6,3,7,8,9,4,1)*.
    - Profundidad en **INORDEN**: Como el recorrido postorden pero cuidando que no aparezcan, siempre que sea posible, dos nodos del mismo padre (nodos hermanos) de forma consecutiva en la secuencia de recorrido. En nuestro ejemplo se tendría: *(5,2,1,6,3,7,4,8,9)*. Para los árboles binarios, en este tipo de recorrido, jamás pueden aparecer consecutivos dos nodos hermanos.
- **Insertar/Eliminar** elementos en el árbol: Al igual que en las listas, hay que distinguir entre insertar el nodo raíz, un nodo interno o un nodo hoja. También resulta muy importante nunca cambiar de forma incorrecta o perder un enlace, pues supondría la imposibilidad de acceder a parte o a todo el árbol.
- **Ordenar** los elementos del árbol: Ordenar un árbol no implica cambiar su estructura, sino modificar el contenido de los nodos para que sigan algún tipo de orden deseado. Las posibles ordenaciones de un árbol son aquellas que hacen que al hacer el recorrido de un árbol se obtenga una secuencia ordenada según algún criterio. Por ejemplo, nuestro árbol de ejemplo está ordenado en amplitud, pues al hacer un recorrido de este tipo se obtiene la secuencia ordenada *(1,2,3,4,5,6,7,8,9)*. El mismo árbol ordenado en profundidad preorden sería: *(1,2,3,4,5,6,7,8,9)*.



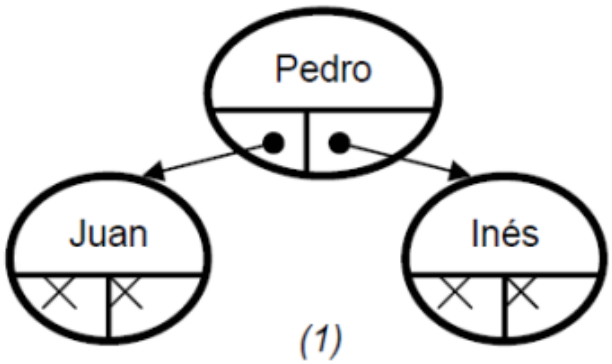
- **Buscar** elementos en el árbol: Consiste en localizar el/los nodo/s que contiene un dato igual al dato dado como referencia. Es muy importante darse cuenta de que según esté el árbol ordenado o no, si está balanceado o no, y dependiendo del recorrido que se haga, las búsquedas pueden ser más o menos efectivas (rápidas).
- **Contar** los elementos del árbol: Básicamente consiste en recorrer el árbol en cualquiera de sus formas e ir contando los elementos. Existen sin embargo variantes de esta contabilidad. Por ejemplo, podría solicitarse el número de nodos que tiene el nivel N de un árbol, lo cual exigiría un recorrido en amplitud teniendo en cuenta siempre el nivel donde se está en cada momento. Otra posible cuenta sería conocer el número de nodos que hay desde la raíz hasta el nodo hoja de menor nivel, etc.
- **Balancear** el árbol: Consiste en hacer que un árbol no-balanceado pase a ser balanceado. Esta operación sí que supone un cambio en la estructura del árbol, suponiendo siempre el cambio de nodo raíz y la variación de niveles de ciertos nodos. Veamos un ejemplo de balanceo de un árbol binario:

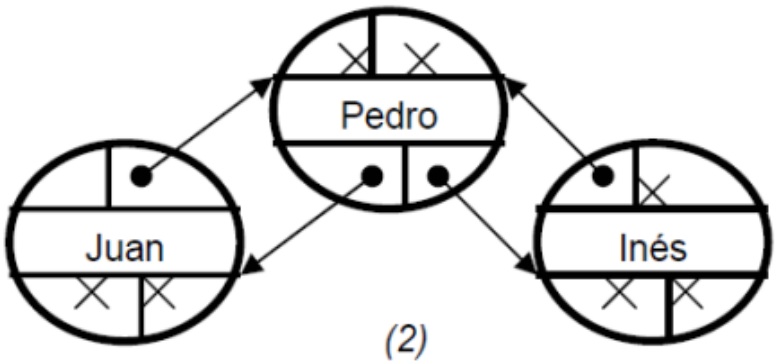


Representación

Ya hemos visto una primera forma de representar un árbol muy intuitiva y clarificadora, usando circunferencias para representar los nodos o elementos del árbol y líneas para representar los enlaces.

No obstante, sin tocar en absoluto la estructura del árbol, conviene considerar la forma de enlazar los nodos del árbol, pues al igual que con las listas, los nodos del árbol pueden estar simplemente enlazados o doblemente enlazados. Esta diferencia afecta en la facilidad o dificultad de implementar ciertas operaciones sobre los árboles, sobre todo las diferentes operaciones de recorrido del árbol. Cuando se quiere especificar en un gráfico el tipo de enlaces del árbol, se utiliza una forma parecida a las listas, veamos un ejemplo de un mismo árbol simplemente enlazado y doblemente enlazado.





Implantación

Al igual que las listas, con los árboles, según diversas consideraciones sobre el tamaño, manejo o rendimiento que queramos que tengan ciertas operaciones, tales como recorridos, ordenaciones, inserciones, etc. los árboles pueden implantarse con arrays, ficheros secuenciales o punteros. Estructuras de datos todas ellas muy comunes en la mayoría de los lenguajes de programación.

Lo más habitual es que se utilice memoria dinámica, es decir, punteros, para la implantación de todo tipo de árboles, aunque en ciertas ocasiones pueden considerarse la implantación con memoria estática mediante arrays.

Algoritmos: Ordenación, Búsqueda, Recursión, Grafos

La búsqueda de un elemento es, con diferencia, la operación más utilizada en los TAD, y la más importante. Por otra parte, la ordenación de los elementos del TAD puede resultar de gran ayuda a la hora de realizar búsquedas, así pues, la ordenación de los elementos del TAD es la segunda operación más útil. Se pueden distinguir dos tipos dentro de estos algoritmos:

- Algoritmos **Recursivos**: Los algoritmos recursivos son aquellos que se basan en el uso de rutinas de programación que se llaman a sí mismas.
- Algoritmos **Iterativos**: Se consideran algoritmos iterativos a todos aquellos que no son recursivos.

Ordenación

La finalidad de los algoritmos de ordenación es organizar ciertos datos (estos datos pueden estar contenidos en diferentes estructuras, ya sean TAD o en ficheros) en un orden creciente o decreciente mediante una regla prefijada (numérica, alfabética, …).

Atendiendo al tipo de elemento que se quiera ordenar puede ser:

- **Ordenación interna**: Los datos se encuentran en memoria (ya sean tablas, listas, árboles, etc).
- **Ordenación externa**: Los datos están en un dispositivo de almacenamiento externo (ficheros), y su ordenación es más lenta que la interna.

En este apartado vamos a estudiar los métodos de ordenación interna para los TAD vistos en apartados anteriores, esto es, para ordenar tablas, listas y árboles. Inicialmente nos centraremos en los TAD tabla y lista (simple o doble) y más adelante comentaremos el caso de listas múltiples y árboles.

Además, aquí consideraremos métodos de ordenación iterativa, es decir, no recursivos. Cuando estudiemos la recursividad veremos ejemplos de ordenaciones de tipo recursivo.

Los principales algoritmos de ordenación interna de tipo iterativo son:

- Selección
- Burbuja
- Inserción Directa
- Inserción Binaria
- Shell
- Intercalación

Selección

Este método consiste en buscar el elemento más pequeño del TAD y ponerlo en la primera posición; luego, entre los restantes, se busca el elemento más pequeño y se coloca en segundo lugar, y así sucesivamente hasta colocar el último elemento.

La idea de esta ordenación es independiente del tipo de TAD a la que se aplique, lo único que puede variar son ciertos aspectos o detalles en la forma explícita de hacer la ordenación según se implemente el TAD con memoria estática (arrays) o memoria dinámica (punteros).

Por ejemplo, si tenemos el array{40,21,4,9,10,35}, o la lista (40,21,4,9,10,35), los pasos a seguir son:

{4,21,40,9,10,35} ← Se coloca el 4, el más pequeño, en primera posición: se cambia el 4 por el 40.
{4,9,40,21,10,35} ← Se coloca el 9, en segunda posición: se cambia el 9 por el 21.
{4,9,10,21,40,35} ← Se coloca el 10, en tercera posición: se cambia el 10 por el 40.
{4,9,10,21,40,35} ← Se coloca el 21, en tercera posición: ya está colocado.
{4,9,10,21,35,40} ← Se coloca el 35, en tercera posición: se cambia el 35 por el 40.

Como se tienen N elementos, el número de comprobaciones que hay que hacer es de N*(N-1)/2, luego el coste del algoritmo es $O(N^2)$

Burbuja

Consiste en comparar pares de elementos adyacentes e intercambiarlos entre sí hasta que estén todos ordenados. Utilizando como ejemplo la tabla anterior {40,21,4,9,10,35}:

Primera pasada: Se comienza por el primer elemento.

{21,40,4,9,10,35} ← Se cambia el 21 por el 40.
{21,4,40,9,10,35} ← Se cambia el 40 por el 4.
{21,4,9,40,10,35} ← Se cambia el 9 por el 40.
{21,4,9,10,40,35} ← Se cambia el 40 por el 10.
{21,4,9,10,35,40} ← Se cambia el 35 por el 40.

Segunda pasada: Se comienza por el segundo elemento.

{4,21,9,10,35,40} ← Se cambia el 21 por el 4.
{4,9,21,10,35,40} ← Se cambia el 9 por el 21.
{4,9,10,21,35,40} ← Se cambia el 21 por el 10.

Llegado a este punto ya están ordenados, pero el algoritmo realmente acaba cuando se comienza el recorrido a partir del penúltimo elemento. Por tanto, si hay N elementos, para estar seguro de que el array está ordenado, hay que hacer N-1 pasadas, por lo que habría que hacer (N-1)*(N-i-1) comparaciones, para cada i desde 1 hasta N-1. El número de comparaciones es, por tanto, N(N-1)/2, lo que nos deja un coste del algoritmo, al igual que en la selección, de $O(N^2)$.

Inserción directa

En este método lo que se hace es tener una sublista ordenada de elementos de la lista, o array, e ir insertando el resto en el lugar adecuado para que la sublista no pierda el orden. La sublista ordenada se va haciendo cada vez mayor, de modo que al final la lista entera queda ordenada. Utilizando como ejemplo la tabla anterior {40,21,4,9,10,35}:

{40,21,4,9,10,35} ← La primera sublista ordenada es {40}.

Insertamos el 21:
{40,40,4,9,10,35} ← Usamos una variable auxiliar aux=21;
{21,40,4,9,10,35} ← Ahora la sublista ordenada es {21,40}.

Insertamos el 4:
{21,40,40,9,10,35} ← aux=4;
{21,21,40,9,10,35} ← aux=4;
{4,21,40,9,10,35} ← Ahora la sublista ordenada es {4,21,40}.

Insertamos el 9:
{4,21,40,40,10,35} ← aux=9;
{4,21,21,40,10,35} ← aux=9;
{4,9,21,40,10,35} ← Ahora la sublista ordenada es {4,9,21,40}.

Insertamos el 10:
{4,9,21,40,40,35} ← aux=10;
{4,9,21,21,40,35} ← aux=10;
{4,9,10,21,40,35} ← Ahora la sublista ordenada es {4,9,10,21,40}.

Y por último insertamos el 35:
{4,9,10,21,40,40} ← aux=35;
{4,9,10,21,35,40} ← Ya está ordenado.

En el peor de los casos, el número de comparaciones que hay que realizar es de N*(N+1)/2-1, lo que nos deja un tiempo de ejecución en $O(N^2)$. En el mejor caso (cuando la lista ya estaba ordenada), el número de comparaciones es (N-2), y todas ellas son falsas, con lo que no se produce ningún intercambio. El tiempo de ejecución está en O(N).

Inserción binaria

Es el mismo método que la inserción directa, excepto que la búsqueda del orden de un elemento en la sublista ordenada se realiza mediante una búsqueda binaria, lo que en principio supone un ahorro de tiempo. No obstante, dado que para la inserción sigue siendo necesario un desplazamiento de los elementos, el ahorro, en la mayoría de los casos, no se produce, si bien hay compiladores que realizan optimizaciones que lo hacen ligeramente más rápido.

Shell

Es una mejora del método de inserción directa, utilizado cuando la tabla tiene un gran número de elementos. En este método no se compara a cada elemento con el de su izquierda, como en el de inserción, sino con el que está a un cierto número de lugares (llamado salto) a su izquierda. Este salto es constante, y su valor inicial es N/2 (siendo N el número de elementos, y siendo división entera). Se van dando pasadas hasta que en una pasada no se intercambie ningún elemento de sitio. Entonces el salto se reduce a la mitad,

y se vuelven a dar pasadas hasta que no se intercambie ningún elemento, y así sucesivamente hasta que el salto vale 1. Utilizando como ejemplo {40,21,4,9,10,35}:

Salto=3:

Primera pasada:
{9,21,4,40,10,35} ← Se intercambian el 40 y el 9.
{9,10,4,40,21,35} ← Se intercambian el 21 y el 10.

Salto=1:

Primera pasada:
{9,4,10,40,21,35} ← Se intercambian el 10 y el 4.
{9,4,10,21,40,35} ← Se intercambian el 40 y el 21.
{9,4,10,21,35,40} ← Se intercambian el 35 y el 40.

Segunda pasada:
{4,9,10,21,35,40} ← Se intercambian el 4 y el 9.

Con sólo 6 intercambios se ha ordenado el array, cuando por inserción se necesitaban muchos más.

Intercalación

No es propiamente un método de ordenación, consiste en la unión de dos tablas o listas ordenadas de modo que la unión esté también ordenada. Para ello, basta con recorrer los TAD de izquierda a derecha e ir cogiendo el menor de los dos elementos, de forma que sólo aumenta el contador del array del que sale el elemento siguiente para el array-suma. Si quisiéramos sumar las tablas {1,2,4} y {3,5,6}, los pasos serían:

Inicialmente:	i1=0, i2=0, is=0.
Primer elemento:	mínimo entre 1 y 3 = 1. Suma={1}. i1=1, i2=0, is=1.
Segundo elemento:	mínimo entre 2 y 3 = 2. Suma={1,2}. i1=2, i2=0, is=2.
Tercer elemento:	mínimo entre 4 y 3 = 3. Suma={1,2,3}. i1=2, i2=1, is=3.
Cuarto elemento:	mínimo entre 4 y 5 = 4. Suma={1,2,3,4}. i1=3, i2=1, is=4.

Como no quedan elementos del primer array, basta con poner los elementos que quedan del segundo array en la suma:

Suma = {1,2,3,4} + {5,6} = {1,2,3,4,5,6}

Búsqueda

Como se ha mencionado anteriormente la búsqueda es la operación más importante en el procesamiento de la información, y permite la recuperación de datos previamente almacenados. El tipo de búsqueda se puede clasificar como interna o externa, según el lugar en el que esté almacenada la información (en memoria o en dispositivos externos). Todos los algoritmos de búsqueda tienen dos finalidades:

- Determinar si el elemento buscado se encuentra en el conjunto en el que se busca.
- Si el elemento está en el conjunto, hallar la posición en la que se encuentra.

En este apartado nos centraremos en la búsqueda interna iterativa. Como principales algoritmos en tablas y listas tenemos las búsquedas:

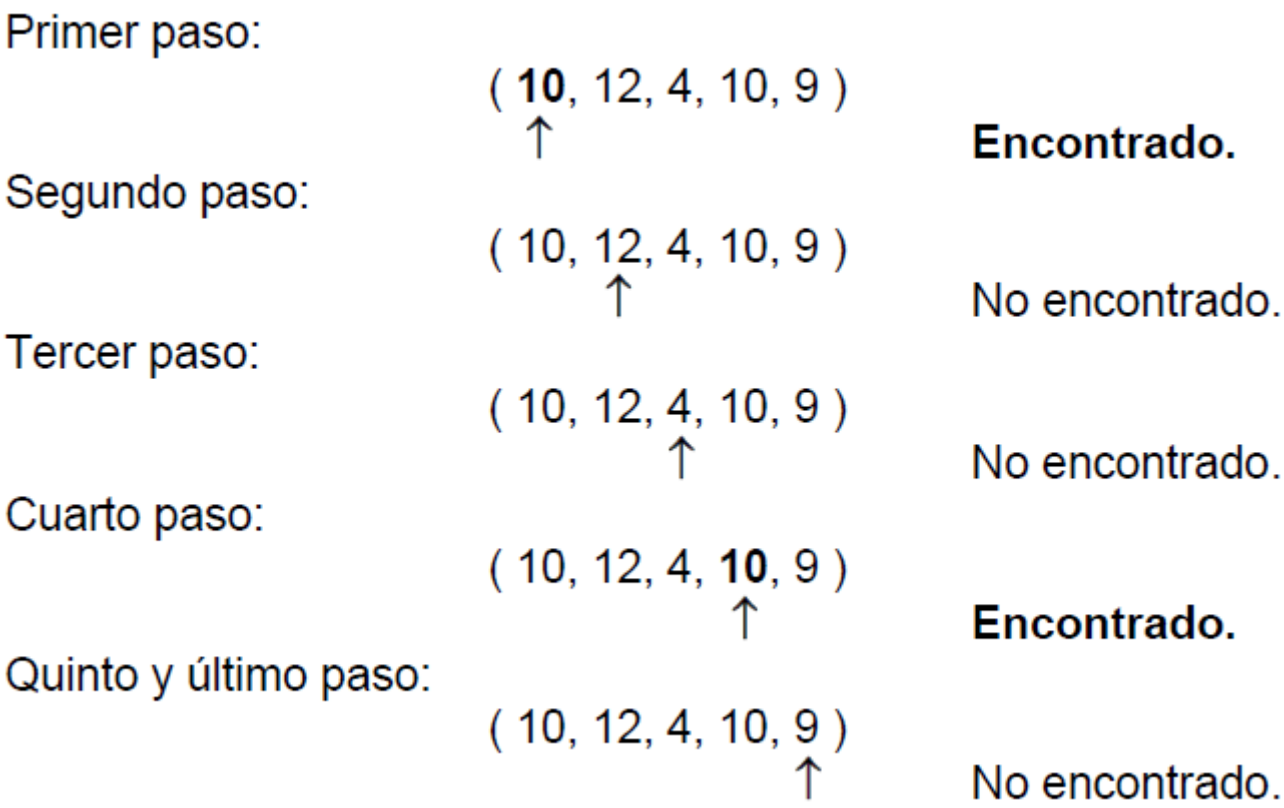
- Secuencial
- Binario o Dicotómica
- Utilizando tablas Hash

Búsqueda Secuencial

Esta búsqueda consiste en recorrer y examinar cada uno de los elementos hasta encontrar el o los elementos buscados, o hasta que se han mirado todos los elementos.

Por ejemplo, para la lista (10,12,4,10,9) donde queremos encontrar el elemento cuyo contenido es 10, la idea del algoritmo sería:

Utilizando como ejemplo la tabla anterior {40,21,4,9,10,35}:



Puesto que queremos encontrar todas las ocurrencias del valor 10, es necesario recorrer siempre toda la lista o tabla, así pues habrá que recorrer los N elementos, lo cual hace que el coste del algoritmo sea N, y su orden O(N).

Si sólo queremos encontrar la primera ocurrencia que se produzca, si tenemos la certeza de que no puede haber elementos repetidos, se puede parar la búsqueda en cuanto se produzca la primera ocurrencia, con lo cual el algoritmo es más eficiente. En este caso, el número medio de comparaciones que hay que hacer antes de encontrar el elemento buscado es de (N+1)/2. Aún así, el orden sigue siendo O(N).

Búsqueda binaria o dicotómica

Para utilizar este algoritmo, se precisa que la tabla o lista considerada esté ordenada. La búsqueda binaria consiste en dividir la tabla por su elemento medio en dos subtablas más pequeñas, y comparar el elemento con el del centro. Si coinciden, la búsqueda se termina. Si el elemento es menor, debe estar (si está) en la primera subtabla, y si es mayor está en la segunda.

Por ejemplo, para buscar el elemento 3 en {1,2,3,4,5,6,7,8,9} se realizarían los siguientes pasos:

Se toma el elemento central y se divide el array en dos:
{1,2,3,4}-5-{6,7,8,9}
Como el elemento buscado (3) es menor que el central (5), debe estar en el primer subtabla: {1,2,3,4}

Se vuelve a dividir el array en dos:
{1}-2-{3,4}
Como el elemento buscado es mayor que el central, debe estar en el segundo subtabla: {3,4}

Se vuelve a dividir en dos:
{}-3-{4}
Como el elemento buscado coincide con el central, lo hemos encontrado.

Si al final de la búsqueda todavía no lo hemos encontrado, y la subtabla a dividir está vacía {}, quiere decir que el elemento no se encuentra en la tabla.

En general, este método realiza {log2 (N+1)} comparaciones antes de encontrar el elemento, o antes de descubrir que no está. Este número es muy inferior que el necesario para la búsqueda lineal para casos grandes.

Este método también se puede implementar de forma recursiva, siendo la función recursiva la que divide la tabla o lista.

Búsqueda utilizando Tablas Hash

Este método no es realmente un método de búsqueda, sino una forma de mejorar la velocidad de búsqueda al utilizar algún otro método.

Consiste en asignar a cada elemento un índice mediante una transformación del elemento. Esta correspondencia se realiza mediante una función de conversión, llamada función hash. La correspondencia más sencilla es la identidad, esto es, al número 0 se le asigna el índice 0, al elemento 1 el índice 1, y así sucesivamente.

Pero si los números a almacenar son demasiado grandes esta función es inservible. Por ejemplo, se quiere guardar en un array la información de 1000 usuarios de una empresa, y se elige el número de DNI como elemento identificativo. Es inviable hacer un array de 100.000.000 elementos, sobre todo porque se desaprovecha demasiado espacio. Por eso, se realiza una transformación al número de DNI para que nos de un número menor, de 3 cifras pues hay 1000 usuarios, y utilizar este resultado de la función hash como índice de un array de 1000 elementos para guardar a los empleados.

Así, dado un DNI a buscar, bastaría con realizar la transformación según la función hash a un número de 3 cifras; usar este número como índice de búsqueda del array de 1000 elementos, con cualquiera de los métodos anteriores, y obtener el contenido del array dado por la posición cuyo índice es el resultado de la función hash.

La función de hash ideal debería ser biyectiva, esto es, que a cada elemento le corresponda un índice, y que a cada índice le corresponda un elemento, pero no siempre es fácil encontrar esa función, e incluso a veces es inútil, ya que puede no saberse “a priori” el número de elementos a almacenar.

La función de hash depende de cada problema y de cada finalidad, y se pueden utilizar con números o cadenas, pero las más utilizadas son:

- *Restas sucesivas*: Esta función se emplea con claves numéricas entre las que existen huecos de tamaño conocido, obteniéndose direcciones consecutivas.

- *Aritmética modular*: El índice de un número es resto de la división de ese número entre un número N prefijado, preferentemente primo. Los número se guardarán en las direcciones de memoria de 0 a N-1. Este método tiene el problema de que cuando hay (N+1) elementos, al menos un índice es señalado por dos elementos (teorema del palomar). A este fenómeno se le llama colisión.
- *Mitad del cuadrado*: Consiste en elevar al cuadrado la clave y coger las cifras centrales. Este método también presenta problemas de colisión.
- *Truncamiento*: Consiste en ignorar parte del número y utilizar los elementos restantes como índice. También se produce colisión.
- *Plegamiento*: Consiste en dividir el número en diferentes partes, y operar con ellas (normalmente con suma o multiplicación). También se produce colisión.

Ahora se nos presenta el problema de qué hacer con las colisiones, es decir, el **Tratamiento de Colisiones**. ¿Qué pasa cuando a dos elementos diferentes les corresponde el mismo índice?. Pues bien, hay tres posibles soluciones:

- Cuando el índice correspondiente a un elemento ya está ocupado, se le asigna el primer índice libre a partir de esa posición. Este método es poco eficaz, porque al nuevo elemento se le asigna un índice que podrá estar ocupado por un elemento posterior a él, y la búsqueda se ralentiza, ya que no se sabe la posición exacta del elemento.
- También se pueden reservar unos cuantos lugares al final del array para alojar a las colisiones. Este método también tiene un problema: ¿Cuánto espacio se debe reservar? Además, sigue la lentitud de búsqueda si el elemento a buscar es una colisión.
- Lo mejor es en vez de crear un array de números, crear un array de punteros, donde cada puntero señala el principio de una lista enlazada. Así, cada elemento que llega a un determinado índice se pone en el último lugar de la lista de ese índice. El tiempo de búsqueda se reduce considerablemente, y no hace falta poner restricciones al tamaño del array, ya que se pueden añadir nodos dinámicamente a la lista.

Recursividad

Definición de recursividad

La **recursividad** es una característica que permite que una determinada acción se pueda realizar en función de invocar la misma acción pero en un caso más sencillo, hasta llegar a un punto (caso base) donde la realización de la acción sea muy sencilla. Esta forma de ver las cosas es útil para resolver problemas definibles en sus propios términos. En cierta medida, es análogo al principio de inducción.

Para desarrollar algoritmos recursivos hay que partir del supuesto de que ya hay un algoritmo que resuelve una versión más sencilla del problema. A partir de esta suposición debe hacerse lo siguiente:

- Identificar subproblemas atómicos de resolución inmediate. Los denominados *casos base*.
- Descomponer el problema en subproblemas resolubles mediante el algoritmo preexistente; la solución de estos subproblemas debe aproximarnos a los casos base.

No existen problemas intrínsecamente recursivos o iterativos; cualquier proceso iterativo puede expresarse de forma recursiva y viceversa. Si bien ciertos problemas se prestan mucho mejor que otros al uso de la recursividad.

En el mundo de las matemáticas, un clásico ejemplo de recursividad lo tenemos en el cálculo del factorial de un número, a saber:

```
Factorial del número natural n (incluido el 0) = n!
     (1) si n = 0 entonces: 0! = 1
     (2) si n > 0 entonces: n! = n · (n-1)!
```

Aunque en este estudio de los TAD hemos evitado el uso de lenguajes de programación específicos, por ser una de las características de los tipos abstractos de datos la independencia de su implantación; en esta ocasión recurriremos al lenguaje C para poner un ejemplo muy claro e ilustrativo de cómo implementar un algoritmo recursivo. El siguiente programa es un ejemplo del cálculo del factorial de un número n:

```
int factorial(int n)
{
     if (n == 0) return 1;             // Caso base
     return (n * factorial(n-1));      // Llamada a sí mismo
}
```

La función factorial es llamada pasándole un determinado entero y devuelve otro numero entero. Como se observa, en cada llamada recursiva se reduce el valor de n, llegando el caso en el que n es 0 y no efectúa más llamadas recursivas.

Aunque la función factorial se preste muy bien al uso de la recursividad, no quiere decir esto, tal como hemos mencionado anteriormente, que no pueda ser implementado de forma iterativa. De hecho el factorial puede obtenerse con facilidad sin necesidad de emplear funciones recursivas, es más, el uso del programa anterior es muy ineficiente (con un número n grande, al ejecutarse en una computadora, consumiría mucha más memoria y tiempo que si se usase un algoritmo iterativo), pero es un ejemplo muy claro.

Uso de la recursión

La pregunta que surge de manera natural es: *¿Cuándo utilizar la recursividad?*. No se debe utilizar si la solución iterativa es sencilla y clara. En otros casos, obtener una solución iterativa es mucho más complicado que una solución recursiva, y entonces se planteará si merece la pena transformar la solución recursiva en una iterativa.

Por otra parte, hay TAD que, debido a sus características, sus operaciones se adaptan muy bien al uso de la recursión. Casi todos los algoritmos basados en los esquemas de vuelta atrás y divide y vencerás son recursivos. Otras estructuras que se adaptan muy bien a la recursividad son los grafos, y cuando se habla de grafos se está incluyendo a las listas y a los árboles.

Si nos centramos en el mundo de la programación, algunos lenguajes de programación no admiten el uso de recursividad. Es obvio que en este caso se requerirá una solución no recursiva (iterativa). Aunque parezca mentira, es en general mucho más sencillo escribir un programa recursivo que su equivalente iterativo. Y desde luego siempre resulta más económico en cantidad de líneas de código.

Ordenaciones y búsquedas recursivas

Ya hemos visto en apartados anteriores como se realizan ordenaciones y búsquedas iterativas en TAD tipo tabla y lista. Veamos ahora algoritmos de naturaleza recursiva para realizar estas operaciones.

La **Ordenación Rápida (Quicksort)** es un algoritmo de ordenación ilustrativo del uso de la recursividad en ordenaciones. Este método se basa en la táctica “divide y vencerás”, que consiste en ir subdividiendo el TAD considerado, en nuestro caso una tabla o lista en partes más pequeñas, y ordenar éstas.

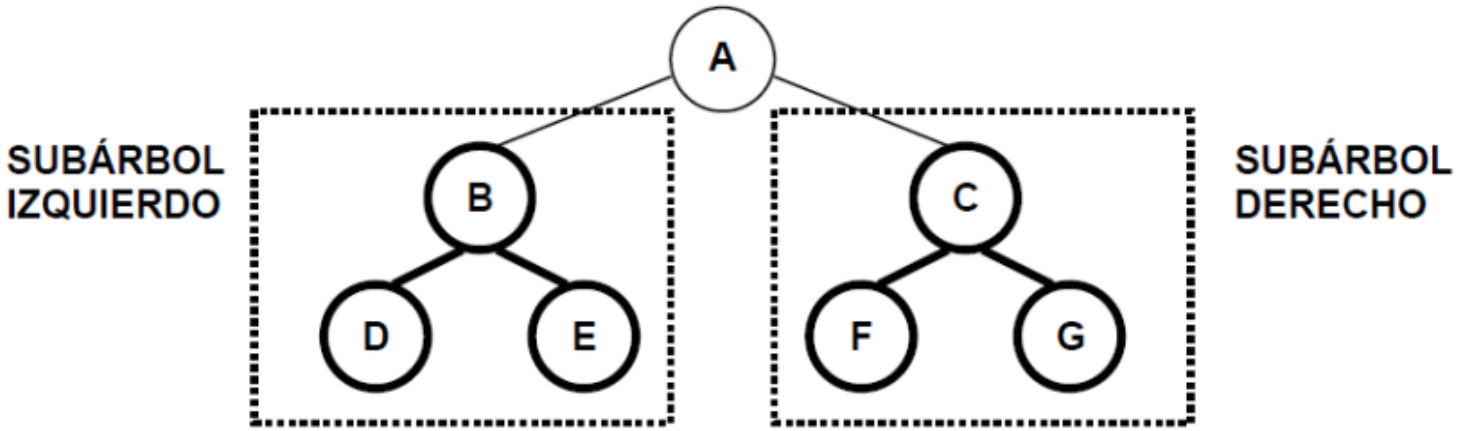
Para hacer esta división, se toma el valor como pivote, y se mueven todos los elementos menores que este pivote a su izquierda, y los mayores a su derecha. A continuación se aplica el mismo método a cada una de las dos partes en las que queda dividido el array.

Normalmente se toma como pivote el primer elemento del array, y se realizan dos búsquedas: una de izquierda a derecha, buscando un elemento mayor que el pivote, y otra de derecha a izquierda, buscando un elemento menor que el pivote. Cuando se han encontrado los dos, se intercambian, y se sigue realizando la búsqueda hasta que las dos búsquedas se encuentran. Por ejemplo, para dividir el array {21,40,4,9,10,35}, los pasos serían:

{21,40,4,9,10,35} ← Se toma como pivote el 21. La búsqueda de izquierda a derecha encuentra el valor 40, mayor que pivote, y la búsqueda de derecha a izquierda encuentra el valor 10, menor que el pivote.

Se intercambian:
{21,10,4,9,40,35} ← Si seguimos la búsqueda, la primera encuentra el valor 40, y la segunda el valor 9, pero ya se han cruzado, así que paramos. Para terminar la división, se coloca el pivote en su lugar (en el número encontrado por la segunda búsqueda, el 9, quedando:
{9,10,4,21,40,35} ← Ahora tenemos dividido el array en dos arrays más pequeños: el {9,10,4} y el {40,35}, y se repetiría el mismo proceso.

Respecto a los árboles, la idea fundamental que hace posible el uso de la recursividad es que realmente un árbol puede considerarse como un nodo raíz del que cuelgan otros árboles, llamados subárboles. Véase la siguiente figura como ejemplo de esta idea de un árbol binario:



Para realizar cualquier exploración, recorrido o búsqueda en un árbol binario (o cualquier árbol genérico) bastaría con aplicar la misma operación a cada uno de sus subárboles, hasta llegar al caso base, que sería la búsqueda o recorrido en un subárbol formado por el nodo raíz y nodos hojas, pues en este caso la búsqueda o recorrido es muy fácil, basta seguir los enlaces del nodo raíz para visitar los nodos hojas. Una vez hecho esto el algoritmo recursivo consiste en hacer lo mismo, esto es, seguir los enlaces del nodo raíz, pero tomando como nodo raíz el nodo padre de los nodos caso base. Se repite el proceso hasta llegar al nodo raíz del árbol completo.

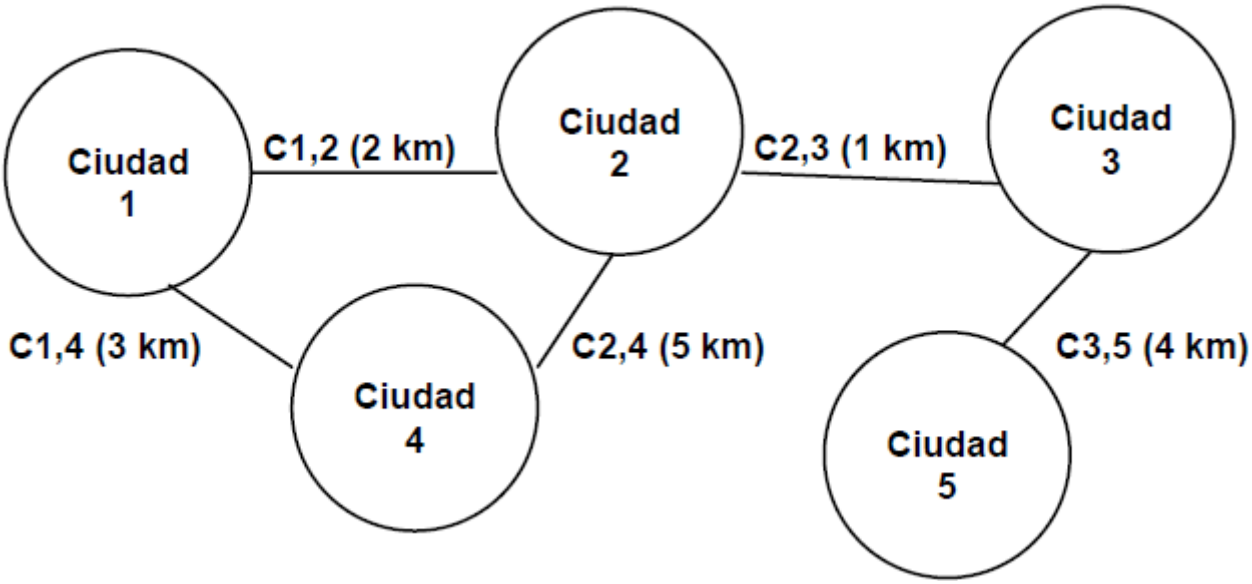
En nuestro ejemplo, los casos bases se darían al llegar a los nodos B y C. Ambos nodos son casos base pues únicamente tienen nodos hojas. Una vez visto sus enlaces y sus nodos hojas se realizaría la misma operación pero tomando el nodo padre de B y C, el nodo A, y tratando a los nodos B y C como si fuesen hojas.

Veremos en el siguiente apartado sobre grafos que un TAD tipo árbol no es más que un tipo particular de grafo. Y con respecto a estos, comprobaremos que sus operaciones, entre las que está la búsqueda y la ordenación, son de índole profundamente recursiva.

Grafos

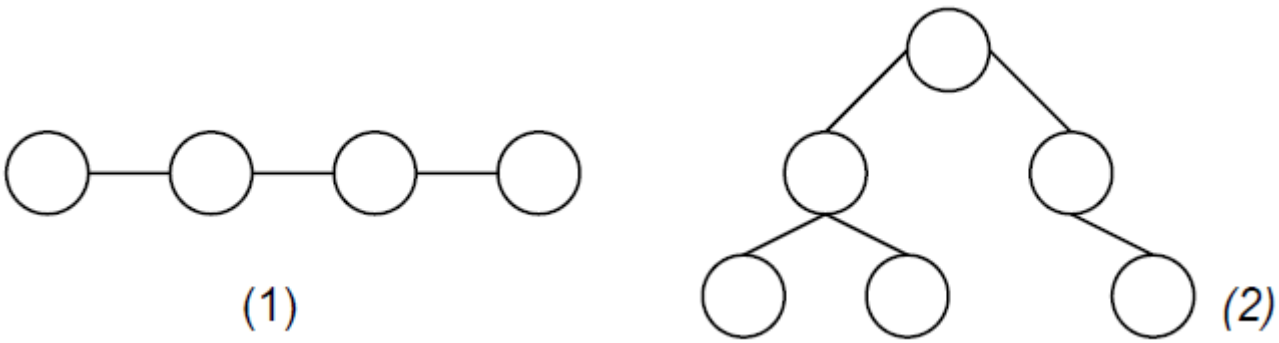
Un grafo es un objeto matemático que se utiliza para representar circuitos, redes, caminos, etc. Los grafos son muy utilizados en computación, ya que permiten resolver problemas muy complejos.

Supongamos el siguiente ejemplo. Disponemos de una serie de ciudades y de carreteras que las unen. De cada ciudad saldrán varias carreteras, por lo que para ir de una ciudad a otra se podrán tomar diversos caminos. Cada carretera tendrá un coste asociado (por ejemplo, la longitud de la misma). Gracias a la representación por grafos podremos elegir el camino más corto que conecta dos ciudades, determinar si es posible llegar de una ciudad a otra, si desde cualquier ciudad existe un camino que llegue a cualquier otra, etc. Para tener una idea visual de lo expuesto, supongamos que representamos las ciudades como circunferencias y los caminos por líneas que unen las distintas circunferencias (ciudades):



Una primera cosa que salta a la vista es que tanto una lista como un árbol no son más que un caso particular de grafo, en donde hemos puesto ciertas restricciones.

Veamos los grafos correspondientes a una lista y un árbol binario:



Componentes de un grafo

Así pues, un grafo consta de:

- **Vértices** (o nodos): Los vértices son objetos que contienen información. Para representarlos se suelen utilizar puntos o circunferencias con el contenido del nodo escrito dentro. En nuestro ejemplo serían las circunferencias que representan las ciudades.

- **Aristas**: Son las conexiones entre vértices. Para representarlos se utilizan líneas. Es frecuente añadir junto a la línea el nombre de los nodos origen y destino y el peso de la arista en algún tipo de unidad.

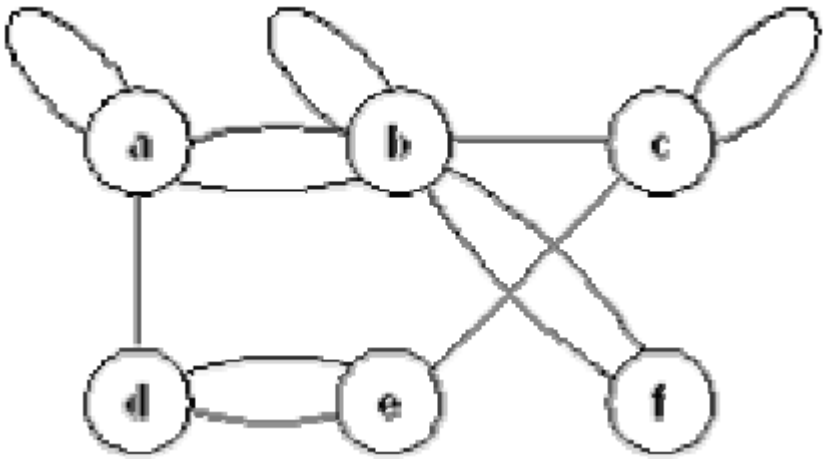
Definición formal de grafo

Conviene decir que la definición de un grafo no depende de su representación. Desde un punto de vista matemático puramente formal, la definición de grafo es la siguiente:

Un grafo G es un par (V(G), A(G)), donde V(G) es un conjunto no vacío de elementos llamados vértices, y A(G) es una familia finita de pares no ordenados de elementos de V(G) llamados aristas.

Al ser una familia de aristas se permite la posibilidad de aristas múltiples en el grafo, es decir, la existencia de más de una arista con el mismo par de vértices como origen y destino. También se permite la existencia de aristas bucles, con inicio y destino el mismo vértice.

Por ejemplo, según lo dicho y utilizando la forma de representación indicada en el apartado anterior, se puede observar que el conjunto de vértices *V(G){a, b, c, d, e, f}* y el de aristas *A(G)* formado por los pares *{a, b}, {a, b}, {b, c}, {a, d}, {d, e}, {d, e}, {b, f}, {b, f}, {c, e}, {a, a}, {b, b} y {c, c}* determinan el grafo de la figura siguiente:



Conceptos fundamentales sobre grafos

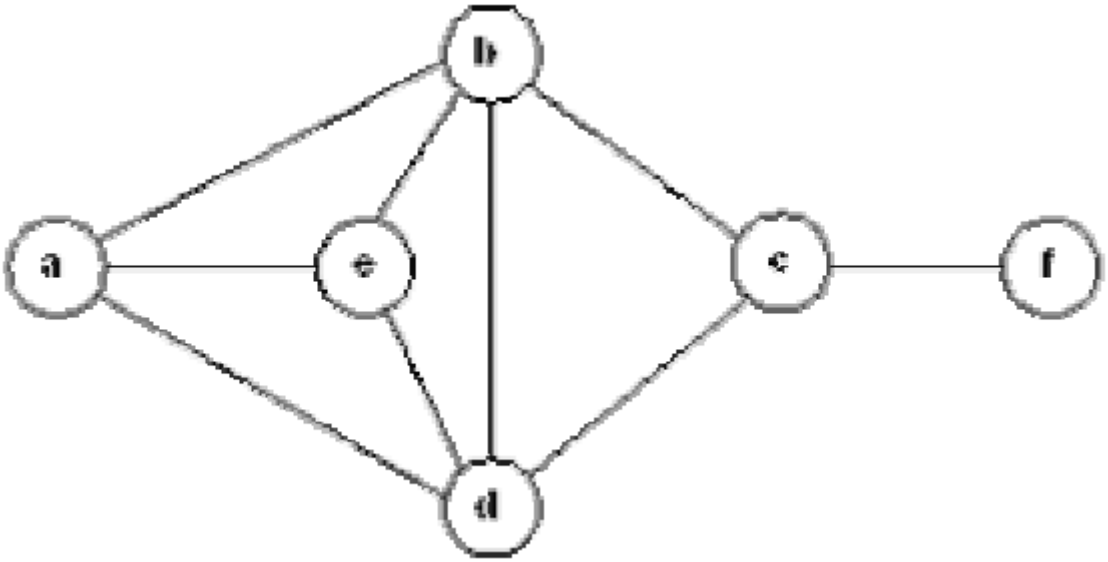
Algunos conceptos fundamentales al hablar de grafos son:

- Un **CAMINO** o **CIRCUITO** entre dos vértices es una lista de vértices en la que dos elementos sucesivos están conectados por una arista del grafo.

Desde un punto de vista formal, cualquier secuencia finita de aristas de G de la forma (v0v1), (v1v2), (v2v3), …, (vm-1vm) será denominada *camino* o *circuito* de G. Es frecuente que cuando entre dos vértices, por ejemplo los vértices “v” y “w”, existe uno o más caminos, sehable de cualquiera de estos caminos como un camino “vw”.

- Se habla de **GRAFO CONEXO** si existe un camino desde cualquier nodo del grafo hasta cualquier otro. Si no es conexo constará de varias *componentes conexas*. Formalmente, un grafo G es *conexo* si para cualquier par de vértices “v”, “w” de G existe un camino de “v” a “w” (“vw”).
- Un **PUENTE** o **ARISTA PUENTE** será cualquier arista de un grafo conexo que mediante su eliminación deje al grafo dividido en dos componentes conexas.

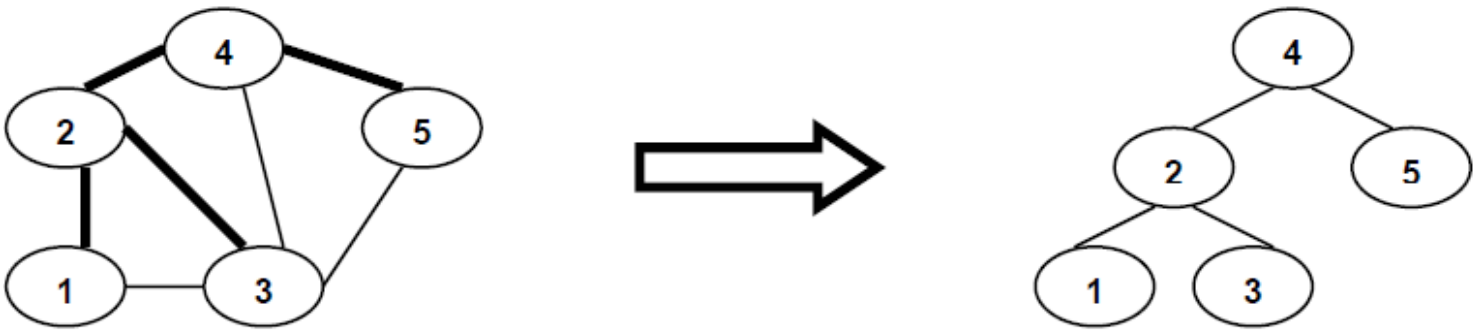
En el grafo de la figura siguiente la arista {c, f} será una arista puente que dejará el grafo G dividido en las componentes conexas formada por los vértices {f} y {a, b, c, d, e}.



- Un **CAMINO SIMPLE** es un camino desde un nodo a otro en que ningún nodo se repite (no se pasa dos veces). Si el camino simple tiene como primer y último elemento al mismo nodo se denomina *ciclo*.

Cuando el grafo no tiene ciclos tenemos un árbol. Varios árboles independientes forman un *bosque*.

- Un **ÁRBOL DE EXPANSIÓN** de un grafo es una reducción del grafo en el que solo entran a formar parte el número mínimo de aristas que forman un árbol y conectan a todos los nodos. Por ejemplo:



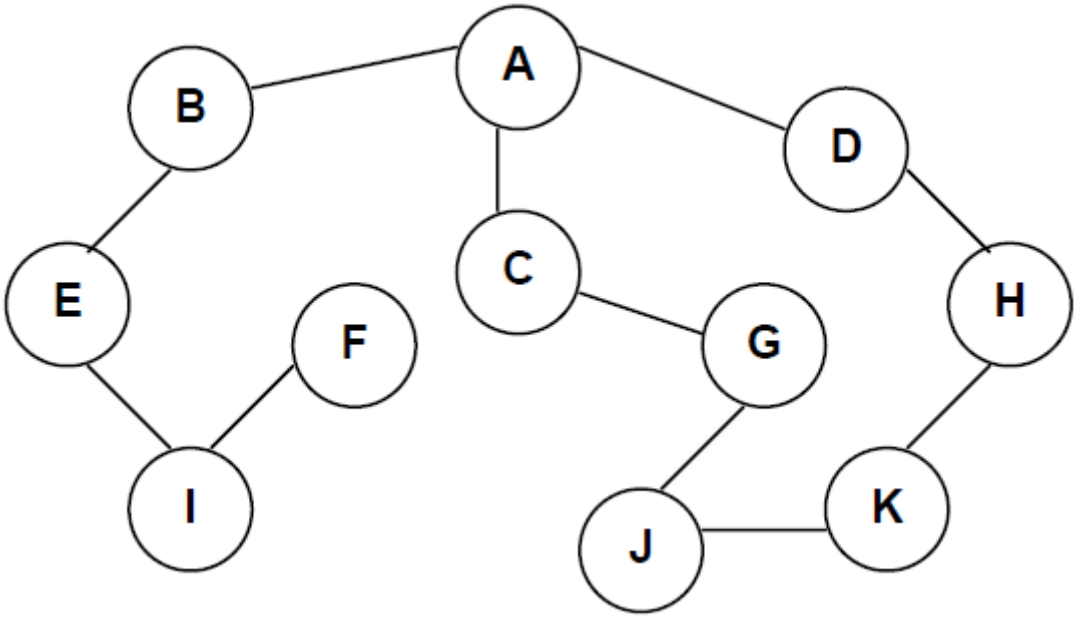
- Según el número de aristas que contiene, se habla de **GRAFO COMPLETO** si cuenta con todas las aristas posibles (es decir, todos los nodos están conectados con todos), **GRAFO DISPERSO** si tiene relativamente pocas aristas y **GRAFO DENSO** si le faltan pocas para ser completo.
- Las aristas son la mayor parte de la veces bidireccionales, es decir, si una arista conecta dos nodos A y B se puede recorrer tanto en sentido hacia B como en sentido hacia A. Estos son llamados **GRAFOS NO DIRIGIDOS**. Sin embargo, en ocasiones tenemos que las uniones son unidireccionales. Estas uniones se suelen dibujar con una flecha y definen un **GRAFO DIRIGIDO**.
- Se habla de **GRAFO PONDERADO** cuando las aristas llevan un coste asociado (un entero denominado **peso**).
- Una **RED** es un grafo dirigido y ponderado.

Exploración de grafos

Cuando se habla de la exploración de un grafo nos referimos a la exploración de todos los vértices (con algún tipo de fin, por ejemplo, un recuento) o hasta que se encuentra uno determinado, es decir, una búsqueda.

El orden en que los vértices éstos son “visitados” decide radicalmente el tiempo de ejecución del algoritmo.

Supongamos el siguiente grafo de ejemplo:



A la hora de explorar el grafo de la figura anterior, nos encontramos con dos métodos distintos:

- Exploración o búsqueda en **Anchura** o **Amplitud**: Lo que prima en la exploración es la dimensión horizontal, de manera que por cada vértice del nodo visitamos primeramente todos los nodos enlazados directamente con él (vecinos), y una vez hecho esto realizamos la misma operación con estos nodos vecinos visitados.
- Exploración o búsqueda en **Profundidad**: Lo que prima en la exploración es la dimensión vertical. Una vez visitado uno de los nodos vecinos de un nodo, antes de visitar a cualquiera de los demás vecinos del nodo, se vuelve a realizar la misma operación con el nodo vecino recién visitado.

Suponiendo que el orden en que están almacenados los nodos en la estructura de datos correspondiente es A-B-C-D-E-F-G-H-I-J-K (en orden alfabético), tenemos que:

- En un recorrido en anchura el orden sería: *A-B-C-D-E-G-H-I-J-K-F*
- El orden que seguiría el recorrido en profundidad será: *A-B-E-I-F-C-G-J-K-H-D*

Implantación de la exploración de grafos

En el ejemplo anterior, es destacable que el nodo D es el último en explorarse en la búsqueda en profundidad pese a ser adyacente al nodo de origen (A). Esto es debido a que primero se explora la rama del nodo C, que también conduce al nodo D.

Es decir, hay que tener en cuenta que es fundamental el orden en que los nodos están almacenados en las estructuras de datos. Si, por ejemplo, el nodo D estuviera antes que el C, en la búsqueda en profundidad se tomaría primero la rama del D (con lo que el último en visitarse sería el C), y en la búsqueda en anchura se exploraría antes H que el G.

Otro punto a observar es que, cuando hemos hablado en el apartado anterior de exploraciones en anchura y en profundidad, aparecen las siguientes frases “una vez hecho esto realizamos la misma operación” y “se vuelve a realizar la misma operación”. Esto está sugiriendo de forma muy clara que las exploraciones en grafos, tanto en anchura como en profundidad tienen una fuerte naturaleza recursiva. De hecho, es frecuente que se implanten ambos algoritmos usando recursividad.

Esto no quiere decir que no puedan implementarse exploraciones de grafos de forma iterativa. Usando algoritmos iterativos, la diferencia principal entre implementar una exploración en anchura frente a una en profundidad está en la estructura de datos usada, de forma que:

- Las *exploraciones* o *búsquedas en anchura* iterativas usa una estructura de datos tipo *cola*, pues las características de este tipo de estructura FIFO (primero en entrar, primero en salir) se adaptan muy bien a este tipo de recorrido.
- Las *exploraciones* o *búsquedas en profundidad* iterativas usa una estructura de datos tipo *pila*, pues las características de este tipo de estructura LIFO (último en entrar, primero en salir) se adaptan muy bien a este tipo de recorridos.

Organizaciones de Ficheros

Todas las aplicaciones necesitan almacenar y recuperar información. En una computadora, cuando se ejecuta una aplicación (un proceso), la información almacenada en la memoria principal electrónica del computador; este es un tipo de memoria volátil, de forma que cuando la aplicación termina la información se pierde. Esto es inaceptable para muchas aplicaciones, que pueden requerir que la información permanezca disponible durante largos periodos de tiempo.

Con respecto a la memoria principal de las computadoras, se trata de un tipo de memoria electrónicas cuyas principales características son:

- La memoria principal tiene poca capacidad de almacenamiento. No se pueden manipular grandes cantidades de datos, ya que puede haber casos en los que no quepan en la memoria principal.
- La memoria principal es volátil.
- Acceso rápido a la información.

Otro problema es que varios procesos pueden necesitar acceder a una misma información de forma concurrente. Como los espacios de memoria de los procesos son privados, un proceso no puede acceder a los datos en el espacio de memoria de otro. La solución es hacer que la información sea independiente de los procesos.

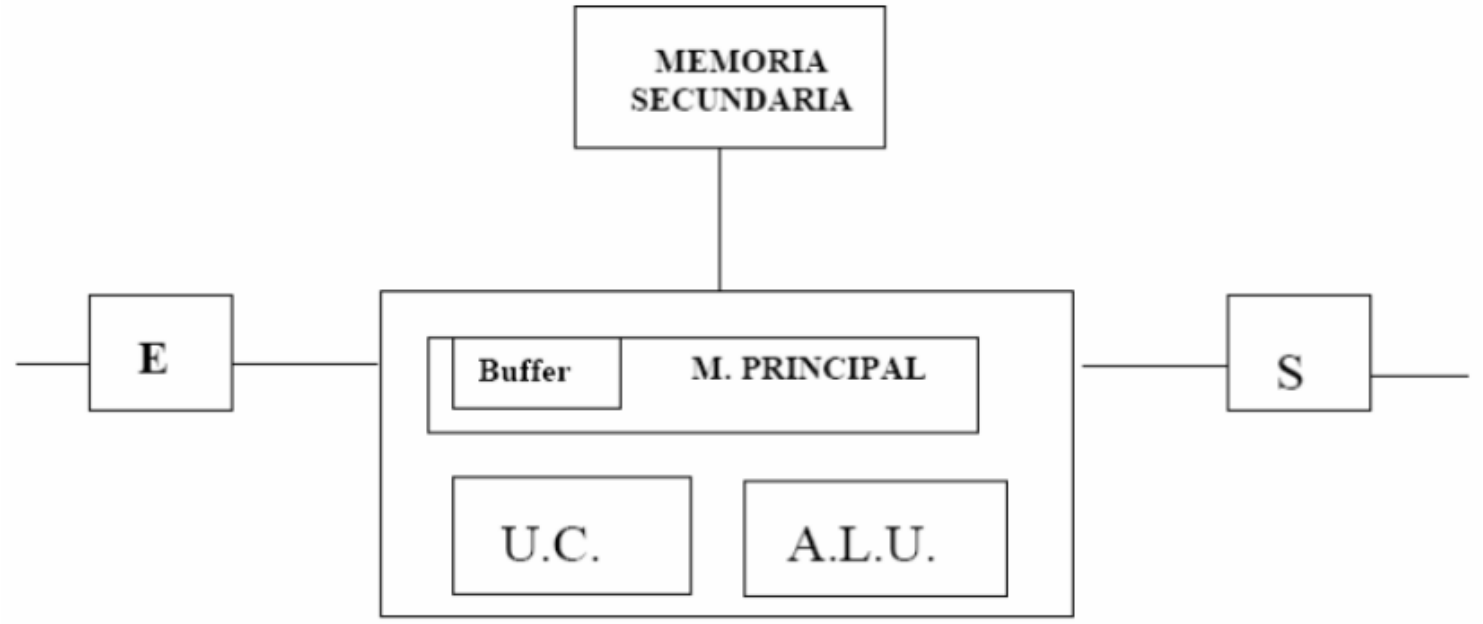
Por tanto, hay tres requisitos esenciales para almacenar la información en discos magnéticos u otros dispositivos en una unidades llamadas **ficheros** o **archivos**.

Un fichero es una abstracción de un mecanismo que permite almacenar información en un dispositivo y leerla posteriormente. Podemos definir un fichero como una colección de información que tiene un nombre.

Los ficheros pueden ser leídos y escritos por cualquier proceso: son una forma de almacenamiento denominada **memoria secundaria**. Sus principales cualidades son:

- Capacidad de almacenamiento sólo limitada por el soporte físico de que se disponga.
- La información está almacenada permanentemente.
- Acceso lento a la información, ya que tiene que ser transportada desde el dispositivo externo hasta la memoria principal para su tratamiento. Existe un área de memoria principal destinada a recibir esta información procedente del dispositivo secundario. Esta área se denomina *Buffer*.

Gráfico de Memoria Principal y Memoria Secundaria:



En la figura anterior representamos de manera muy esquemática la forma de operar de un procesador, siendo:

- **U.C. U**nit **C**ontrol (Unidad de Control). Circuito principal de control del procesador.
- **A.L.U. A**ritmetic **L**ogic **U**nit (Unidad Aritmético Lógica). Circuito especializado en operaciones aritméticas.
- **E**: Datos de **E**ntrada.
- **S**: Datos de **S**alida.

La información almacenada en ficheros debe ser **persistente**, es decir, no debe verse afectada por la creación y finalización de los procesos. La gestión de ficheros es tarea del SO, y la parte del mismo que realiza dicha gestión se conoce como **sistema de ficheros**.

Estructura de un Fichero

De la definición vista de fichero, se deduce que existen diferentes tipos de ficheros en función de:

- La información contenida
- El método de organización de la información

Una primera clasificación de los ficheros se puede hacer según el método usado para codificar la información:

- **Ficheros de texto**: Se guarda la información en caracteres, tal y como se mostraría en pantalla.
- **Ficheros binarios**: Se guarda la información en binario, tal y como está en memoria.

Otra clasificación de los ficheros es según la forma que tiene su estructura. Las formas más usuales son:

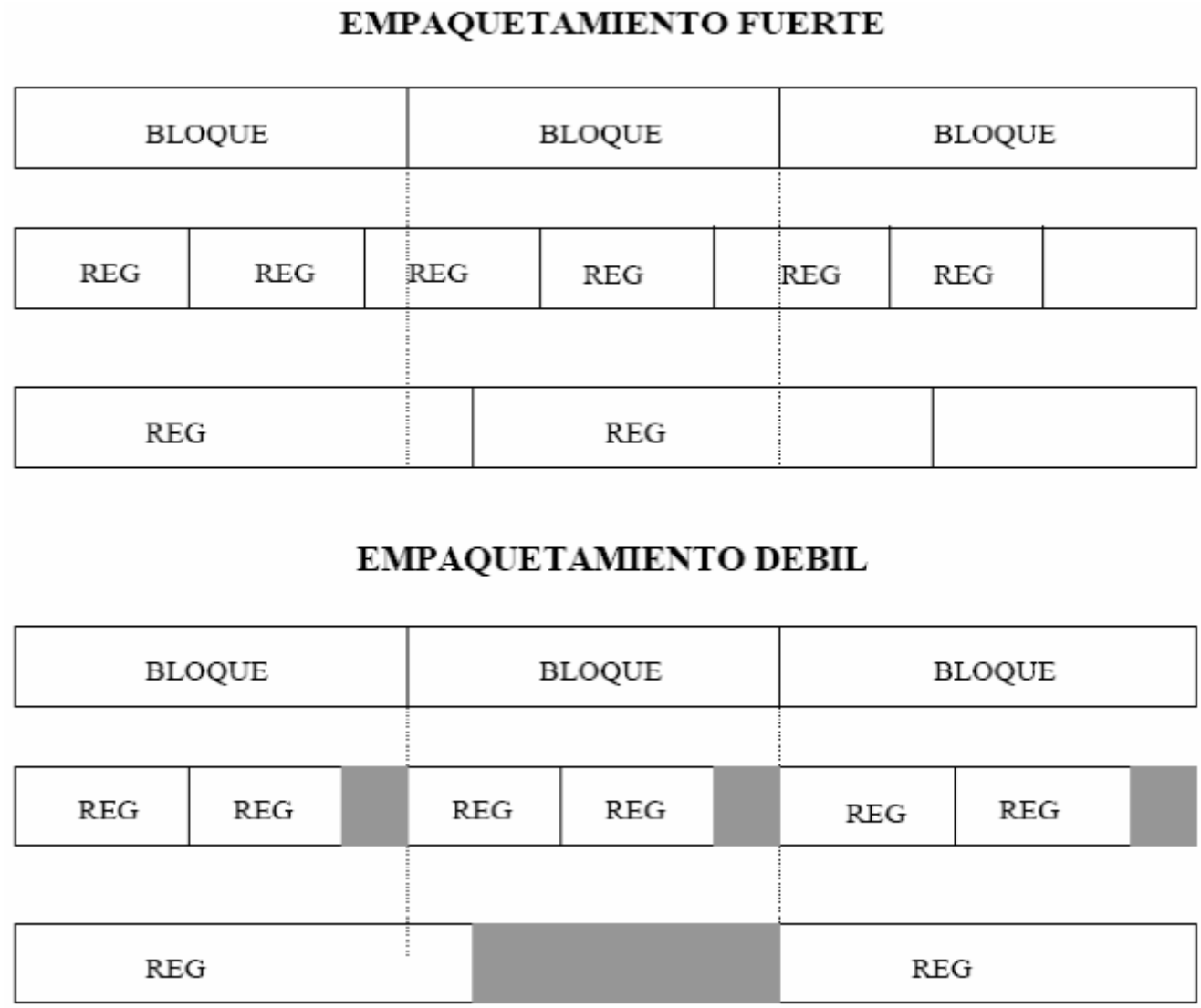
- Organizar un fichero como **una secuencia de bytes**. De esta forma, el sistema operativo no conoce el significado del contenido de los ficheros, lo que simplifica la gestión de los mismos. Serán los programas de aplicación los que deberán de conocer la estructura de los ficheros que utilizan. Este enfoque es el empleado por MS-DOS y UNIX.
- Un esquema más estructurado es considerar un fichero como **una secuencia de registros de longitud fija**, cada uno de los cuales presenta una estructura determinada. La idea es que las operaciones de lectura devuelvan un registro y las escrituras modifiquen o añadan un registro. El SO CP/M usa registros de 128 bytes.

- Una tercera forma es organizar el fichero en forma de **árbol de registros**, que no tienen por qué tener la misma longitud. Cada registro tiene un campo clave por el que está ordenado el árbol de forma que las operaciones de búsqueda por clave se realizan rápidamente. Este esquema se emplea en grandes computadores (mainframes) orientados al proceso de grandes cantidades de información.

Conceptos básicos sobre Ficheros

Veamos ahora una serie de conceptos básicos:

- **Registro lógico**: Un registro es una colección de información relativa a una entidad particular. Por tanto, el registro va a contener a todos aquellos campos lógicamente relacionados, referentes a una determinada entidad, y que pueden ser tratados globalmente por un programa. Por ejemplo la información de un determinado alumno, que contiene los campos DNI, nombre, apellidos, fecha de nacimiento, etc.
- **Clave de un registro lógico**: Una clave es un campo o conjunto de campos de datos que identifica al registro lógico y lo diferencia del resto de registros lógicos del fichero. Por tanto, esta clave debe ser distinta para cada registro.
- **Registro activo**: El registro lógico que va a procesarse en la siguiente operación del fichero.
- **Apuntador**: Marca interna que siempre apunta al registro lógico activo. Se incrementa automáticamente cada vez que se procesa un registro (se lee o se escribe).
- **Marca de fin de fichero**: Una marca situada al final de cada fichero, para no acceder más allá del último registro lógico existente, ya que el tamaño del fichero no está limitado y no se conoce a priori. Existe una función lógica, **eof (end of file)**, que toma el valor verdadero cuando llegamos al final del fichero y falso en caso contrario.
- **Factor de bloqueo**: Factor de bloqueo es el número de registros lógicos que puede contener un registro físico.
- **Registro físico o bloque**: Un registro físico o bloque es la cantidad más pequeña de datos que pueden transferirse en una operación de E/S entre la memoria principal del ordenador y los dispositivos periféricos o viceversa. El tamaño del bloque o registro físico dependerá de las características del ordenador. En la mayoría de los casos el tamaño del bloque suele ser mayor que el de registro lógico. La adaptación consiste en empaquetar en cada bloque tantos registros lógicos como se pueda. El empaquetamiento puede ser de tipo *fuerte* o *débil*, según que se permita o no aprovechar el sobrante de un bloque, situando registros a caballo entre dos bloques contiguos. La siguiente figura ilustra ambas formas de empaquetamiento.



Una vez visto lo que es un registro lógico y teniendo ya en mente la idea de fichero, podemos dar una definición más precisa de lo que es un archivo o fichero.

Un fichero es una colección de registros lógicos relacionados entre sí con aspectos en común y organizados para un propósito específico. Los datos en los archivos deben estar organizados de tal forma que puedan ser recuperados fácilmente, actualizados o borrados y almacenados en el archivo con todos los cambios realizados.

Dese un punto de vista puramente estructural:

Un fichero es una estructura de datos compuesta que agrupa una secuencia de cero o más tuplas, denominadas registros, y que a su vez se pueden componer de otras estructuras de datos a las que se les suele llamar campos.

Operaciones sobre Ficheros

Una vez visto lo que es un fichero y los principales conceptos al hablar de ellos, pasemos ahora a estudiarlos desde un punto de vista operativo. Básicamente se trata de responder: ¿qué operaciones se pueden realizar sobre un fichero? La respuesta es:

- **Creación**: Para poder realizar cualquier operación sobre un fichero es necesario que haya sido creado previamente, almacenando sobre el soporte seleccionado la información requerida para su posterior tratamiento, como por ejemplo el nombre del dispositivo, el nombre del fichero, etc. Con anterioridad a la creación de un archivo se requiere diseñar la estructura del mismo mediante los campos del registro, longitud y tipo de los mismos.

- **Apertura**: Para poder trabajar con la información almacenada en un fichero, éste debe estar abierto, permitiendo así el acceso a los datos, dando la posibilidad de realizar sobre ellos las operaciones de lectura y escritura necesarias.
- **Cierre**: Una vez finalizadas las operaciones efectuadas sobre el fichero, éste debe permanecer cerrado para limitar el acceso a los datos y evitar así un posible deterioro o pérdida de información. Para cerrar un fichero previamente debe estar abierto.
- **Actualización**: Esta operación permite la puesta al día de los datos del fichero mediante la escritura de nuevos registros (alta) y la eliminación (baja) o modificación de los ya existentes. La actualización puede afectar a parte o la totalidad de los registros del fichero. Cuando se escribe un nuevo registro en el fichero se debe comprobar que no existe previamente. La baja de un registro puede ser lógica o física.
    - Una *baja lógica* supone el no borrado del registro en el archivo. Esta baja lógica se manifiesta en un determinado campo del registro con una bandera, indicador o “flag”, o bien con la escritura o rellenado de espacios en blanco en el registro específico.
    - Una *baja física* implica el borrado y desaparición del registro, de modo que se crea un nuevo archivo que no incluye al registro dado de baja.
- **Consulta**: Tiene como fin visualizar la información contenida en el fichero, bien de un modo completo, bien de modo parcial.
- **Borrado o destrucción**: Es la operación inversa a la creación de un fichero. Consiste en la supresión de un fichero del soporte o dispositivo de almacenamiento. El espacio utilizado por el archivo borrado puede ser utilizado por otros archivos. Para borrar un fichero tiene que estar cerrado.
- **Ordenación o clasificación**: Consiste en lograr una nueva disposición sobre el soporte de los registros de un archivo, con una secuencia de ubicación determinada por el valor de uno o varios campos.
- **Compactación o empaquetamiento**: Esta operación permite la reorganización de los registros de un fichero eliminando los huecos libres intermedios existentes entre ellos normalmente ocasionados por la eliminación de registros.

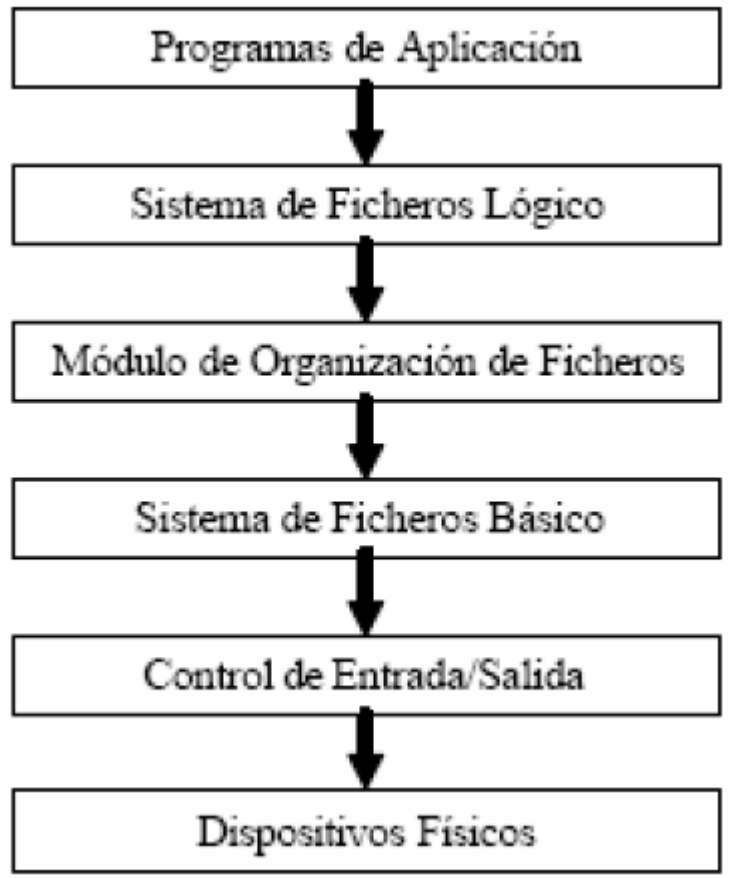
Organización de un Sistema de Ficheros

Los discos magnéticos son la base sobre la que se sustentan los sistemas de ficheros. Para mejorar la eficiencia, la transferencia de información entre memoria y los discos se realiza en unidades denominadas bloques. Cada bloque está formado por uno o varios sectores de disco. El tamaño de sector de un disco suele ser de 512 bytes.

El diseño de un sistema de ficheros plantea dos problemas diferentes:

- Definir cómo el sistema de ficheros aparece al usuario.
- Diseñar los algoritmos y estructuras de datos necesarias para implementar este sistema de ficheros lógico en los dispositivos físicos de almacenamiento secundario.

Un sistema de ficheros se puede estructurar en diferentes capas o niveles, se puede ver en la siguiente figura:



El nivel de control de E/S se compone de los manejadores de dispositivo (*device drivers*) y los manejadores de interrupciones (*interrupt handlers*), que son necesarios para transmitir la información entre la memoria y los discos. Los manejadores de dispositivos reciben instrucciones de bajo nivel, del tipo “escribir o leer el bloque nº x”, y generan el conjunto de instrucciones dependientes del hardware que son enviadas al controlador de disco.

El sistema de ficheros básico transmite las instrucciones de bajo nivel al manejador de dispositivo adecuado para leer y escribir bloques físicos en disco. Cada bloque físico se identifica por su dirección numérica en el disco, que viene dado por el dispositivo, cilindro, superficie y sector.

El módulo de organización de ficheros tiene conocimiento sobre los ficheros, los bloques lógicos que lo componen y los bloques físicos. Mediante el tipo de esquema de asignación de bloques y la localización del fichero, el módulo de organización de ficheros traslada las direcciones de disco lógicas en direcciones de disco físicas. Cada bloque de disco lógico tiene un número (de 0 a N) que no suele coincidir con la dirección de los bloques físicos, por lo que es necesario un mecanismo de traducción. Este módulo también incluye el gestor de espacio libre, que controla los bloques libres para que puedan ser usados posteriormente.

Por último,el sistema de ficheros lógicos proporciona la estructura de directorio conocida por los programas de usuario. También es responsable de proporcionar seguridad y protección al sistema de ficheros.

Por otra parte, para crear un nuevo fichero, un programa de aplicación realiza una llamada al sistema sobre el sistema de ficheros lógico. Este lee el directorio correspondiente en memoria, le añade una nueva entrada y lo escribe en disco. En este proceso, el sistema de ficheros lógico ha solicitado al módulo de organización de ficheros la dirección física del directorio, que se envía al sistema de ficheros básico y al control de E/S. Cuando el directorio ha sido actualizado, el sistema de ficheros lógico lo puede usar para realizar operaciones de E/S.

Para optimizar los accesos, el SO tiene en memoria una tabla de ficheros abiertos que contiene información sobre todos los ficheros abiertos en un momento dado, como nombre, permisos, atributos, direcciones de disco, etc. La primera referencia a un fichero suele ser una operación de apertura, que hace que se realice una búsqueda en la estructura de directorio y se localice la entrada para el fichero que se quiere abrir. Esta entrada se copia en la tabla de ficheros abiertos y el índice de dicha entrada (denominado descriptor de fichero) se devuelve al programa de aplicación, que lo usa para realizar las operaciones sobre el fichero. De esta forma, todas las operaciones relacionadas con el entrada de directorio del fichero se realizan en la tabla de ficheros abiertos en memoria. Cuando un fichero se cierra, la entrada modificada se copia a disco.

Organización y Acceso a Ficheros

Se entiende por **Organización de un fichero** a la manera en la que los datos son estructurados y almacenados internamente en el fichero y sobre el soporte de almacenamiento. El tipo de organización de un fichero se establece durante la fase de creación del mismo. Los requisitos que determinan la organización de un fichero son del tipo: tamaño del fichero, frecuencia de utilización o uso, etc.

La organización de un fichero es muy dependiente del soporte físico en que se almacene. Hay dos tipos de soportes:

- Soportes **Secuenciales**: Los registros están dispuestos físicamente uno a continuación de otro. Para acceder a un determinado registro se necesita pasar por todos los anteriores a él.
- Soportes **Direccionables**: Permiten localizar a un registro directamente por su información (clave) sin tener que pasar por todos los anteriores.

Los tipos de organizaciones de ficheros fundamentales son:

- Organización **Secuencial**.
- Organización **Directa** o **Aleatoria**.
- Organización **Indexada**.

En cuanto al acceso, se entiende por tal al procedimiento necesario que debemos seguir para situarnos sobre un registro concreto con la intención de realizar una operación sobre él. Según las características del soporte empleado y la organización se consideran dos tipos de acceso:

- El **acceso secuencial** implica el acceso a un archivo según el orden de almacenamiento de sus registros, uno tras otro. Se puede dar en dispositivos secuenciales y direccionables.
- El **acceso directo** implica el acceso a un registro determinado, sin que ello implique la consulta de los registros precedentes. Obviamente, sólo puede darse en soportes direccionables.

Organización Secuencial

Son aquellos ficheros caracterizados porque los registros se escriben o graban sobre el soporte de almacenamiento en posiciones de memoria físicamente contiguas, en la misma secuencia u orden en que han sido introducidos, sin dejar huecos o espacios libres entre ellos.

Todos los dispositivos de memoria auxiliar soportan la organización secuencial. El *acceso a los datos* almacenados en estos ficheros siempre es *secuencial* independientemente del soporte utilizado. Los registros organizados secuencialmente tienen un registro especial, el último, que tiene una marca de fin de archivo.

Sus ventajas son:

- Rapidez en el acceso a un bloque de registros que se encuentran almacenados en posiciones de memoria físicamente contiguas.
- No deja espacios vacíos entre registro y registro, optimizado al máximo la memoria ocupada.

Sus inconvenientes son:

- El acceso a registros individuales es muy lento.
- Se tiene que procesar todo el fichero para operaciones de inserción y borrado de registros.

Organización Directa o Aleatoria

Estos ficheros se caracterizan porque los registros se sitúan en el fichero y se accede a ellos a través de una clave, que indica la posición del registro dentro del fichero y la posición de memoria donde está ubicado.

Estos ficheros se almacenan en *soportes direccionables*. Además, los registros han de tener un identificativo o clave, el cual indica la posición de cada registro en el fichero.

Como principales ventajas genéricas de este tipo de organización, tenemos que:

- Cada posición solamente puede ser ocupada por un registro, pues no podemos tener en el fichero más de un registro con el mismo valor de clave.
- El acceso a cualquier registro se hace de una forma directa e inmediata mediante su clave.
- La actualización de un registro es inmediata, sin que se deban utilizar archivos auxiliares para copia.
- Se puede utilizar el acceso secuencial, aunque suponga generalmente una pérdida de tiempo.

Dentro de la organización directa, según el algoritmo utilizado en la gestión de la clave, se pueden distinguir entre:

- Organización directa con **Clave Directa**: La dirección de almacenamiento del registro está indicado por la propia clave.
    - Sus ventajas son:
        - Cada posición solamente puede ser ocupada por un registro, pues no podemos tener en el fichero más de un registro con el mismo valor de clave.
        - Es muy rápido el acceso a los registros individuales.
    - Sus inconvenientes son:
        - Deja gran cantidad de huecos dentro del fichero, con el consecuente desaprovechamiento del soporte de almacenamiento. Esto es debido a que este sistema precisa que el soporte donde se almacena la información tenga una mínima unidad de asignación (denominado *cluster*) a la cual acceder directamente siguiendo unas coordenadas de localización. Estas unidades no pueden ser usadas para almacenar información de distintos ficheros. Si los cluster en que divide el disco son grandes, y los ficheros a almacenar pequeños, habrá muchos cluster que se quedarán a medio llenar, con el consecuente desaprovechamiento de espacio.
        - Una consulta total del fichero puede suponer un gran inconveniente, pues hay que analizar todas las posiciones de memoria, aunque algunas posiciones estén vacías. Para comprender este hecho hagamos el siguiente símil. Supongamos que tenemos un libro con todos sus capítulos correctamente ordenados pero sin índice que nos identifique la página de

inicio de cada capítulo, para leer el libro entero lo único que hay que hacer es comenzar por la primera página, sin embargo, si queremos leer un capítulo en concreto, tendríamos también que empezar desde el principio hasta encontrarlo. Esto sería el análogo a un fichero secuencial. Supongamos ahora un libro con todos los capítulos desordenados, pero con un índice al comienzo del libro que nos indica la página de comienzo de cada capítulo; es decir, el análogo al de un fichero de acceso directo. Leer cada capítulo por separado es ahora muy fácil, basta con buscar el comienzo en el índice, sin embargo, si necesitamos leer el libro entero, de principio a fin, necesitamos constantemente mirar el índice para ir viendo la secuencia de los capítulos, lo cual es muy ineficiente (por no decir pesado). Esto sería el análogo a leer completamente un fichero de acceso directo.

- Organización directa con **Clave Indirecta**: La dirección de almacenamiento se obtiene a partir de la clave, después de realizar algún tipo de transformación. Este tipo de transformación se denomina algoritmo *Hashing* y suele ser de tipo matemático.
    - En este tipo de algoritmo se pueden dar dos situaciones no deseadas (denominadas *colisiones*) que son:
        - Hay direcciones que no corresponden a ninguna clave y, por tanto, zonas de disco sin utilizar.
        - Hay direcciones que corresponden a más de una clave. En este caso se dice que las claves son sinónimas para esa transformación.
    - Hay dos formas de resolver el problema de los sinónimos o colisiones:
        - Buscar secuencialmente en el archivo hasta encontrar una posición libre donde escribir el registro o aplicando a la dirección obtenida un segundo método de direccionamiento. Estos procedimientos son lentos y degradan el archivo.
        - Reservar una zona de desbordamiento o de sinónimos en donde se escribirán los registros que no se pueden escribir en la dirección que le corresponde según la transformación.

Organización Indexada

Al fichero le acompaña un fichero de índice que tiene la función de permitir el acceso directo a los registros del fichero de datos. El índice se puede organizar de diversas formas, las más típicas son:

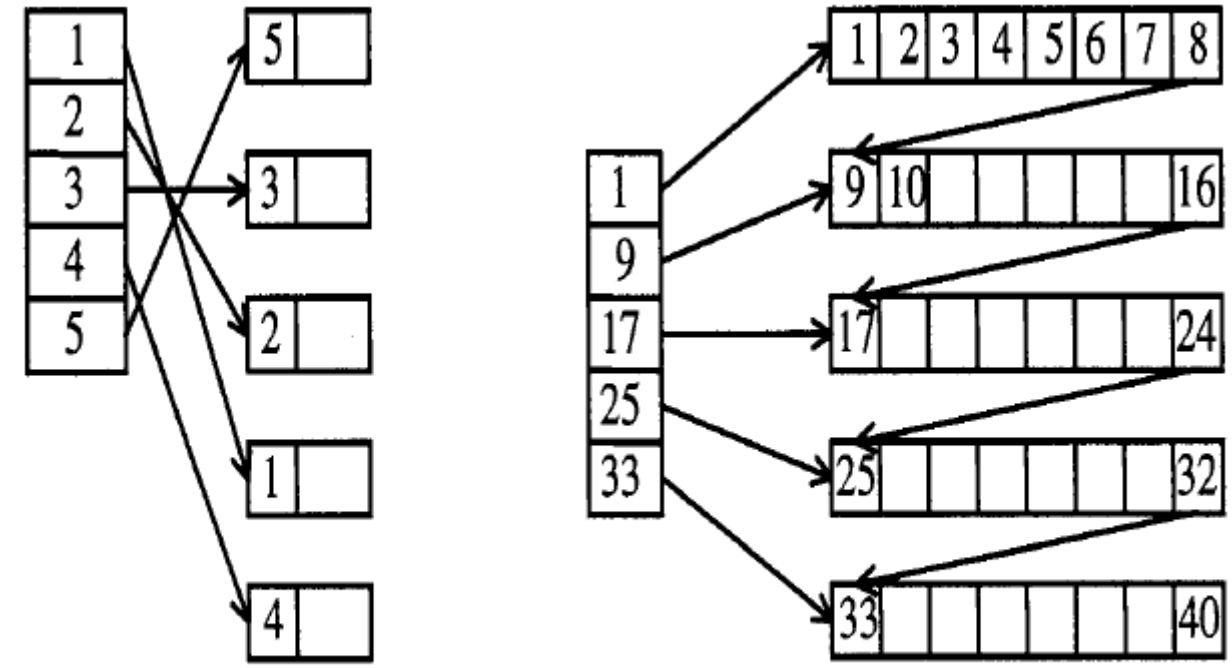
- **Secuencial**
- **Multinivel**
- **Árbol**

A través del índice podremos procesar un fichero de forma secuencial o de forma directa según la clave de indexación, y esto independientemente de como esté organizado el fichero por sí mismo.

El índice debe estar organizado en función de alguno de los campos de los registros de datos. Se pueden tener tantos índices como se quiera variando la clave (o campo) que se emplee. El índice está formado por registros (entradas) que contienen:

- Clave de organización.
- Puntero(s) al fichero de datos, en concreto al registro que corresponda.

Los índices se pueden clasificar en dos tipos, según cada entrada señale a la dirección de un registro del fichero de datos (*índice total o denso*), o bien apunte a un grupo de registros del fichero de datos que debe estar ordenado (*índice escaso o no denso*). En el caso de índices totales, el fichero puede estar desordenado. Véase la siguiente figura como ejemplo:



Índice Total o Denso

Índice Escaso o No Denso

Con el segundo tipo se podría procesar directamente el fichero de datos de forma secuencial. Los índices totales o densos no suelen utilizarse de forma simple, sino combinados con índices escasos o más cortos, de esta manera pueden almacenarse en memoria principal obteniendo así una mayor rapidez de acceso.

Implantación de Ficheros

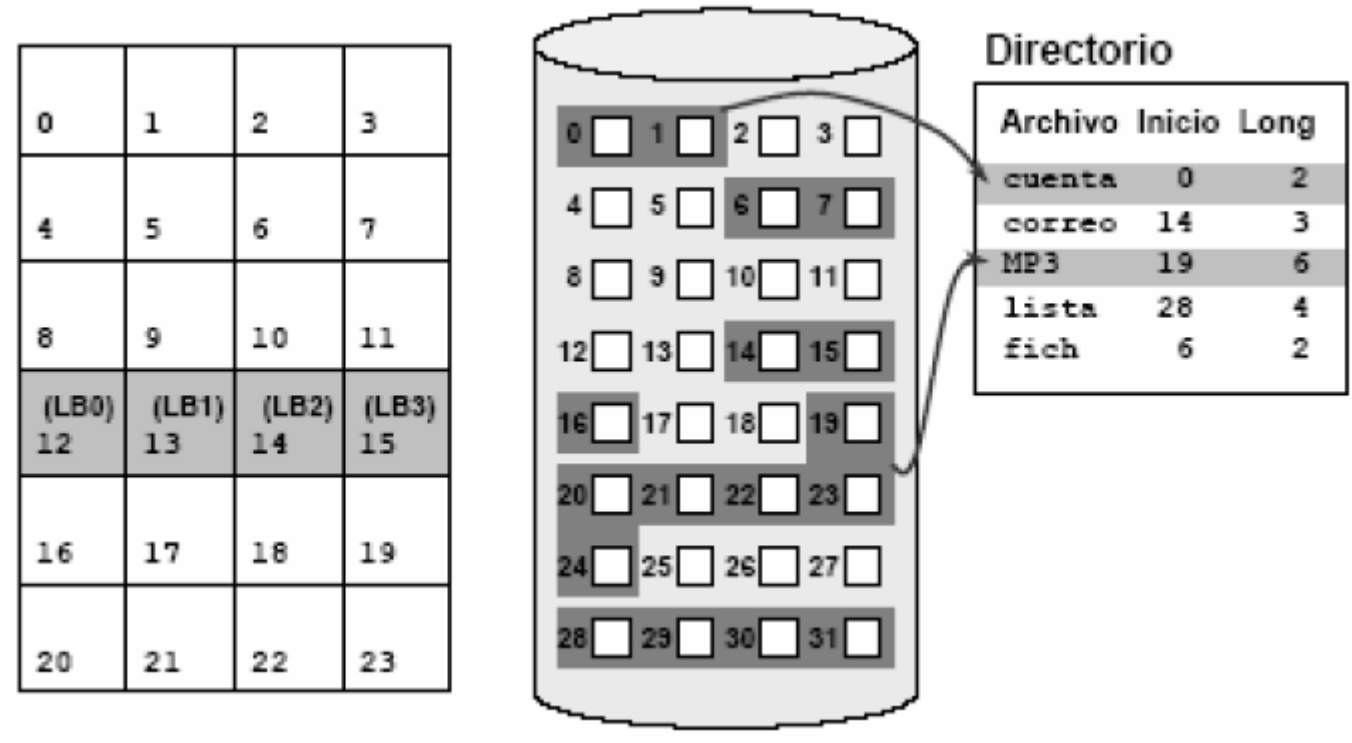
El aspecto básico de la implantación de ficheros consiste en determinar qué bloques de disco están asociados a cada fichero. El problema que se plantea es cómo asignar el espacio libre en disco a los ficheros de forma que ese espacio sea usado de forma eficiente y los ficheros puedan ser accedidos rápidamente. Hay que tener en cuenta que el tamaño de los ficheros es variable, por lo que habrá que diseñar algún mecanismo de almacenamiento dinámico tanto para los ficheros como para el espacio libre.

Existen tres métodos básicos de asignación:

- Asignación **Contigua**.
- Asignación **Enlazada**.
- Asignación **Indexada**.

Asignación Contigua

El esquema de asignación más simple consiste en almacenar cada fichero como una secuencia adyacente de bloques en disco. La asignación contigua de un fichero se define por la dirección del primer bloque y el tamaño del fichero, es decir, simplemente responder a las preguntas ¿dónde empezar? y ¿hasta cuando seguir?.



Las ventajas de este método son:

- Fácil implementación, ya que únicamente hay que conocer la dirección en disco del primer bloque del fichero y el número de bloques que ocupa.
- Eficiencia, ya que la lectura de un fichero se puede realizar en una sola operación. El acceso a un fichero almacenado de forma contigua es sencillo y rápido, siempre y cuando pretendamos leer todo el fichero y no tengamos que hacer búsquedas de parte de su contenido. En este tipo de operaciones se pierde toda la ventaja que proporciona esta forma de asignación. Para este tipo de operaciones veremos que es mucho más ventajosa la asignación.

Para el acceso secuencial, el sistema de ficheros no tiene más que recordar la dirección del último bloque referenciado. El acceso directo del bloque *i* de un fichero que comienza en el bloque *b* se realiza, simplemente, accediendo al bloque *b*+*i*.

Un inconveniente de este esquema de asignación es que no se puede implantar a no ser que se conozca el tamaño de los ficheros en su momento de creación. Si esta información no está disponible, el SO no sabe cuánto espacio en disco debe reservar. Otra desventaja es la fragmentación externa que se produce en el disco, que se origina debido a que entre los ficheros quedan porciones de disco libre que no tienen el tamaño suficiente para ser asignadas a un nuevo fichero. La gravedad de este problema depende del espacio de almacenamiento disponible y del tamaño medio de los ficheros. La compactación de disco es una solución, pero es costos y de poder hacerse sería cuando el sistema estuviese parado.

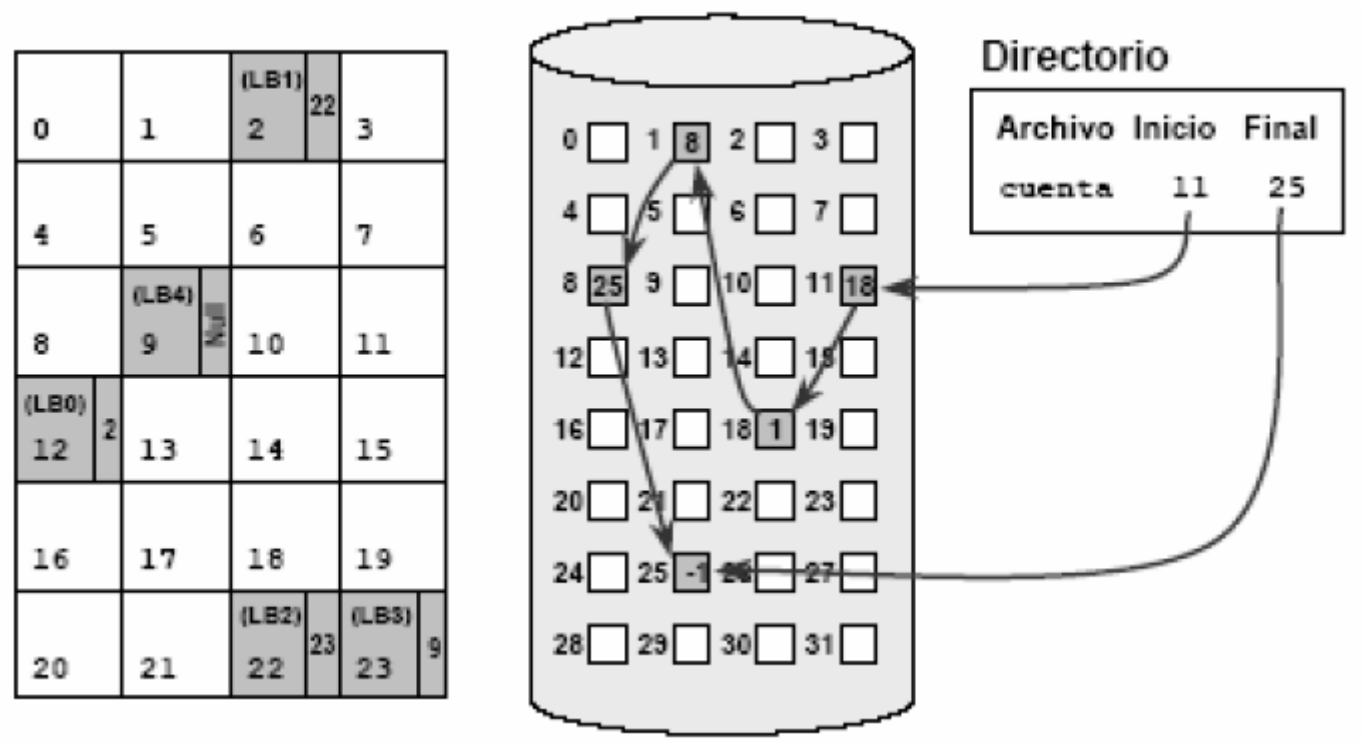
El problema de conocer el tamaño de los ficheros en tiempo de creación se resuelve haciendo que los ficheros sean inmutables. Esto significa que cuando un fichero se crea no se puede modificar. Una modificación implica borrar el fichero original y crear uno nuevo. Por otro lado, los ficheros se han de leer de una sola vez en memoria, por lo que la eficiencia es muy alta, pero a cambio se exigen unos requisitos mínimos de memoria muy elevados. Por último, la fragmentación de disco se soluciona teniendo discos de gran capacidad de almacenamiento.

Asignación Enlazada

Otro esquema consiste en mantener una lista enlazada con los bloques que pertenecen a un fichero, de forma que una palabra del bloque sería un índice al bloque siguiente. Con este método se elimina el problema de la fragmentación, y, al igual que en la asignación contigua, cada entrada de directorio únicamente contiene la dirección del primer bloque del fichero. Los fichero pueden crecer de forma dinámica mientras que exista espacio libre en disco y no es necesario compactar los discos. Un inconveniente es que, mientras que el acceso secuencial es fácil de realizar, el acceso aleatorio es muy lento.

Otro inconveniente viene dado por el espacio requerido por los punteros. Si un puntero requiere 4 bytes y los bloques son de 512 bytes, significa que: (4 / 512) = 0,0078 –> se pierde un 0,78% (casi un 1%) de espacio útil por bloque. Una solución a este problema consiste en no asignar bloques individuales, sino conjuntos de bloques denominados *clusters*. De esta forma el porcentaje de espacio perdido por los punteros se reduce. Además, se mejora el rendimiento de los discos porque se lee más información en una misma operación de lectura, y se reduce la lista de bloques libres. El coste de este método es que se produce un incremento de la fragmentación interna (espacio no ocupado en un bloque o cluster).

Otro problema de este método es la fiabilidad, ya que si un bloque se daña, se puede perder el resto del fichero. Peor aún, sise altera el contenido del puntero, se puede acceder a la lista de bloques libres o a bloques de otros ficheros como si fuesen bloques propios.



Un problema más sutil viene dado por el hecho de que la cantidad de información útil para el usuario que contiene cada bloque no es potencia de dos, ya que el puntero al bloque siguiente, que estará constituido por varias palabras, ocupará parte del bloque, es decir, no se puede utilizar todo el bloque para información útil, siempre habrá que dejar mínimo espacio para el puntero. Esto puede originar una pérdida de eficiencia porque los programas normalmente leen y escriben en bloques cuyo tamaño es potencia de dos. Esta desventaja del esquema de lista encadenada desaparece si el puntero de cada bloque de disco se almacena en una tabla o índice de memoria. Así cada bloque únicamente contiene datos útiles.

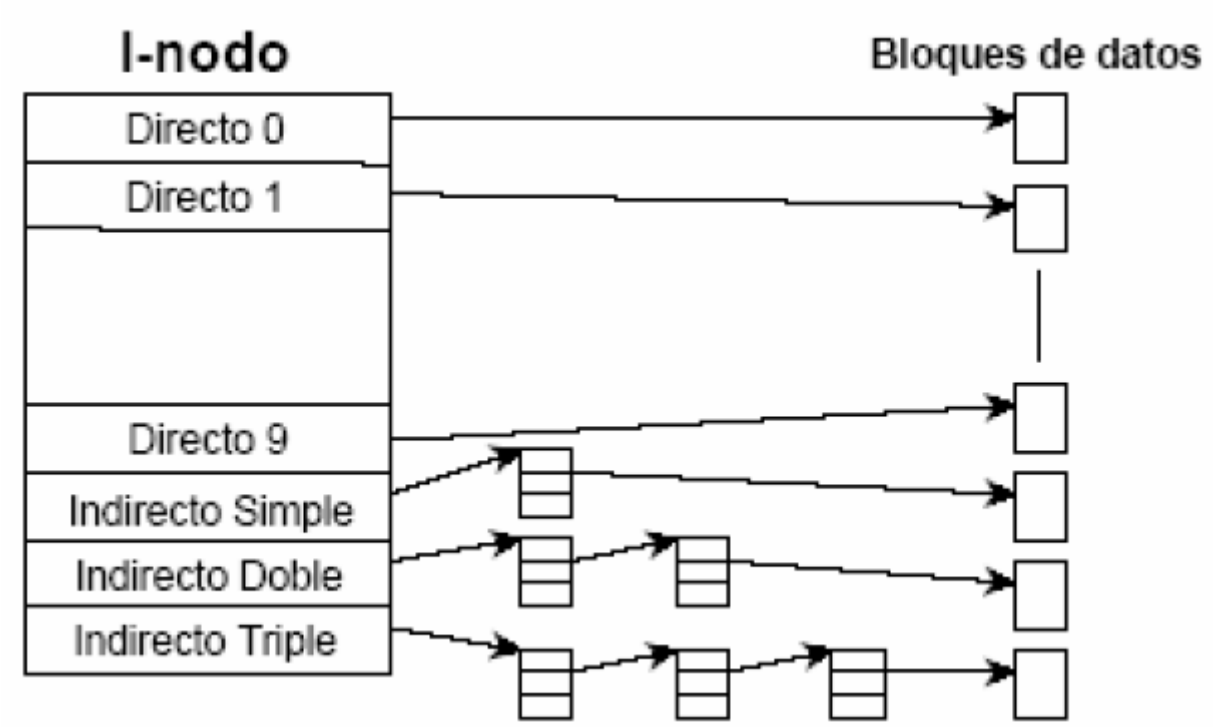
Además, aunque el acceso aleatorio implica seguir una cadena, ésta está toda en memoria, por lo que la búsqueda es mucho más eficiente que en el esquema anterior. De igual modo, es suficiente que cada entrada de directorio contenga únicamente la dirección

del primer bloque para localizar todo el fichero. Este esquema es empleado por el SO MS-DOS. Para que este esquema funcione, la tabla debe permanecer entera en memoria todo el tiempo. Si el disco es grande, el gasto de memoria principal puede ser considerable.

Asignación Indexada

El último método para determinar qué bloques pertenecen a un fichero consiste en asociar a cada fichero un nodo índice (*index-node o i-node*), que incluye información sobre los atributos del fichero y las direcciones de los bloques de disco que pertenecen al mismo.

Las direcciones de los primeros bloques se almacenan en el propio nodo índice, de forma que, para ficheros pequeños, toda la información necesaria para su localización siempre está en memoria cuando son abiertos. Para ficheros de mayor tamaño, una de las direcciones del nodo índice es la dirección de un bloque simple indirecto, que contiene a su vez direcciones de otros bloques del fichero. Si no es suficiente, existen dos direcciones, una para un bloque doble indirecto, que contiene direcciones de bloques simples directos, y otra para un bloque triple indirecto, que contiene direcciones de bloques dobles indirectos.



Una ventaja es que permite el acceso directo a cualquier bloque del fichero. Además, los accesos a un fichero, una vez abierto, no implican accesos adicionales a disco a no ser que sean necesarios bloques indirectos.

Un inconveniente es que añadir o borrar bloques en medio del fichero implica tener que reorganizar los punteros en el bloque de índices, lo cual constituye una operación costosa.

Directorios

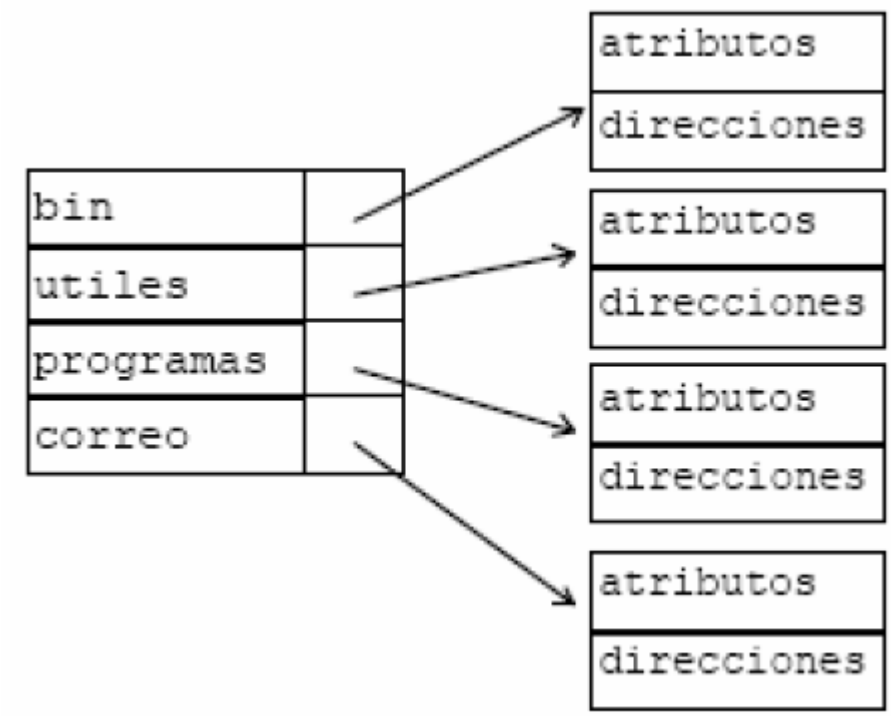
Los sistemas de ficheros pueden contener grandes volúmenes de información, por lo que los SO proporcionan mecanismos para estructurarlos. Esta estructuración se suele realizar a dos niveles.

En primer lugar, el sistema de ficheros se divide en particiones, que se pueden considerar como discos virtuales. Un disco puede contener una o varias particiones, y una partición puede estar repartida. En segundo lugar, cada partición contiene una tabla de contenidos de la información que contiene. Esta tabla se denomina directorio, y su función principal

es hacer corresponder un nombre de un fichero con una entrada en la tabla. Cada entrada contiene, como mínimo, el nombre del fichero.

A continuación se pueden almacenar los atributos y direcciones del fichero en disco, o un puntero a otra estructura de datos que contiene dicha información. Esta estructura de datos se suele conocer con el nombre **nodo índice**.

Mostramos los atributos y direcciones a una estructura de datos:



Cuando se abre un fichero el SO busca en el directorio correspondiente hasta que encuentra el nombre del fichero, toma la información que necesita y la introduce en una tabla residente en la memoria principal. Las siguientes referencias al fichero utilizarán la información que ya está en memoria.

Implantación de Directorios

Para acceder a un fichero, ya sea para lectura o escritura, previamente hay que abrir el fichero. Esta operación implica que el SO, a partir de la ruta de acceso, debe localizar la entrada de directorio correspondiente. Esta entrada proporciona la información necesaria para localizar los bloques de disco del fichero. Esta información, según el tipo de sistema, puede ser la dirección en disco del fichero completo (asignación contigua), el número del primer bloque (los dos métodos basados en listas encadenadas) o el número de nodo índice. En todo caso, la principal función del sistema de directorios es asociar el nombre en ASCII de un fichero con la información precisa para localizar los datos. Los atributos de los ficheros se pueden almacenar en la misma entrada de directorio. Si usan nodos índice, es posible almacenar los atributos en el nodo índice.

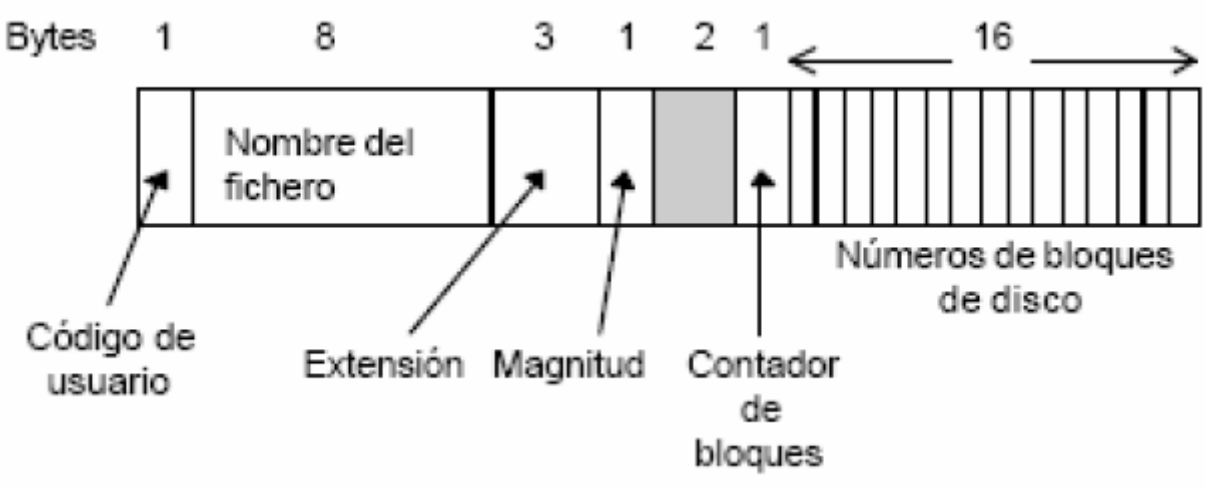
A continuación se exponen brevemente la implantación de directorios en varios SO: CP/M, MS-DOS y UNIX. No vamos a mencionar nada de las características de cada SO, simplemente mencionamos estos SO para ver distintas filosofías de implementar directorios.

Directorios en CP/M

En CP/M solamente existe un directorio, por lo que el sistema de ficheros únicamente tiene que buscar el nombre de un fichero en dicho directorio. Si un fichero ocupa más de 16 bloques se le asignan más entradas de directorio.

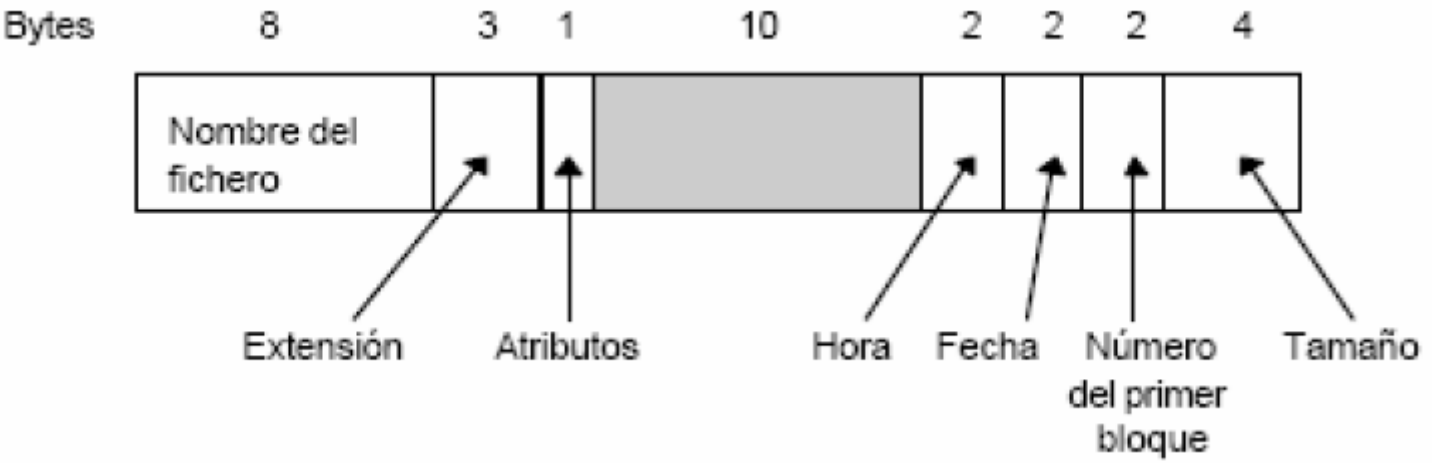
Los campos de una entrada de directorio en CP/M son los siguientes:

- El código de usuario (1 byte) permite conocer el propietario del fichero.
- El nombre y la extensión del fichero (8 y 3 bytes, respectivamente).
- Si un fichero ocupa más de 16 bloques, el campo magnitud (1 byte) indica el orden que ocupa esa entrada.
- El contador de bloques (1 byte) indica cuántos bloques están en uso de los 16 posibles.
- Los últimos 16 campos contienen las direcciones de bloques de disco del fichero. Como el último bloque puede no estar lleno, es imposible conocer con exactitud el tamaño en bytes de un fichero.



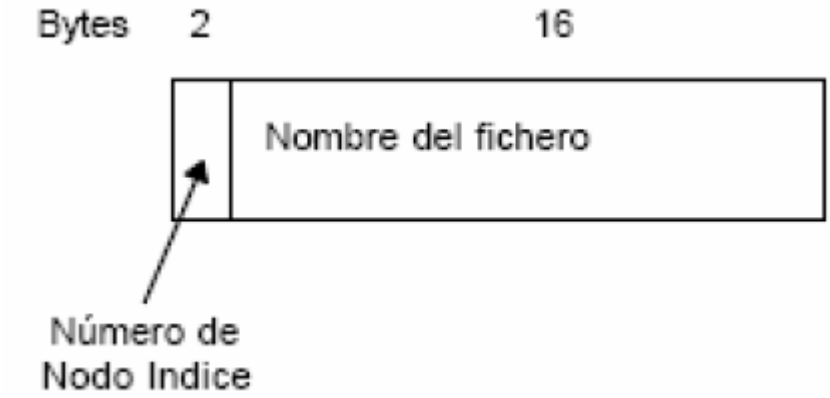
Directorios en MS-DOS

MS-DOS tiene un sistema de ficheros jerárquico. Cada entrada de directorio tiene 32 bytes, y contiene, entre otros datos, el nombre del fichero, la extensión, los atributos y el número del primer bloque en disco del fichero. Este número es un índice a una tabla de asignación denominada FAT (*File Allocation Table*). Un directorio puede contener otros directorios, lo que origina la estructura jerárquica del sistema de ficheros.



Directorios en UNIX

La estructura de una entrada de directorio en un sistema UNIX tradicional es muy simple, y contiene únicamente el nombre del fichero y el número de su nodo índice. Toda la información sobre el fichero reside en el propio nodo índice.



Formatos de Información y Ficheros

Independientemente de la plataforma con que se trabaje, la información obtenida se genera con un ordenador y se suele almacenar en un fichero, con la intención de recuperarla más tarde cuando sea necesaria, o compartirla con los demás a través de algún medio de transmisión de datos. En consecuencia, es conveniente conocer los formatos de archivo más indicados para almacenar los distintos tipos de información que deben contener.

En este apartado vamos a estudiar los ficheros desde un punto de vista operativo, es decir, según el uso o función que se le da al fichero. Podría decirse que desde el punto de vista del usuario, al que le interesa guardar datos en un archivo y tenerlo localizado de alguna forma en su ordenador independientemente de cómo esté organizado el fichero o cual sea su estructura.

Formatos de Información

La mayoría de aplicaciones suelen guardar la información que producen en formatos de fichero propios, de modo que podemos editarlos posteriormente con la garantía que se respetarán todas las peculiaridades de los datos y el nivel de edición que poseían en el momento de guardarlos, ahora, cuando compartimos información debemos ser muy cuidadosos con la elección del tipo de fichero ya que no debemos asumir que todo el mundo posee nuestras mismas herramientas ni nuestro mismo sistema.

No se trata de analizar en profundidad las características de todos los tipos de archivo, sino indicar algunas referencias generales sobre los más usados que nos ayuden a decidir el formato adecuado en cada ocasión.

Ficheros con Información de Texto

Si queremos almacenar o compartir un fichero de texto tenemos dos formatos básicos independientes de la plataforma, es decir, son legibles con un editor de texto sobre cualquier SO:

- **TXT**, para ficheros de texto plano.
- **RTF**, Rich Text Format (Formato de texto enriquecido) cuando sea necesario incluir en el texto algunos elementos de realce como cursivas o negrita.

Ficheros con Información de Imagen

Hay gran variedad de formatos de archivos gráficos, la mayoría compatibles con cualquier plataforma. Entre los más habituales se encuentran:

- **JPG** para imágenes de tono continuo en mapa de bits. Es un formato comprimido pues prescinde de los datos de color de la imagen que no están en el espectro visible.

- **GIF** usado especialmente con animaciones y gráficos con regiones transparentes. Tiene algunos problemas legales con los términos de su licencia.
- **PNG** tiene similares características al GIF aunque se trata de un formato más evolucionado y de mayor calidad, con muy buenos ratios de compresión y soporte para multitransparencia. Posee una licencia libre.
- **TIFF** se utiliza para almacenar imágenes sin pérdida de calidad, por lo que genera tamaños de archivo mayores que el resto pese a que incorpora un algoritmo de compresión.
- **SVG** para ilustraciones vectoriales.

Ficheros con Información Compuesta

Cuando se trata de compartir documentos que integran texto con imágenes o gráficos, o la composición y aspecto son fundamentales por tratarse de formularios estandarizados o similares, tenemos dos alternativas:

- PS es un documento PostScript o formato de impresión capaz de ser visualizado con alguna aplicación auxiliar e impreso sin problemas, directamente. Mantienen la misma calidad de resolución que el documento original.
- PDF es una versión del anterior, desarrollada por la compañía Adobe que se usa frecuentemente para compartir documentación en Internet.

Ficheros con Información Comprimida

Para aliviar las dificultades de transmitir archivos de gran tamaño a través de las redes o ahorrar espacio en disco, se desarrollaron distintos algoritmos de compresión capaces de reducir la cantidad de memoria ocupada por un fichero. Los formatos más usados son **ZIP**, **RAR**.

Tipos de Fichero según su uso

Se pueden clasificar en:

- Ficheros **Permanentes**: Contienen los datos relevantes para una aplicación. Su vida es larga y no pueden generarse de forma inmediata. Entre estos se puede distinguir entre:
  - Ficheros **maestros**: contienen el estado actual de los datos suceptibles de ser modificados en la aplicación. Ejemplo: el fichero de clientes de un banco.
  - Ficheros **constantes**: contienen datos fijos para la aplicación. Ejemplo: el fichero de códigos postales.
  - Ficheros **históricos**: contienen datos que fueron actuales en tiempos anteriores. Ejemplo: el fichero de clientes que se han dado de baja.
- Ficheros **Temporales**: Contienen datos relevantes para un proceso o programa. Su vida es corta y se utilizan para actualizar los ficheros permanentes.
  - Ficheros de **movimientos**: almacenan resultados de un programa que han de ser utilizados por otro, dentro de la misma tarea. Ejemplo: el fichero de movimientos de una cuenta bancaria.
  - Ficheros de **maniobras**: almacenan datos que un programa no puede conservar en memoria principal por falta de espacio. Ejemplo: editores, compiladores y programas de cálculo numérico.
  - Ficheros de **resultados**: se utilizan para almacenar datos elaborados que van a ser transferidos a un dispositivo de salida. Ejemplo: un fichero de impresión.

Denominación de Ficheros

Los ficheros tienen asignados un nombre a través del cual los usuarios se refieren a ellos. Sin embargo, las reglas de denominación de ficheros difieren dependiendo del SO. Muchos SO dividen el nombre de los ficheros en dos partes separadas por un punto. La

primera parte sería el nombre propiamente dicho. La segunda parte se suele denominar *extensión* y suele aportar información sobre el contenido del fichero.

Las reglas básicas de denominación de ficheros en los SO MS-DOS, UNIX y Windows NT son:

- **MS-DOS**: El nombre de un fichero se compone de un máximo de 8 caracteres, seguidos, opcionalmente, por un punto y una extensión de tres caracteres como máximo. Las mayúsculas y minúsculas son consideradas iguales. Por ejemplo: *archivo12.doc*.
- **UNIX**: El nombre de un fichero se compone de un máximo de 256 caracteres. Se distinguen las mayúsculas de las minúsculas. Un fichero puede tener más de una extensión. Por ejemplo: *image.tar.z*.
- **Windows NT**: El nombre de un fichero se compone de un máximo de 256 caracteres. Las mayúsculas y minúsculas no son distinguibles y los ficheros pueden tener más de una extensión.

En muchos casos, las extensiones son meras convenciones y no relacionadas con el contenido de los ficheros. Por otro lado, muchas aplicaciones requieren que los ficheros que utilizan tengan unas extensiones concretas.

Bibliografía

- Scribd (Ibiza Ales)

Diseño de programas. Diseño estructurado. Análisis de transformación y de transacción. Cohesión y acoplamiento.

Introducción al Diseño Estructurado

Conceptos Generales Sobre el Diseño

Definición: “Diseño es el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso, o sistema, con los suficientes detalles como para permitir su realización física”.

El objetivo del diseñador es producir un modelo de una entidad que se construirá más adelante. El proceso por el cual se desarrolla el modelo combina:

- La intuición y los criterios en base a la experiencia de construir entidades similares.
- Un conjunto de principios y/o heurísticas que guían la forma en la que se desarrolla el modelo.
- Un conjunto de criterios que permiten discernir sobre la calidad.
- Un proceso de iteración que conduce finalmente a una representación del diseño final.

La actividad de Diseño se dedica a asignar porciones de la especificación estructurada (también conocida como modelo esencial) a procesadores adecuados (sean máquinas o humanos) y a labores apropiadas (o tareas, particiones, etc) dentro de cada procesador. Dentro de cada labor, la actividad de diseño se dedica a la creación de una jerarquía apropiada de módulos de programas y de interfaces entre ellos para implantar la

especificación creada en la actividad de análisis. Además, la actividad de diseño se ocupa de la transformación de modelos de datos de entidad/relación en un diseño de base de datos.

¿Qué es Diseño Estructurado?

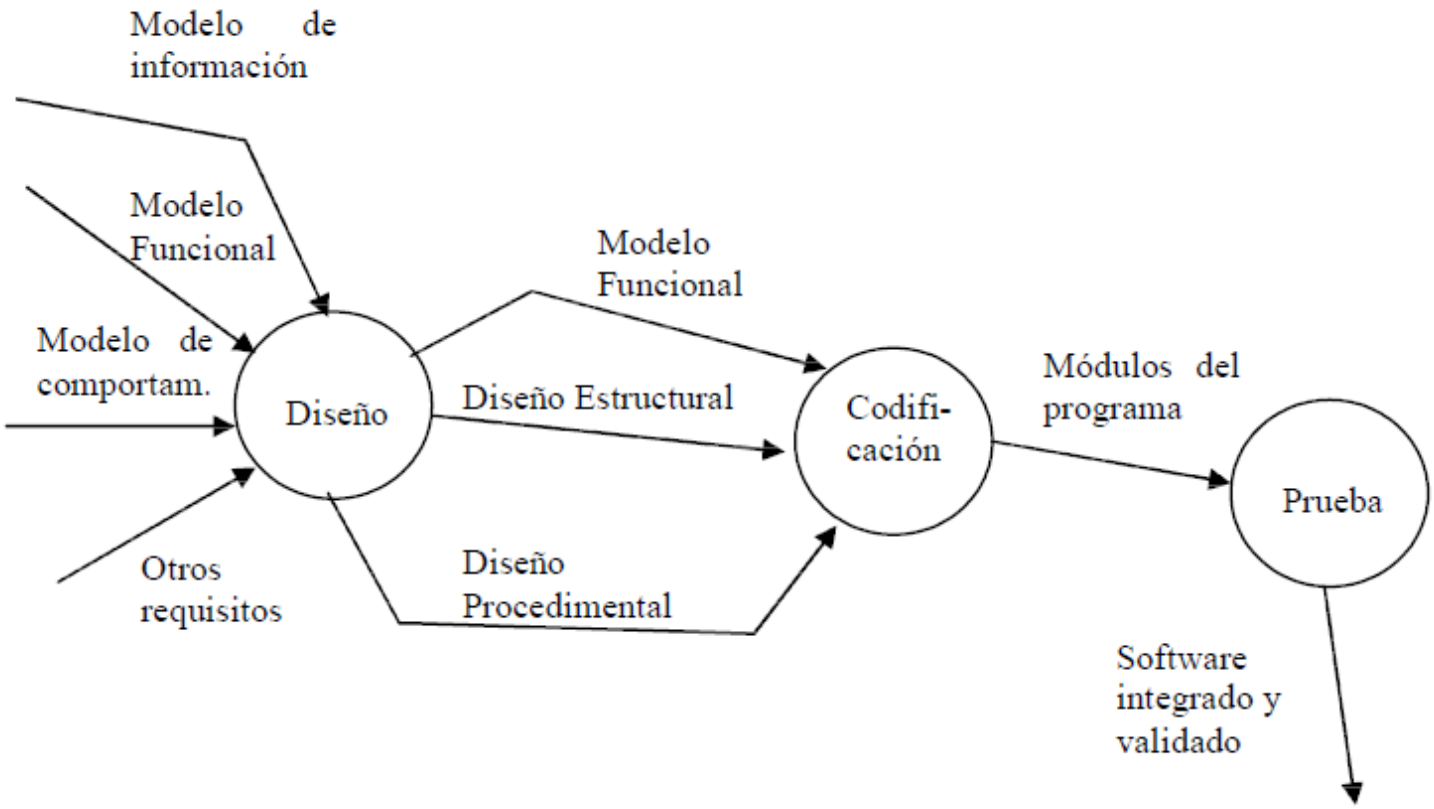
Definición: “Diseño estructurado es el proceso de decidir qué componentes, y la interconexión entre los mismos, para solucionar un problema bien especificado”.

El diseño es una actividad que comienza cuando el analista de sistemas ha producido un conjunto de requerimientos funcionales lógicos para un sistema, y finaliza cuando el diseñador ha especificado los componentes del sistema y las relaciones entre los mismos.

Frecuentemente analista y diseñador son la misma persona, sin embargo es necesario que se realice un cambio de enfoque mental al pasar de una etapa a la otra. *Al abordar la etapa de diseño, la persona debe quitarse el sombrero de analista y colocarse el sombrero de diseñador*.

Una vez que se han establecido los requisitos del software (en el análisis), el diseño del software es la primera de tres actividades técnicas: *diseño*, *codificación* y *prueba*. Cada actividad transforma la información de forma que finalmente se obtiene un software para computadora válido.

En la figura se muestra el flujo de información durante la fase de desarrollo. Los requisitos del sistema, establecidos mediante los *modelos de información*, *funcional* y *de comportamiento*, alimentan el proceso del diseño. Mediante alguna metodología (en nuestro caso, estructurada basada en el flujo de información) se realiza el diseño estructural, procedimiental y de datos.



El *diseño de datos* transforma el modelo del campo de información, creado durante el análisis, en las estructuras de datos que se van a requerir para implementar el software.

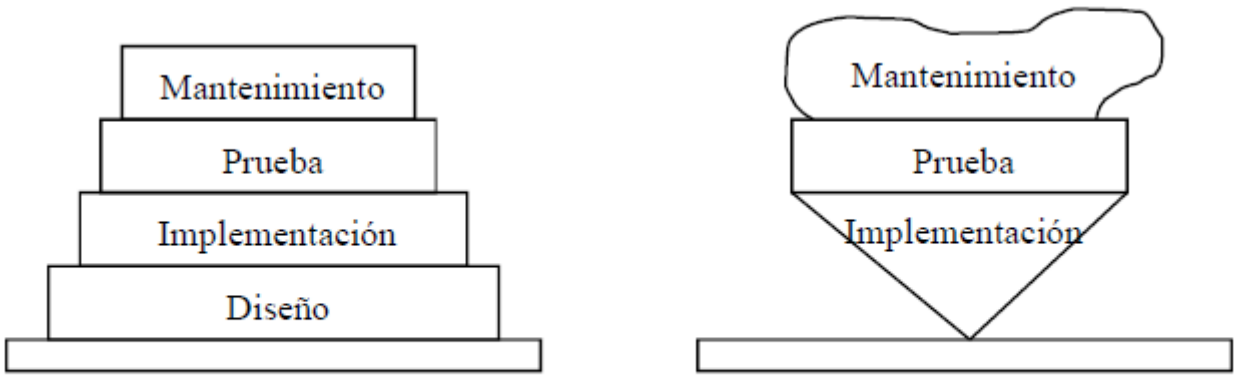
El *diseño estructural* define las relaciones entre los principales elementos estructurales del programa. El objetivo principal del diseño estructural es desarrollar una estructura de programa modular y representar las relaciones de control entre los módulos.

El *diseño procedimental* transforma los elementos estructurales en una descripción procedimental del software. El diseño procedimental se realiza después de que se ha establecido la estructura del programa y de los datos. Define los algoritmos de procesamiento necesarios.

Concluido el diseño se genera el código fuente y para integrar y validar el software, se llevan a cabo pruebas de testeo.

Las fases del diseño, codificación y prueba absorben el 75% o más del coste de la ingeniería del software (excluyendo el mantenimiento). Es aquí donde se toman *decisiones* que afectarán finalmente al éxito de la implementación del programa y, con igual importancia, a la facilidad de mantenimiento que tendrá el software. Estas decisiones se llevan a cabo durante el diseño del software, haciendo que sea un paso *fundamental* de la fase de desarrollo.

La importancia del diseño del software se pueden sentar con una única palabra: *calidad*. El diseño es el proceso en el que se asienta la calidad del desarrollo del software. El diseño produce las representaciones del software de las que puede evaluarse su calidad. El diseño sirve como base para todas las posteriores etapas del desarrollo y de la fase de mantenimiento. Sin diseño nos arriesgamos a construir un sistema inestable, un sistema que falle cuando se realicen pequeños cambios, un sistema que pueda ser difícil de probar, un sistema cuya calidad no pueda ser evaluada hasta más adelante en el proceso de ingeniería de software, cuando quede poco tiempo y se haya gastado ya mucho dinero.



Con Diseño *Sin Diseño*

Objetivos del Diseño Estructurado

El diseño estructurado, tiende a transformar el desarrollo de software de una práctica artesanal a una disciplina de ingeniería.

- Eficiencia
- Mantenibilidad
- Modificabilidad
- Flexibilidad
- Generalidad
- Utilidad

"Diseño" significa planear la forma y método de una solución. Es el proceso que determina las características principales del sistema final, establece los límites en

performance y calidad que la mejor implementación puede alcanzar, y puede determinar a qué costos se alcanzará.

El diseño se caracteriza usualmente por un gran número de decisiones técnicas individuales. En orden de transformar el desarrollo de software en una disciplina de ingeniería, se debe sistematizar tales decisiones, hacerlas más explícitas y técnicas, y menos implícitas y artesanales.

Un ingeniero no busca simplemente *una* solución, busca la *mejor* solución, dentro de las *limitaciones* reconocidas, y realizando *compromisos* requeridos en el trabajo del mundo real.

En orden de convertir el diseño de sistemas de computadoras en una disciplina de ingeniería, previo a todo, debemos definir *objetivos técnicos claros* para los programas de computadora. Es esencial además comprender las *restricciones* primarias que condicionan las soluciones posibles.

Para realizar decisiones concisas y deliberadas, debemos identificar los *puntos de decisión*.

Finalmente necesitamos una *metodología* que nos asista en la *toma de decisiones*.

Dadas estas cosas: objetivos, restricciones, decisiones reconocidas, y una metodología efectiva, podemos obtener soluciones de ingeniería, y no artesanales.

Diseño estructurado y calidad del software

Un concepto importante a clarificar es el de *calidad*. Desafortunadamente, muchos diseñadores se conforman con un sistema que “funcione” sin reparar en un *buen* sistema.

Una corriente de pensamiento estima que un programa es bueno si sus algoritmos son astutos y no obvios a otro programador; esto refleja la “inteligencia” del programador.

Otra escuela de pensamiento asocia calidad con incremento de la velocidad de ejecución y disminución de los requerimientos de memoria central. Estos son aspectos de un concepto más amplio: *eficiencia*. En general, se busca diseños que hagan un uso inteligente de los *recursos*. Estos recursos no incluyen solamente procesador y memoria, también incluyen almacenamiento secundario, tiempo de periféricos de entrada/salida, tiempo de líneas de teleproceso, tiempo de personal y más.

Otra medida de calidad es la *confiabilidad*. Es importante notar que si bien la confiabilidad del software puede ser vista como un problema de depuración de errores en los programas, es también un problema de diseño. La confiabilidad se expresa en como MTBF (mean time between fairules: tiempo medio entre fallas).

Un concepto muy relacionado a la confiabilidad y de suma importancia es el de *mantenibilidad*. Podemos definir la mantenibilidad como:

$$\text{Mantenibilidad del sistema} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}.$$

donde:

- MBTF: tiempo medio entre fallas.
- MTTR: tiempo medio de reparación (mean time to repair).

Diremos que un sistema es mantenible si permite la detección, análisis, rediseño y corrección de errores fácilmente.

En tanto la mantenibilidad afecta la viabilidad del sistema en un entorno relativamente constante, la *modificabilidad* influye en los costos de mantener un sistema viable en condiciones de cambio de requerimientos. La modificabilidad es la posibilidad de realizar modificaciones y extensiones a partes del sistema, o agregar nuevas partes con facilidad (no corrección de errores).

En estudios realizados se determinó que las organizaciones abocadas al procesamiento de datos invierten aproximadamente un 50% del presupuesto en mantenimiento de los sistemas, involucrando esto corrección de errores y modificaciones, razón por la cual la mantenibilidad y la modificabilidad son dos objetivos primarios en el diseño de software.

La *flexibilidad* representa la facilidad de que el mismo sistema pueda realizar variaciones sobre una misma temática, sin necesidad de modificaciones.

La *generalidad* expresa el alcance sobre un determinado tema.

Flexibilidad y generalidad son dos objetivos importantes en el diseño de sistemas del tipo de propósitos generales.

La *utilidad* o facilidad de uso es un factor importante que influye en el éxito del sistema y su aceptación por parte del usuario. Un sistema bien diseñado pero con interfaces muy “duras” tiende a ser resistido por los usuarios.

Finalmente diremos que eficiencia, mantenibilidad, modificabilidad, flexibilidad, generalidad y utilidad, con componentes de la *calidad* objetiva de un sistema.

En términos simples también diremos que nuestro objetivo primario es obtener sistemas de *costo mínimo*. Es decir, es nuestro interés obtener sistemas económicos para desarrollar, operar, mantener y modificar.

Restricciones, compromisos y decisiones del Diseño

Podemos ver los objetivos técnicos del diseño como constituyendo una “función objetivo” de un problema de optimización, la cual se desea maximizar, sujeta a ciertas restricciones.

Como regla, las restricciones sobre un proceso de diseño de un sistema, caen en dos categorías: *restricciones de desarrollo* y *restricciones operacionales*.

Las **restricciones de desarrollo** son limitaciones al consumo de recursos durante el período de desarrollo, y pueden ser expresadas en términos generales o descomponerla en sus partes como ser tiempo de máquina y horas/hombre. Dentro de las restricciones de desarrollo, entran también las *restricciones de planificación*. Estas se refieren a metas y plazos a ser cumplidos (“el módulo X debe terminarse para Febrero”).

Las **restricciones operacionales** pueden ser expresadas en términos técnicos, como ser máximo tamaño de memoria disponible, máximo tiempo de respuesta aceptable, etc.

El carácter de muchas decisiones de diseño no fija límites rígidos, si no un intervalo de tolerancia, dentro del cual el diseñador puede moverse a costa de variaciones en otros aspectos del sistema. Por ejemplo se puede priorizar eficiencia, en detrimento de facilidad de mantenimiento, o velocidad de ejecución contra tamaño de memoria utilizada.

La esencia del diseño en el mundo real y las decisiones inherentes al mismo es obtener una solución de *compromiso*.

El diseño total es el resultado acumulativo de un gran número de *decisiones técnicas* incrementales.

Principios utilizados por el Diseño Estructurado

Abstracción

La noción psicológica de abstracción permite concentrarse en un problema al mismo nivel de generalización, independientemente de los detalles irrelevantes de bajo nivel. El uso de la abstracción también permite trabajar con conceptos y términos que son familiares al entorno del problema, sin tener que transformarlos a una estructura no familiar.

Cada paso de un proceso de ingeniería de software es un refinamiento del nivel de abstracción de la solución de software.

Conforme nos movemos por diferentes niveles de abstracción, trabajamos para crear *abstracciones* de datos y de procedimientos. Una *abstracción procedural* es una determinada secuencia de instrucciones que tienen una función limitada y específica.

Una *abstracción de datos* es una determina colección de datos que describen un objeto.

- **Rumbaugh: O.O. Modeling and Design**
  La abstracción es el exámen selectivo de ciertos aspectos de un problema. El objetivo de la abstracción es aislar aquellos aspectos que son importantes para algún propósito y suprimir aquellos aspectos que no son importantes. La abstracción debe realizarse siempre con un propósito, ya que el propósito determina que es y que no es relevante. Muchas abstracciones son posibles sobre una misma cosa, dependiendo de cual sea su propósito.
- **Alan Cameron Will (Object Expert Jan/Feb 1996)**
  La abstracción, para mí, es cercana a palabras como "teórico", "esotérico", "académico", e "impráctico". Pero en un sentido en particular, significa la separación de los aspectos más importantes de un determinado problema, del detalle. Este el el único camino que tengo para abordar con mi mente finita cualquier tema complejo.

Refinamiento sucesivo

El *refinamiento sucesivo* es una primera estrategia de diseño descendente propuesta por Niclaus Wirth. La arquitectura de un programa se desarrolla en niveles sucesivos de refinamiento de los detalles procedimentales. Se desarrolla una jerarquía descomponiendo una declaración macroscópica de una función de una forma sucesiva, hasta que se llega a las sentencias del lenguaje de programación.

Modularidad

La arquitectura implica modularidad, el software se divide en componentes con nombres y ubicaciones determinados, que se denominan *módulos*, y que se integran para satisfacer los requisitos del problema.

Arquitectura del software

La *arquitectura del software* se refiere a dos características importantes del software de computadoras:

- la estructura jerárquica de los componentes procedimientales (módulos)
- la estructura de datos

Jerarquía de control

La *jerarquía de control*, también denominada *estructura de programa*, representa la organización (frecuentemente jerárquica) de los componentes del programa (módulos) e

implica una jerarquía de control. No representa aspectos procedimentales del software, tales como secuencias de procesos, o la repetición de operaciones.

Estructura de datos

La *estructura de datos* es una representación de la relación lógica existente entre los elementos individuales de datos. Debido a que la estructura de la información afectará invariablemente al diseño procedimental final, la estructura de datos es tan importante como la estructura del programa en la representación de la arquitectura del software.

Procedimientos del software

La estructura del programa define la jerarquía de control, independientemente de las decisiones y secuencias de procesamiento. El procedimiento del software se centra sobre los detalles de procesamiento de cada módulo individual.

El procedimiento debe proporcionar una especificación precisa del procesamiento, incluyendo la secuencia de sucesos, los puntos concretos de decisiones, la repetición de operaciones, e incluso la organización/estructura de los datos.

Ocultamiento de la información

El principio de *ocultamiento de la información* sugiere que los módulos se han de caracterizar por decisiones de diseño que los oculten unos a otros. Los módulos deben especificarse y diseñarse de forma que la información (procedimientos y datos) contenida dentro de un módulo sea accesible a otros módulos únicamente a través de las *interfaces* formales establecidas para cada módulo.

Conceptos Básicos de Diseño Estructurado

Estrategia del Diseño Estructurado

Cuando se trata con un problema de diseño de reducida envergadura, por ejemplo un sistema que pueda ser desarrollado en un par de semanas,no se tienen mayores problemas, y el desarrollador puede tener todos los elementos del problema “en mente” a la vez. Sin embargo cuando se trabaja en proyectos de gran escala, es difícil que una sola persona sea capaz de llevar todas las tareas y tener en mente todos los elementos a la vez.

El diseño exitoso se basa en un viejo principio conocido desde los días de Julio César: *Divide y conquistarás*.

Específicamente, diremos que el costo de implementación de un sistema de computadora podrá minimizarse cuando pueda separarse en partes:

- *manejablemente pequeñas*
- *solucionables separadamente*

Por supuesto la interpretación de manejablemente pequeña varía de persona en persona. Por otro lado muchos intentos de particionar sistemas en pequeñas partes arribaron incrementos en los tiempos de implementación. Esto se debe fundamentalmente al segundo punto: solucionables separadamente. En muchos sistemas para implementar la parte A, debemos conocer algo sobre la B, y para implementar algo de B, debemos conocer algo de C.

De manera similar, podemos decir que el costo de *mantenimiento* puede ser minimizado cuando las partes de un sistema son:

- *fácilmente relacionables con la aplicación*
- *manejablemente pequeñas*

- *corregibles separadamente*

Muchas veces la persona que realiza la modificación no es quien diseñó el sistema. Es importante que las partes de un sistema sean manejablemente pequeñas en orden de simplificar el mantenimiento. Un trabajo de encontrar y corregir un error en una “pieza” de código de 1000 líneas es muy superior a hacerlo con piezas de 20 líneas. No solo disminuye el tiempo de localizar la falla sino que si la modificación es muy engorrosa, puede reescribirse la pieza completamente. Este concepto de “módulos descartables” ha sido utilizado con éxito muchas veces.

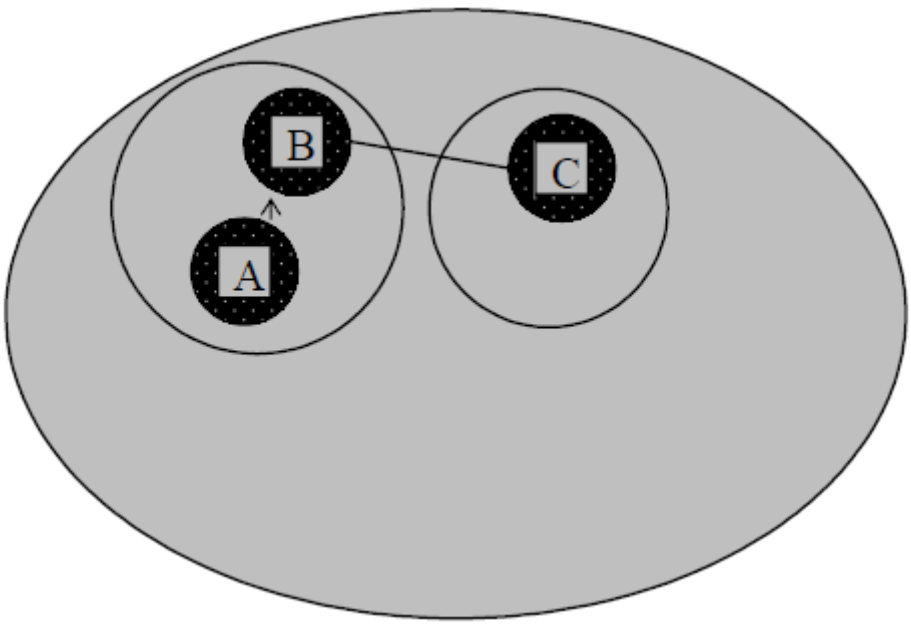
Por otra parte, para minimizar los costos de mantenimiento debemos lograr que cada pieza sea independiente de otra. En otras palabras debemos ser capaces de realizar modificaciones al módulo A sin introducir efectos indeseados en el módulo B.

Finalmente, diremos que el costo de *modificación* de un sistema puede minimizarse si sus partes son:

- *fácilmente relacionables con la aplicación*
- *modificables separadamente*

En resumen, podemos afirmar lo siguiente: los costos de implementación, mantenimiento y modificación, generalmente serán minimizados cuando *cada pieza del sistema corresponda a exactamente una pequeña, bien definida pieza del dominio del problema, y cada relación entre las piezas del sistema corresponde a relaciones entre piezas del dominio del problema*.

En la siguiente figura apreciamos este concepto.



Particionamiento y Organización

Un buen diseño estructurado es un ejercicio de particionamiento y organización de los componentes de un sistema.

Entenderemos por particionamiento, la subdivisión de un problema en subproblemas más pequeños, de tal forma que cada subproblema corresponda a una pieza del sistema. La cuestión es: ¿Dónde y cómo debe dividirse el problema? ¿Qué aspectos del problema deben pertenecer a la misma pieza del sistema, y cuales a distintas piezas? El diseño estructurado responde estas preguntas con dos principios básicos:

- *Partes del problema altamente interrelacionadas deberán pertenecer a la misma pieza del sistema*.

- *Partes sin relación entre ellas, deben pertenecer a diferentes piezas del sistema sin relación directa*.

Otro aspecto importante del diseño estructurado es la organización del sistema. Debemos decidir como se interrelacionan las partes, y que parte está en relación con cual.

El objetivo es organizar el sistema de tal forma que no existan piezas más grandes de lo estrictamente necesario para resolver los aspectos del problema que ella abarca. Igualmente importante, es el evitar la introducción de relaciones en el sistema, que no existe en el dominio del problema.

El concepto de Cajas Negras

Una caja negra es un sistema (o un componente) con entradas conocidas, salidas conocidas, y generalmente transformaciones conocidas, pero del cual no se conoce el contenido en su interior.

En la vida diaria existe innumerable cantidad de ejemplos de uso cotidiano: una radio, un televisor, un automóvil, son cajas negras que usamos a diario sin conocer (en general) como funciona en su interior. Solo conocemos como controlarlos (entradas) y las respuestas que podemos obtener de los artefactos (salidas).

El concepto de caja negra utiliza el principio de *abstracción*.

Este concepto es de suma utilidad e importancia en la ingeniería en general, y por ende en el desarrollo de software. Lamentablemente muchas veces para poder hacer un uso efectivo de determinado módulo, el diseñador debe revisar su contenido ante posibles contingencias como ser comportamientos no deseados ante determinados valores. Por ejemplo es posible que una rutina haya sido desarrollada para aceptar un determinado rango de valores y falla si se la utiliza con valores fuera de dicho rango, o produce resultados inesperados. Una buena documentación en tales casos, es de utilidad pero no transforma al módulo en una verdadera caja negra. Podríamos hablar en todo caso de “cajas blancas”.

Los módulos de programas de computadoras pueden variar en un amplio rango de aproximación al ideal de caja negra. En la mayoría de los casos podemos hablar de “cajas grises”.

La Estructura de los Programas de Computadora

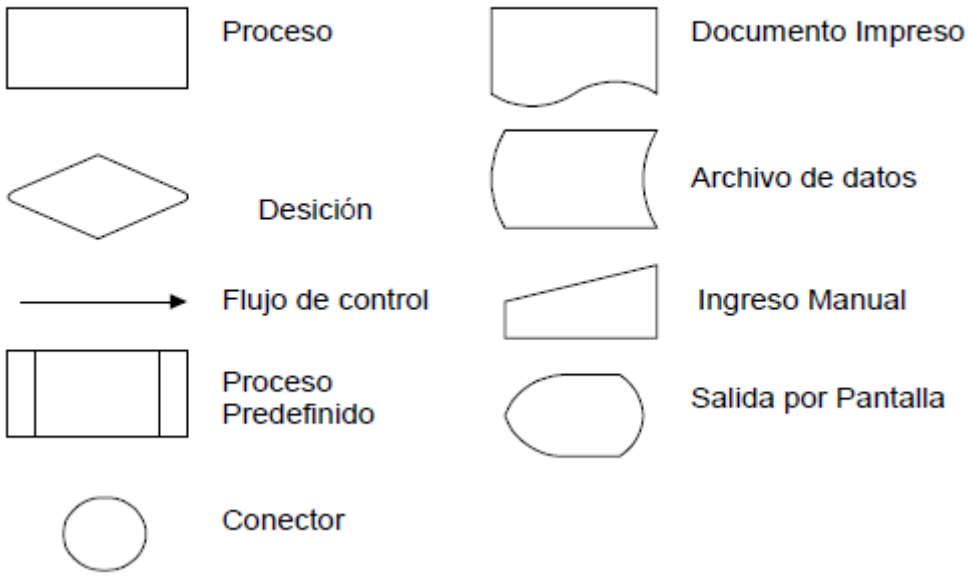
Diagramas de Flujo y Diagramas de Estructura

Normalmente los procedimientos se representan con *diagramas de flujo* (no confundir con diagramas de flujo de datos) los cuales modelan la secuencia de operaciones que realiza el programa a través del tiempo.

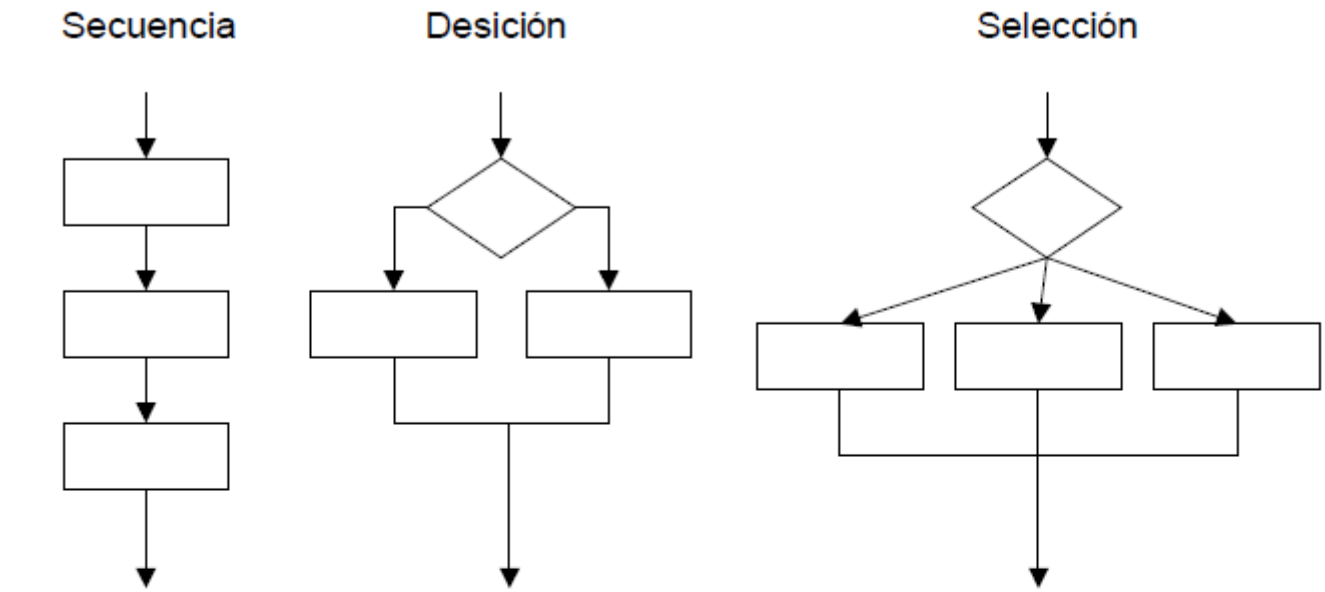
Un *diagrama de estructura* en cambio no modela la secuencia de ejecución sino la *jerarquía de control* existente entre los módulos que conforman el programa, independientemente del factor tiempo. Existe un módulo raíz de máximo nivel, del cual dependen los demás, en una estructura arborescente.

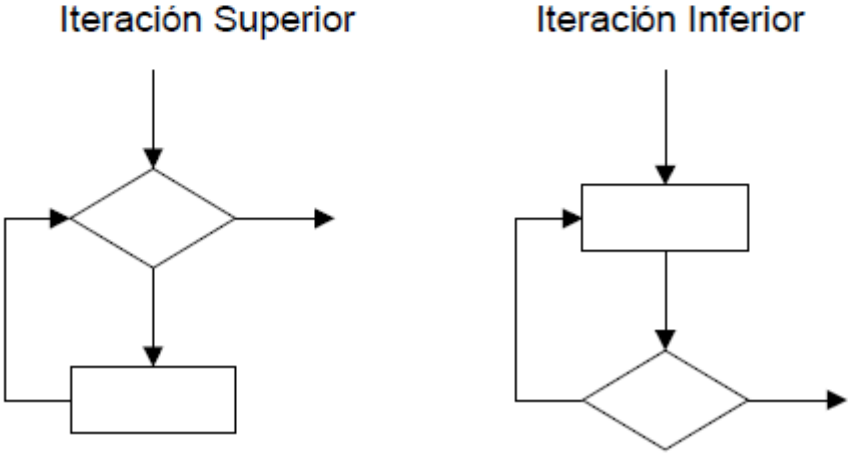
Notación de los Diagramas de Flujo de Control

Simbología Básica

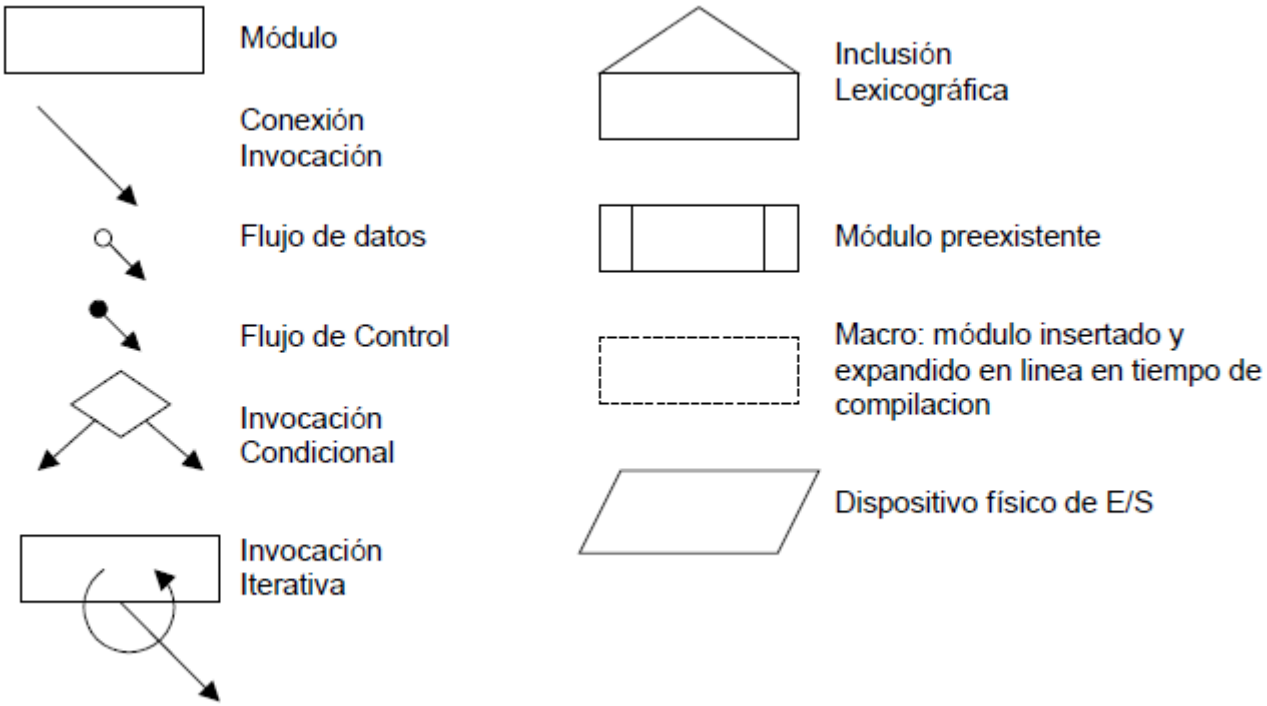


Construcciones Estructuradas





Notación de los Diagramas de Estructura



Ejemplo comparativo entre Diagramas de Procesamiento y de Estructura

Diagrama de Flujo:

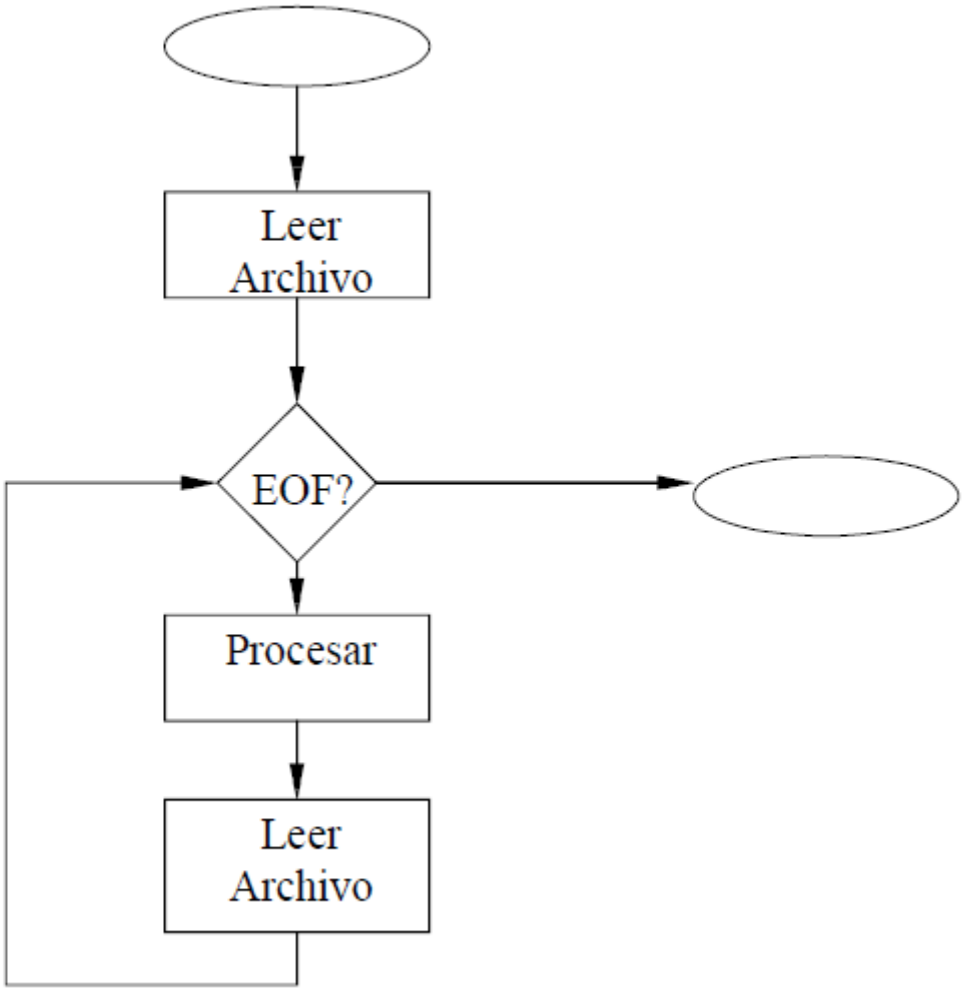
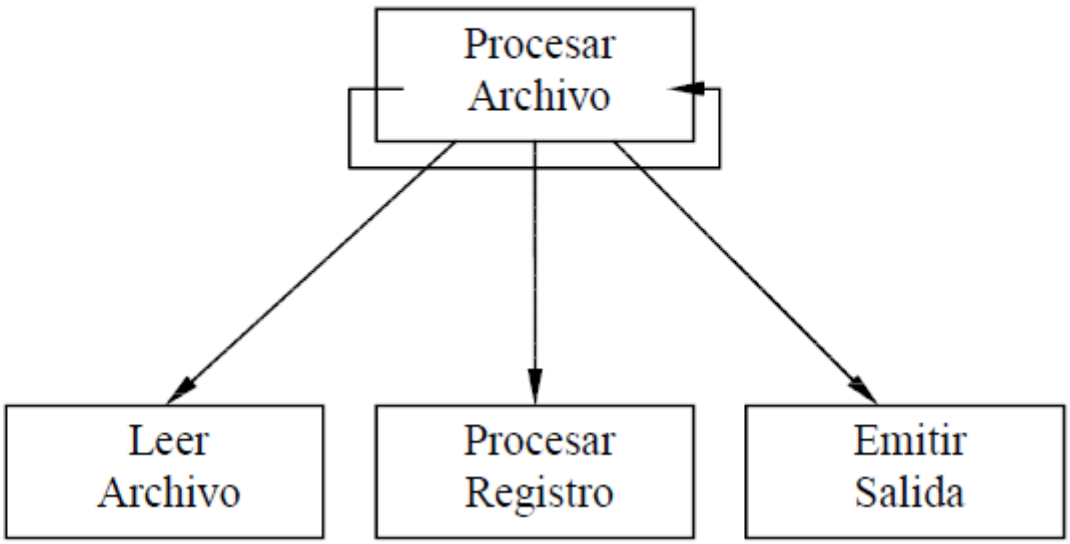


Diagrama de Estructura:



Acoplamiento

Introducción

Muchos aspectos de la modularización pueden ser comprendidos solo si se examinan módulos en relación con otros. En principio veremos el concepto de *independencia*: diremos que dos módulos son totalmente independientes si ambos pueden funcionar completamente sin la presencia del otro. Esto implica que no existen interconexiones entre los módulos, y que se tiene un valor cero en la escala de "dependencia".

En general veremos que a mayor número de interconexiones entre dos módulos, se tiene una menor independencia.

El concepto de independencia funcional es una derivación directa del de modularidad y de los conceptos de abstracción y ocultamiento de la información.

La cuestión aquí es: ¿cuánto debe conocerse acerca de un módulo para poder comprender otro módulos? Cuanto más debamos conocer acerca del módulo B para poder comprender el módulo A, menos independientes serán A y B.

La simple cantidad de conexiones entre módulos, no es una medida completa de la independencia funcional. La dependencia funcional se mide con dos criterios cualitativos: *acoplamiento* y *cohesión*. Estudiaremos en principio el primero de ellos.

Módulos altamente "acoplados" estarán unidos por fuertes interconexiones, módulos débilmente acoplados tendrán pocas y débiles interconexiones, en tanto que los módulos "desacoplados" no tendrán interconexiones entre ellos y serán independientes.

> El *acoplamiento* es un concepto abstracto que nos indica el grado de interdependencia entre módulos.

En la práctica podemos materializarlo como la probabilidad de que en la codificación, depuración, o modificación de un determinado módulo, el programador necesite tomar conocimiento acerca de partes de otro módulo. Si dos módulos están fuertemente acoplados, existe una alta probabilidad de que el programador necesite conocer uno de ellos en orden de intentar realizar modificaciones al otro.

Claramente, el costo total del sistema se verá fuertemente influenciado por el grado de acoplamiento entre módulos.

Factores que influencian el Acoplamiento

Los cuatro factores principales que influyen en el acoplamiento entre módulos son:

- *Tipo de conexión entre módulos*: los sistemas normalmente conectados, tienen menor acoplamiento que aquellos que tienen conexiones patológicas.
- *Complejidad de la interface*: está determinada por la cantidad, accesibilidad y estructura de la información que define la interface.
- *Tipo de flujo de información en la conexión*: los sistemas con acoplamiento de datos tienen menor acoplamiento que los sistema con acoplamiento de control y estos a su vez menos que los que tienen acoplamiento híbrido.
- *Momento en que se produce el ligado de la Conexión*: conexiones ligadas a referentes fijos en tiempo de ejecución, resultan con menor acoplamiento que cuando el ligado tiene lugar en tiempo de carga, el cual tiene a su vez menor acoplamiento que cuando el ligado se realiza en tiempo de ligado (link-edición), el cual tiene menos acoplamiento que el que se realiza en tiempo de compilación, todos los que a su vez tiene menos acoplamiento que cuando el ligado se realiza en tiempo de codificación.

Acoplamiento en Entorno Común (common-environment coupling)

Siempre que dos o más módulos interactúan con un entorno de datos común, se dice que dichos módulos están en *acoplamiento por entorno común*.

Ejemplos de entorno común pueden ser áreas de datos globales como la DATA DIVISION de COBOL, un archivo en disco.

El acoplamiento de entorno común es una forma de acoplamiento de segundo orden, distinto de los tratados anteriormente. La severidad del acoplamiento dependerá de la cantidad de módulos que acceden simultáneamente al entorno común. En el caso extremo de solo dos módulos donde uno utiliza como entrada los datos generados por el otro hablaremos de un acoplamiento de *entrada/salida*.

El punto es que el acoplamiento por entorno común no es necesariamente malo y deba ser evitado a toda costa. Por el contrario existen ciertas circunstancias en que es una opción válida.

Desacoplamiento

El concepto de acoplamiento invita a un concepto recíproco: *desacoplamiento*.

Desacoplamiento es cualquier método sistemático o técnica para hacer más independientes a los módulos de un programa.

Cada tipo de acoplamiento generalmente sugiere un método de desacoplamiento. Por ejemplo, el acoplamiento causado por ligado, puede desacoplarse cambiando los parámetros apropiados.

El desacoplamiento desde el punto de vista funcional, rara vez puede realizarse, excepto en los comienzos de la fase del diseño.

Como regla general, una disciplina de diseño que favorezca el acoplamiento de entrada/ salida y el acoplamiento de control por sobre el acoplamiento por contenido y el acoplamiento híbrido, y que busque limitar el alcance del acoplamiento por entorno común es el enfoque más efectivo.

Otras técnicas para reducir el acoplamiento son:

- Convertir las referencias implícitas en explícitas. Lo que puede verse con mayor facilidad es más fácil de comprender.
- Estandarización de las conexiones.
- Uso de “buffers” para los elementos comunicados en una conexión. Si un módulo puede ser diseñado desde el comienzo asumiendo que un buffer mediará cada corriente de comunicación, las cuestiones temporización, velocidad, frecuencia, etc, dentro de un módulo no afectarán al diseño de otros.
- Localización. Utilizado para reducir el acoplamiento por entorno común. Consiste en dividir el área común en regiones para que los módulos solo tengan acceso a aquellos datos que les son de su estricta incumbencia.

Cohesión

Introducción: Relación Funcional

Hemos visto que la determinación de módulos en un sistema no es arbitraria. La manera en la cual dividimos físicamente un sistema en piezas (particularmente en relación con la

estructura del problema) puede afectar significativamente la complejidad estructural del sistema resultante, así como el número total de referencias intermodulares.

Adaptar el diseño del sistema a la estructura del problema (o estructura de la aplicación, o dominio del problema) es una filosofía de diseño sumamente importante. A menudo encontramos que elementos de procesamiento del dominio del problema altamente relacionados, son trasladados en código altamente interconectado. Las estructuras que agrupan elementos del problema altamente interrelacionados, tienden a ser modularmente efectivas.

Imaginemos que tengamos una magnitud para medir el grado de relación funcional existente entre pares de módulos. En términos de tal medida, diremos que el sistema más modularmente efectivo será aquel cuya suma de relación funcional entre pares de elementos que pertenezcan a diferentes módulos sea mínima. Entre otras cosas, esto tiende a minimizar el número de conexiones intermodulares requeridas y el acoplamiento intermodular.

> Esta relación funcional intramodular se conoce como *cohesión*. La cohesión es la medida cualitativa de cuan estrechamente relacionados están los elementos internos de un módulo.

Otros términos utilizados frecuentemente son “fuerza modular”, “ligazón” y “funcionalidad”.

En la práctica un elemento de procesamiento simple aislado, puede estar funcionalmente relacionado en diferentes grados a otros elementos. Como consecuencia, diferentes diseñadores, con diferentes “visiones” o interpretaciones de un mismo problema, pueden obtener diferentes estructuras modulares con diferentes niveles de cohesión y acoplamiento. A esto se suma el inconveniente de que muchas veces es difícil evaluar el grado de relación funcional de un elemento respecto de otro.

La cohesión modular puede verse como el cemento que amalgama junto a los elementos de procesamiento dentro de un mismo módulo. Es el factor más crucial en el diseño estructurado, y el de mayor importancia en un diseño modular efectivo.

Este concepto representa la técnica principal que posee un diseñador para mantener su diseño lo más semánticamente próximo al problema real, o dominio del problema.

Claramente los conceptos de cohesión y acoplamiento están íntimamente relacionados. Un mayor grado de cohesión implica uno menor de acoplamiento. Maximizar el nivel de cohesión intramodular en todo el sistema resulta en una minimización del acoplamiento intermodular.

Niveles de Cohesión

Diferentes principios asociativos fueron desenvolviéndose a través de los años por medio de la experimentación, argumentos teóricos, y la experiencia práctica de muchos diseñadores.

Existen siete niveles de cohesión distinguibles por siete principios asociativos. Estos se listan a continuación en orden creciente del grado de cohesión, de menor a mayor relación funcional:

- Cohesión Casual (la peor)
- Cohesión Lógica (sigue a la peor)
- Cohesión Temporal (de moderada a pobre)
- Cohesión de Procedimiento (moderada)
- Cohesión de Comunicación (moderada a buena)
- Cohesión Secuencial

- Cohesión Funcional (la mejor)

Podemos visualizar el grado de cohesión como un espectro que va desde un máximo a un mínimo.

Cohesión Casual (la peor)

La *cohesión casual* ocurre cuando existe poca o ninguna relación entre los elementos de un módulo.

La cohesión casual establece un punto cero en la escala de cohesión.

Es muy difícil encontrar módulos puramente casuales. Puede aparecer como resultado de la modularización de un programa ya escrito, en el cual el programador encuentra una determinada secuencia de instruccciones que se repiten de forma aleatoria, y decide por lo tanto agruparlas en una rutina.

Otro factor que influenció muchas veces la confección de módulos casualmente cohesivos, fue la mala práctica de la programación estructurada, cuando los programadores mal entendían que modularizar consistía en cambiar las sentencias GOTO por llamadas a subrutinas.

Finalmente diremos que si bien en la práctica es difícil encontrar módulos casualmente cohesivos en su totalidad, es común que tengan elementos casualmente cohesivos. Tal es el caso de operaciones de inicialización y terminación que son puestas juntas en un módulo superior.

Debemos notar que si bien la cohesión casual no es necesariamente perjudicial (de hecho es preferible un casualmente cohesivo a uno lineal), dificulta las modificaciones y mantenimiento del código.

Cohesión Lógica (sigue a la peor)

Los elementos de un módulo están *lógicamente* asociados si puede pensarse en ellos como pertenecientes a la misma clase lógica de funciones, es decir, aquellas que pueden pensarse como juntas lógicamente.

Por ejemplo, se puede combinar en un módulo simple todos los elementos de procesamiento que caen en la clase de “entradas”, que abarca todas las operaciones de entrada.

Podemos tener un módulo que lea desde consola una tarjeta con parámetros de control, registros con transacciones erróneas de un archivo en cinta, registros con transacciones válidas de otro archivo en cinta, y los registros maestros anterior de un archivo en disco. Este módulo que podría llamarse “Lecturas”, y que agrupa todas las operaciones de entrada, es lógicamente cohesivo.

La cohesión lógica es más fuerte que la casual, debido a que representa un mínimo de asociación entre el problema y los elementos del módulo. Sin embargo podemos ver que un módulo lógicamente cohesivo no realiza una función específica, sino que abarca una serie de funciones.

Cohesión Temporal (de moderada a pobre)

Temporal cohesión significa que todos los elementos de procesamiento de una colección ocurren en el mismo periodo de tiempo durante la ejecución del sistema. Debido a que dicho procesamiento debe o puede realizarse en el mismo periodo de tiempo, los elementos asociados temporalmente pueden combinarse en un único módulo que los ejecute a la misma vez.

Existe una relación entre cohesión lógica y la temporal, sin embargo, la primera no implica una relación de tiempo entre los elementos de procesamiento. La cohesión temporal es más fuerte que la cohesión lógica, ya que implica un nivel de relación más: el factor tiempo. Sin embargo la cohesión temporal aún es pobre en nivel de cohesión y acarrea inconvenientes en el mantenimiento y modificación del sistema.

Un ejemplo común de cohesión temporal son las rutinas de inicialización (start-up) comúnmente encontradas en la mayoría de los programas, donde se leen parámetros de control, se abren archivos, se inicializan variables contadores y acumuladores, etc.

Cohesión de Procedimiento (moderada)

Elementos de procesamiento relacionados *proceduralmente* son elementos de una unidad procedural común. Estos se combinan en un módulo de cohesión procedural. Una unidad procedural común puede ser un proceso de iteración (loop) y de decisión, o una secuencia lineal de pasos. En este último caso la cohesión es baja y es similar a la cohesión temporal, con la diferencia que la cohesión temporal no implica una determinada secuencia de ejecución de los pasos.

Al igual que en los casos anteriores, para decir que un módulo tiene *solo* cohesión procedural, los elementos de procesamiento deben ser elementos de alguna iteración, decisión, o secuencia, pero no deben estar vinculados con ningún principio asociativo de orden superior.

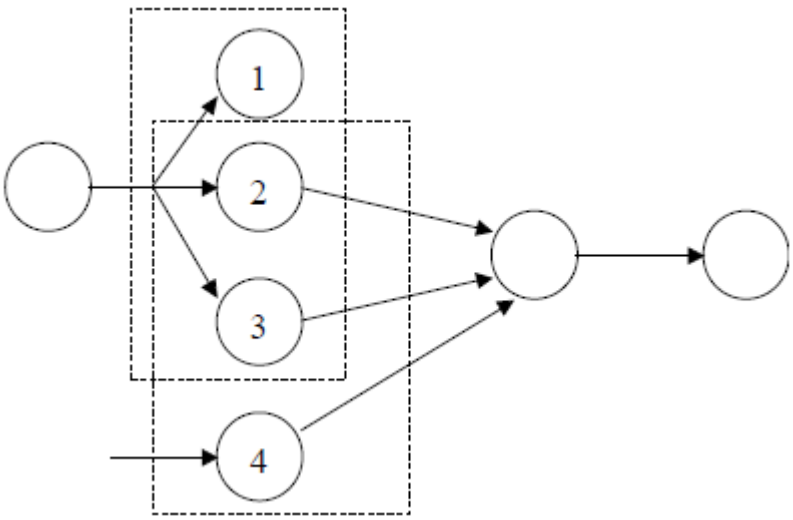
La cohesión procedural asocia elementos de procesamiento sobre la base de sus relaciones algorítmicas o procedurales.

Este nivel de cohesión comúnmente se tiene como resultado de derivar una estructura modular a partir de modelos de procedimiento como ser diagramas de flujo, o diagramas Nassi-Shneiderman.

Cohesión de Comunicación (moderada a buena)

Ninguno de los niveles de cohesión discutidos previamente están fuertemente vinculados a una estructura de problema en particular. *Cohesión de Comunicación* es el menor nivel en el cual encontramos una relación entre los elementos de procesamiento que es intrínsecamente *dependiente del problema.*

Decir que un conjunto de elementos de procesamiento están vinculados por comunicación significa que *todo los elementos operan sobre el mismo conjunto de datos de entrada o de salida*.



En el diagrama de la figura podemos observar que los elementos de procesamiento 1, 2 y 3 están asociados por comunicación sobre la corriente de datos de entrada, en tanto que 2, 3 y 4 se vinculan por los datos de salida.

Los diagramas de flujo de datos (DFD) son un medio objetivo para determinar si los elementos en un módulo están asociados por comunicación.

Las relaciones por comunicación presentan un grado de cohesión aceptable.

La cohesión por comunicación es común en aplicaciones comerciales. Ejemplos típicos pueden ser:

- un módulo que imprima o grabe un archivo de transacciones
- un módulo que reciba datos de diferentes fuentes y los transforme y ensamble en una línea de impresión

Cohesión Secuencial

El siguiente nivel de cohesión en la escala es la asociación *secuencial*. En ella, los datos de salida (resultados) de un elemento de procesamiento sirven como datos de entrada al siguiente elemento de procesamiento.

En términos de un diagrama de flujo de datos de un problema, la cohesión secuencial combina una cadena lineal de sucesivas transformaciones de datos.

Este es claramente un principio asociativo relacionado con el dominio del problema.

Cohesión Funcional (la mejor)

En el límite superior del espectro de relación funcional encontramos la cohesión funcional. En un módulo completamente funcional, cada elemento de procesamiento, es parte integral de, y esencial para, la realización de una función simple.

En términos prácticos podemos decir que cohesión funcional es aquella que no es secuencial, por comunicación, por procedimiento, temporal, lógica o casual.

Los ejemplos más claros y comprensibles provienen del campo de las matemáticas. Un módulo para realizar el cálculo de raíz cuadrada ciertamente será altamente cohesivo, y probablemente, completamente funcional. Es improbable que haya elementos superfluos más allá de los absolutamente esenciales para realizar la función matemática, y es improbable que elementos de procesamiento puedan ser agregados sin alterar el cálculo de alguna forma.

En contraste, un módulo que calcule la raíz cuadrada y coseno, es improbable que sea enteramente funcional (deben realizarse dos funciones ambiguas).

En adición a estos ejemplos matemáticos obvios, usualmente podemos reconocer módulos funcionales que son elementales en naturaleza. Un módulo llamado LEER-REGISTRO-MAESTRO o TRATAR-TRANS-TIPO3, presumiblemente serán funcionalmente cohesivos, en cambio TRATAR-TODAS-TRANS presumiblemente realizará más de una función y será lógicamente cohesivo.

Criterios para establecer el grado de cohesión

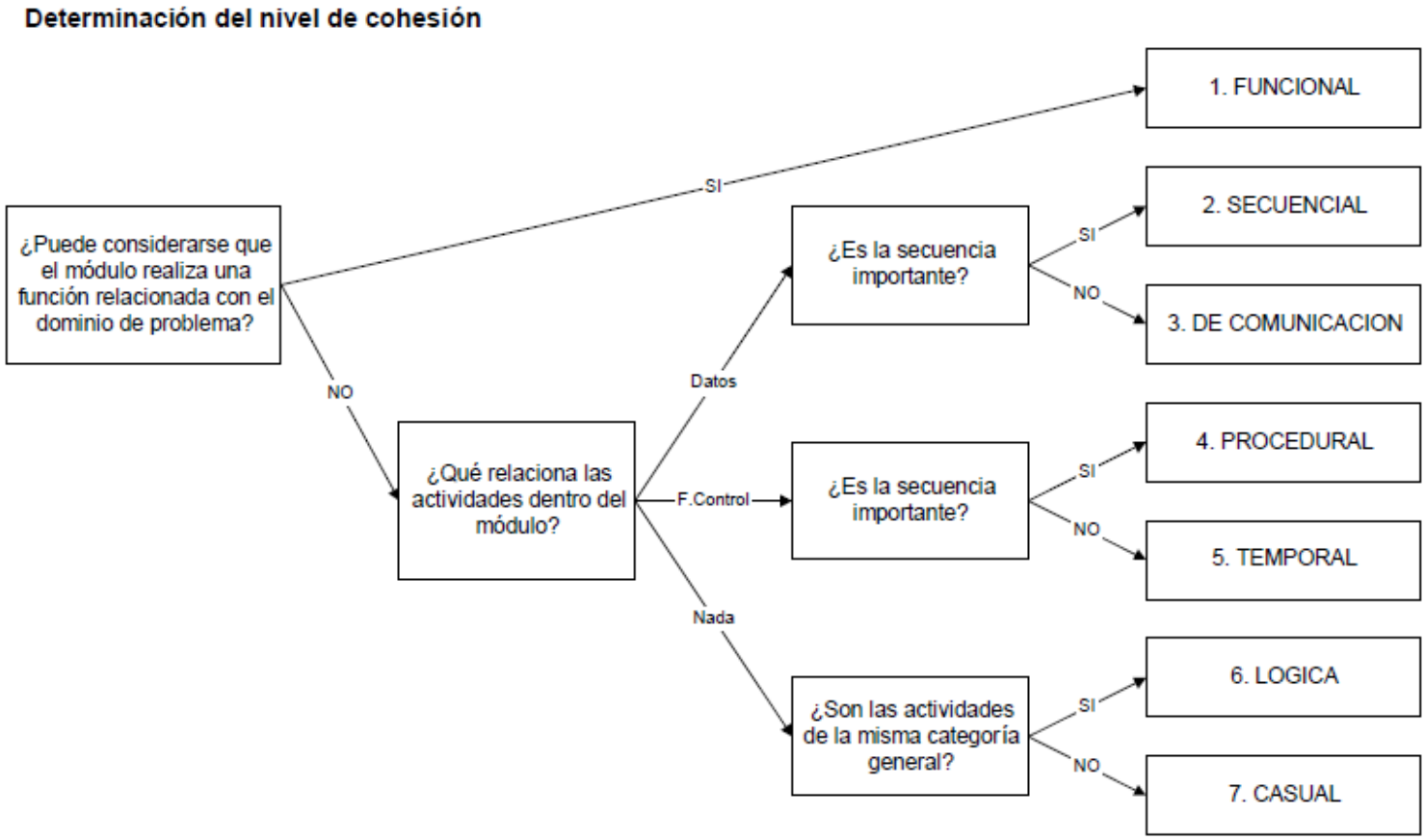
Una técnica útil para determinar si un módulo está acotado funcionalmente, es escribir una frase que describa la función (propósito) del módulo y luego examinar dicha frase. Puede hacerse la siguiente prueba:

1. Si la frase resulta ser una sentencia compuesta, contiene una coma, o contiene más de un verbo, probablemente el módulo realiza más de una función: por tanto, probablemente tienen vinculación secuencial o de comunicación.
2. Si la frase contiene palabras relativas al tiempo, tales como “primero”, “a continuación”, “entonces”, “después”, “cuando”, “al comienzo”, etc, entonces probablemente el módulo tiene una vinculación secuencial o temporal.
3. Si el predicado de la frase no contiene un objeto específico sencillo a continuación del verbo, probablemente el módulo esté acotado lógicamente. Por ejemplo *editar todos los datos* tiene una vinculación lógica; *editar sentencia fuerte* puede tener vinculación funcional.
4. Palabras tales como “inicializar”, “limpiar”, etc, implican vinculación temporal.

Los módulos acotados funcionalmente siempre se pueden describir en función de sus elementos usando una sentencia compuesta. Pero si no se puede evitar el lenguaje anterior, siendo aún una descripción completa de la función del módulo, entonces probablemente el módulo no esté acotado funcionalmente.

Es importante notar que no es necesario determinar el nivel preciso de cohesión. En su lugar, lo importante es intentar conseguir una cohesión alta y saber reconocer la cohesión baja, de forma que se pueda modificar el diseño del software para que disponga de una mayor independencia funcional.

Árbol de valuación



Análisis de Transformación

Introducción

> El *Análisis de Transformación, o diseño centrado en la transformación*, es una estrategia para derivar diseños estructurados que son bastante buenos (con respecto a su modularidad) y que necesitan solo una modesta reestructuración para llegar al diseño final.

Es una forma particular de la estrategia descendente (top-down), que toma ventaja de la perspectiva global. Aplicado rigurosamente, el análisis de transacción conduce a estructuras que son altamente factorizadas. Produce un número variable de módulos en los niveles intermedios de la jerarquía, los cuales representan composición de funciones básicas. Siempre se trata de evitar que los módulos intermedios realicen cualquier "trabajo" excepto el de control y coordinación de sus subordinados.

El propósito de la estrategia es el de identificar las funciones de procesamiento primarias del sistema, las entradas de alto nivel de dichas funciones, y las salidas de alto nivel. Se crean módulos de alto nivel dentro de la jerarquía que realizan cada una de estas tareas: creación de entradas de alto nivel, transformación de entradas en salidas de alto nivel, y procesamiento de dichas salidas.

> El análisis de transformación es un modelo de *flujo de información* más que un modelo procedural.

La estrategia de análisis de transformación consiste de **cuatro pasos** principales:

1. Plantear el problema como un diagrama de flujo de datos.
2. Identificar los elementos de datos aferentes y eferentes.
    - *Datos Aferentes*.
      Son aquellos elementos de datos de alto nivel que habiendo sido removidos de sus entradas físicas, todavía pueden considerarse entradas al sistema.
    - *Datos Eferentes*.
      Son elementos de datos que desde sus salidas física a través de los flujos, hasta que no puedan seguir siendo considerados como datos de salida del sistema.
3. Factorización del primer nivel.
4. Factorización de las ramas aferente, eferente y de transformación.

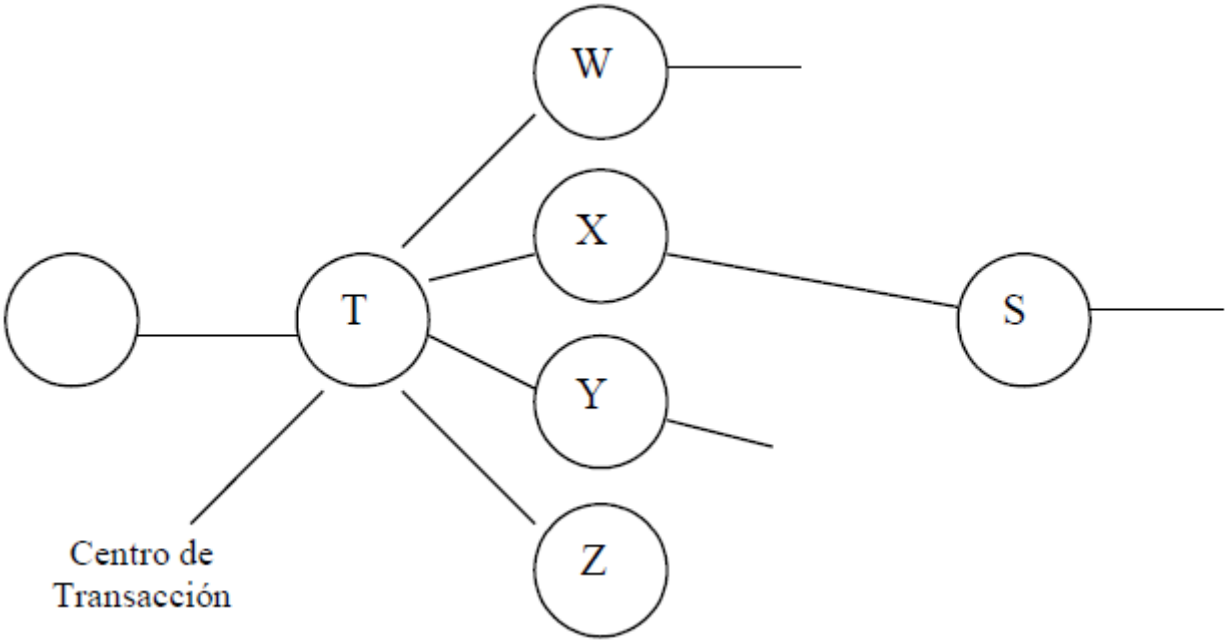
Análisis de Transacción

Introducción

En el anterior capítulo exploramos la estrategia del análisis de transformación como la estrategia principal para el diseño de programas y sistemas bien estructurados. En verdad, el análisis de transformación, servirá de guía en el diseño de la mayoría de los sistemas. Sin embargo hay numerosas situaciones en las cuales estrategias adicionales pueden utilizarse para suplementar, y aún reemplazar, el enfoque básico del análisis de transformación.

Una de estas estrategias suplementarias principales se conoce como *Análisis de Transacción*.

El análisis de transacción es sugerido por un DFD del siguiente tipo:



En este DFD existe una transformación que bifurca la corriente de datos de entrada en varias corrientes de salida discretas. En muchos sistemas tal transformación puede ocurrir dentro de la transformación *central*. En otros, podremos encontrarla tanto en las ramas aferentes como eferentes del diagrama de estructura.

La frase análisis de transacción sugiere que construiremos un sistema alrededor del concepto de “transacción”, y para muchos la palabra transacción está asociada con sistemas administrativos. Esto si bien es cierto, es común encontrar centros de transacción en los sistemas administrativos, también pueden encontrarse en otro tipo de sistemas como los de tiempo real, aplicaciones de ingeniería, etc.

Un factor importante es como definimos el término transacción. En el sentido más general podemos decir:

> *Una transacción es cualquier elemento de datos, control, señal, evento, o cambio de estado, que causa, dispara o inicia alguna acción o secuencia de acciones.*

Acorde a esta definición, un gran número de situaciones encontradas en aplicaciones de procesamiento de datos comunes pueden ser consideradas transacciones. Por ejemplo cualquiera de los siguientes casos pueden considerarse transacciones:

- El operador presiona el botón de inicio de un dispositivo de entrada.
- Algún tipo de datos de entrada que designe un ingreso en el inventario.
- Un carácter de escape desde una terminal, indicando la necesidad e un procesamiento especial.
- Una interrupción de hardware ante un índice fuera de los rangos definidos dentro de un programa de aplicación.
- Un cuelgue o descuelgue de teléfono para un sistema de control de llamadas telefónicas.

Bibliografía

- Universidad Tecnológica Nacional – F.R.R.

Pruebas. Planificación y documentación. Utilización de datos de prueba. Pruebas de software, hardware, procedimientos y datos.

Documentación y Pruebas en el Desarrollo Tradicional del Software

Documentación y Desarrollo de Software

En general se habla mucho de la documentación, pero no se hace, no se le asigna presupuesto, no se la mantiene y casi nunca está al día en los proyectos de desarrollo de software. Lo importante es la disponibilidad de la documentación que se necesita en el momento en que se la necesita.

Muchas veces se hace porque hay que hacerla y se escribe, con pocas ganas, largos textos, a la vez que se está convencido de estar haciendo un trabajo inútil. A veces se peca por exceso y otras por defecto. Ocurre mucho en la Web y con productos RAD. En ocasiones se olvida que el mantenimiento también debe llegar a la documentación.

La documentación se suele clasificar en función de las personas o grupos a los cuales está dirigida:

- Documentación para los desarrolladores.
- Documentación para los usuarios.
- Documentación para los administradores o soporte técnico.

La documentación para desarrolladores es aquélla que se utiliza para el propio desarrollo del producto y, sobre todo, para su mantenimiento futuro. Se documenta para comunicar estructura y comportamiento del sistema o de sus partes, para visualizar y controlar la arquitectura del sistema, para comprender mejor el mismo y para controlar el riesgo, entre otras cosas. Obviamente, cuanto más complejo es el sistema, más importante es la documentación.

En este sentido, todas las fases de un desarrollo deben documentarse: requerimientos, análisis, diseño, programación, pruebas, etc. Una herramienta muy útil en este sentido es una notación estándar de modelado, de modo que mediante ciertos diagramas se puedan comunicar ideas entre grupos de trabajo.

Hay decenas de notaciones, tanto estructuradas como orientadas a objetos. Un caso particular es el de UML. De todas maneras, los diagramas son muy útiles, pero siempre y cuando se mantengan actualizados, por lo que más vale calidad que cantidad.

La documentación para desarrolladores a menudo es llamada **modelo**, pues es una simplificación de la realidad para comprender mejor el sistema como un todo.

Otro aspecto a tener en cuenta cuando se documenta o modela, es el del nivel de detalle. Así como cuando construimos planos de un edificio podemos hacer planos generales, de arquitectura, de instalaciones y demás, también al documentar el software debemos cuidar el nivel de detalle y hacer diagramas diferentes en función del usuario de la documentación, concentrándonos en un aspecto a la vez.

De toda la documentación para los desarrolladores, nos interesa especialmente en esta obra aquella que se utiliza para documentar la programación, y en particular hemos analizado la que se usa para documentar desarrollos orientados a objetos.

La documentación para usuarios es todo aquello que necesita el usuario para la instalación, aprendizaje y uso del producto. Puede consistir en guías de instalación, guía del usuario, manuales de referencia y guía de mensajes.

En el caso de los usuarios que son programadores, esta documentación se debe acompañar con ejemplos de uso recomendados o de muestra y una reseña de efectos no evidentes de las bibliotecas.

Más allá de todo esto, debemos tener en cuenta que la estadística demuestra que los usuarios no leen los manuales a menos que nos les quede otra opción. Las razones pueden ser varias, pero un análisis de la realidad muestra que se recurre a los manuales solamente cuando se produce un error o se desconoce cómo lograr algo muy puntual, y recién cuando la ayuda en línea no satisface las necesidades del usuario. Por lo tanto, si bien es cierto que debemos realizar manuales, la existencia de un buen manual nunca nos libera de hacer un producto amigable para el usuario, que incluso contenga ayuda en línea. Es incluso deseable proveer un software tutorial que guíe al usuario en el uso del sistema, con apoyo multimedia, y que puede llegar a ser un curso on-line.

Buena parte de la documentación para los usuarios puede empezar a generarse desde que comienza el estudio de requisitos del sistema. Esto está bastante en consonancia con las ideas de *extreme programming* y con metodologías basadas en casos de uso.

La documentación para administradores o soporte técnico, a veces llamada manual de operaciones, contiene toda la información sobre el sistema terminado que no hace al uso por un usuario final. Es necesario que tenga una descripción de los errores posibles del sistema, así como los procedimientos de recuperación. Como esto no es algo estático, pues la aparición de nuevos errores, problemas de compatibilidad y demás nunca se puede descartar, en general el manual de operaciones es un documento que va engrosándose con el tiempo.

Las Pruebas en el Desarrollo de Software

Calidad, errores y pruebas

La calidad no es algo que se pueda agregar al software después de desarrollado si no se hizo todo el desarrollo con la cantidad en mente. Muchas veces parece que el software de calidad es aquel que brinda lo que se necesita con adecuada velocidad de procesamiento. En realidad, es mucho más que eso. Tiene que ver con la corrección, pero también con usabilidad, costo, consistencia, confiabilidad, compatibilidad, utilidad, eficiencia y apego a los estándares.

Todos estos aspectos de la calidad pueden ser objeto de tests o pruebas que determinen el grado de calidad. Incluso la documentación para el usuario debe ser probada.

Como en todo proyecto de cualquier índole,siempre se debe tratar que las fallas sean mínimas y poco costosas, durante todo el desarrollo. Y además, es sabido que cuanto más tarde se encuentra una falla, más caro resulta eliminarla. Es claro que si un error es descubierto en la mitad del desarrollo de un sistema, el costo de su corrección será mucho menor al que se debería enfrentar en caso de descubrirlo con el sistema instalado y en funcionamiento.

Desde el punto de vista de la programación,nos interesa la ausencia de errores (corrección), la confiabilidad y la eficiencia. Dejando de lado las dos últimas, nos

concentraremos en este capítulo en las pruebas que determinan que un programa está libre de errores.

Un **error** es un comportamiento distinto del que espera un usuario razonable. Puede haber errores aunque se hayan seguido todos los pasos indicados en el análisis y en el diseño, y hasta en los requisitos aprobados por el usuario. Por lo tanto,no necesariamente un apego a los requisitos y un perfecto seguimiento de las etapas nos lleva a un producto sin errores, porque aún en la mente de un usuario pudo haber estado mal concebida la idea desde el comienzo. De allí la importancia del desarrollo incremental, que permite ir viendo versiones incompletas del sistema.

Por lo tanto, una primera fuente de errores ocurre antes de los requerimientos o en el propio proceso de análisis. Pero también hay errores que se introducen durante el proceso de desarrollo posterior. Así, puede haber errores de diseño y errores de implementación. Finalmente, puede haber incluso errores en la propia etapa de pruebas y depuración.

Categorías de pruebas

Según la naturaleza de lo que se esté controlando, las pruebas se pueden dividir en dos categorías:

- Pruebas centradas en la verificación.
- Pruebas centradas en la validación.

Las primeras sirven para determinar la consistencia entre los requerimientos y el programa terminado. Soporta metodologías formales de testeo, de mucho componente matemático. De todas maneras, hay que ser cuidadoso, porque no suele ser fácil encontrar qué es lo que hay que demostrar. La verificación consisten en determinar si estamos construyendo el sistema correctamente, a partir de los requisitos.

En general a los informáticos no les gustan las pruebas formales, en parte porque no las entienden y en parte porque requieren un proceso matemático relativamente complejo.

La validación consiste en saber si estamos construyendo el sistema correcto. Las tareas de validación son más informales. Las pruebas suelen mostrar la presencia de errores, pero nunca demuestran su ausencia.

Las pruebas y el desarrollo de software

La etapa de pruebas es una de las fases del ciclo de vida de los proyectos. Se la podría ubicar después del análisis, el diseño y la programación, pero dependiendo del proyecto en cuestión y del modelo de proceso elegido, su realización podría ser en forma paralela a las fases citadas o inclusive repertirse varias veces durante la duración del proyecto.

La importancia de esta fase será mayor o menor según las características del sistema desarrollado, llegando a ser vital en sistemas de tiempo real u otros en los que los errores sean irrecuperables.

Las pruebas no tienen el objeto de prevenir errores sino de detectarlos. Se efectúan sobre el trabajo realizado y se deben encarar con la intención de descubrir la mayor cantidad de errores posible.

Para realizar las pruebas se requiere gente que disfrute encontrando errores, por eso no es bueno que sea el mismo equipo de desarrollo el que lleve a cabo este trabajo. Además, es un principio fundamental de las auditorías. Por otro lado, es bastante común que a quien le guste programar no le guste probar y viceversa.

A veces se dan por terminadas las pruebas antes de tiempo. En las pruebas de caja blanca no es mala idea probar un 85% de las ramas y dar por terminado luego de esto. Otra posibilidad es la siembra de errores y seguir las pruebas hasta que se encuentren un 85%

de los errores sembrados, lo que presumiblemente implica que se encontró un 86% de los no sembrados. Otros métodos se basan en la estadística y las comparaciones, ya sea por similitud con otro sistema en cantidad de errores o por el tiempo de prueba usado en otro sistema.

En un proyecto ideal, podríamos generar casos de prueba para cubrir todas las posibles entradas y todas las posibles situaciones por las que podría atravesar el sistema. Examinaríamos así exhaustivamente el sistema para asegurar que su comportamiento sea perfecto. Pero hay un problema con esto: el número de casos de prueba para un sistema complejo es tan grande que no alcanzaría una vida para terminar con las pruebas. Como consecuencia, nadie realiza una prueba exhaustiva de nada salvo en sistemas triviales.

En un sistema real, los casos de prueba se deben hacer sobre las partes del sistema en los cuales una buena prueba brinde un mayor retorno de la inversión o en las cuales un error represente un riesgo mayor.

Las pruebas cuestan mucho dinero. Pero para ello existe una máxima: “pague por la prueba ahora o pague el doble por el mantenimiento después”.

Todo esto lleva a que se deban planificar bien las pruebas, con suficiente anticipación, y determinar desde el comienzo los resultados que se deben obtener.

La idea de *extreme programming* es más radical: propone primero escribir los programas de prueba y después la aplicación, obligando a correr las pruebas siempre antes de una integración. Se basa en la idea bastante acertada de que los programas de prueba son la mejor descripción de los requerimientos.

Las pruebas son prácticas a realizar en diversos momentos de la vida del sistema de información para verificar:

- El correcto funcionamiento de los componentes del sistema.
- El correcto ensamblaje entre los distintos componentes.
- El funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
- El funcionamiento correcto del sistema integrado de hardware y software en el entorno de operación.
- Que el sistema cumple con el funcionamiento esperado y permite al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.
- Que los cambios sobre un componente de un sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.

Las diversas pruebas a que debe ser sometido un sistema deben ser realizadas tanto por el equipo de desarrolladores, como por los usuarios, equipos de operación y mantenimiento en la implantación, aceptación y mantenimiento del sistema de información.

Tipos de pruebas

Analizaremos 7 tipos de pruebas:

- Revisiones de código.
- Pruebas unitarias.
- Pruebas de integración.
- Pruebas de sistema.
- Pruebas de implantación.
- Pruebas de aceptación.
- Pruebas de regresión.

No son tipos de pruebas intercambiables, ya que testean cosas distintas.

Otra posible clasificación de las pruebas es:

- De caja blanca o de código
- De caja negra o de especificación

En las primeras se evalúa el contenido de los módulos, mientras en las segundas se trata al módulo como una caja cerrada y se lo prueba con valores de entrada, evaluando los valores de salida. Vistas de este modo, las pruebas de caja negra sirven para verificar especificaciones.

Las pruebas unitarias suelen ser de caja blanca o de caja negra, mientras que las de integración, sistema y aceptación son de caja negra. Las tareas de depuración luego de encontrar errores son más bien técnicas de caja blanca, así como las revisiones de código. En todos los casos, uno de los mayores desafíos es encontrar los datos de prueba: hay que encontrar un subconjunto de todas las entradas que tengan alta probabilidad de detectar el mayor número de errores.

Revisiones de código

Las revisiones de código son las únicas que se podrían omitir de todos los tipos de pruebas, pero tal vez sea buen idea por lo menos hacer alguna de ellas:

- Pruebas de escritorio.
- Recorridos de código.
- Inspecciones de código.

La *prueba de escritorio* rinde muy poco, tal vez menos de lo que cuesta, pero es una costumbre difícil de desterrar. Es bueno centrarse en buscar anomalías típicas, como variables u objetos no inicializados o que no se usan, ciclos infinitos y demás.

Los *recorridos* rinden mucho más. Son exposiciones del código escrito frente a pares. El programador, exponiendo su código, encuentra muchos errores. Además da ideas avanzadas a programadores nuevos que se lleva a recorrer.

Las llamadas *inspecciones de código* consisten en reuniones en conjunto entre los responsables de la programación y los responsables de la revisión.Tienen como objetivo revisar el código escrito por los programadores para chequear que cumpla con las normas que se hayan fijado y para verificar la eficiencia del mismo. Se realizan siguiendo el código de un pequeño porcentaje de módulos seleccionados al azar o según su grado de complejidad. Las inspecciones se pueden usar en sistemas grandes, pero con cuidado para no dar idea de estar evaluando al programador. Suelen servir porque los revisores están más acostumbrados a ver determinados tipos de errores comunes a todos los programadores. Además, después de una inspección a un programador, de la que surge un tipo de error, pueden volver a inspeccionar a otro para ver si no cayó en el mismo error.

El concepto de *extreme programming* propone programar de a dos, de modo que uno escribe y el otro observa el trabajo. Si el que está programando no puede avanzar en algún momento, sigue el que miraba. Y si ambos se traban pueden pedir ayuda a otro par. Esta no sólo es una forma más social de programación, sino que aprovecha las mismas ventajas de los recorridos e inspecciones de código, y puede prescindir de ellos.

Pruebas unitarias

Las pruebas unitarias se realizan para controlar el funcionamiento de pequeñas porciones de código como ser subprogramas (en la programación estructurada) o métodos (en POO). Generalmente son realizadas por los mismos programadores puesto que al conocer con

mayor detalle el código, se les simplifica la tarea de elaborar conjuntos de datos de prueba para testearlo.

Si bien una prueba exhaustiva sería impensada teniendo en cuenta recursos, plazos, etc, es posible y necesario elegir cuidadosamente los casos de prueba para recorrer tantos caminos lógicos como sea posible. Inclusive procediendo de esta manera, deberemos estar preparados para manejar un gran volumen de datos de prueba.

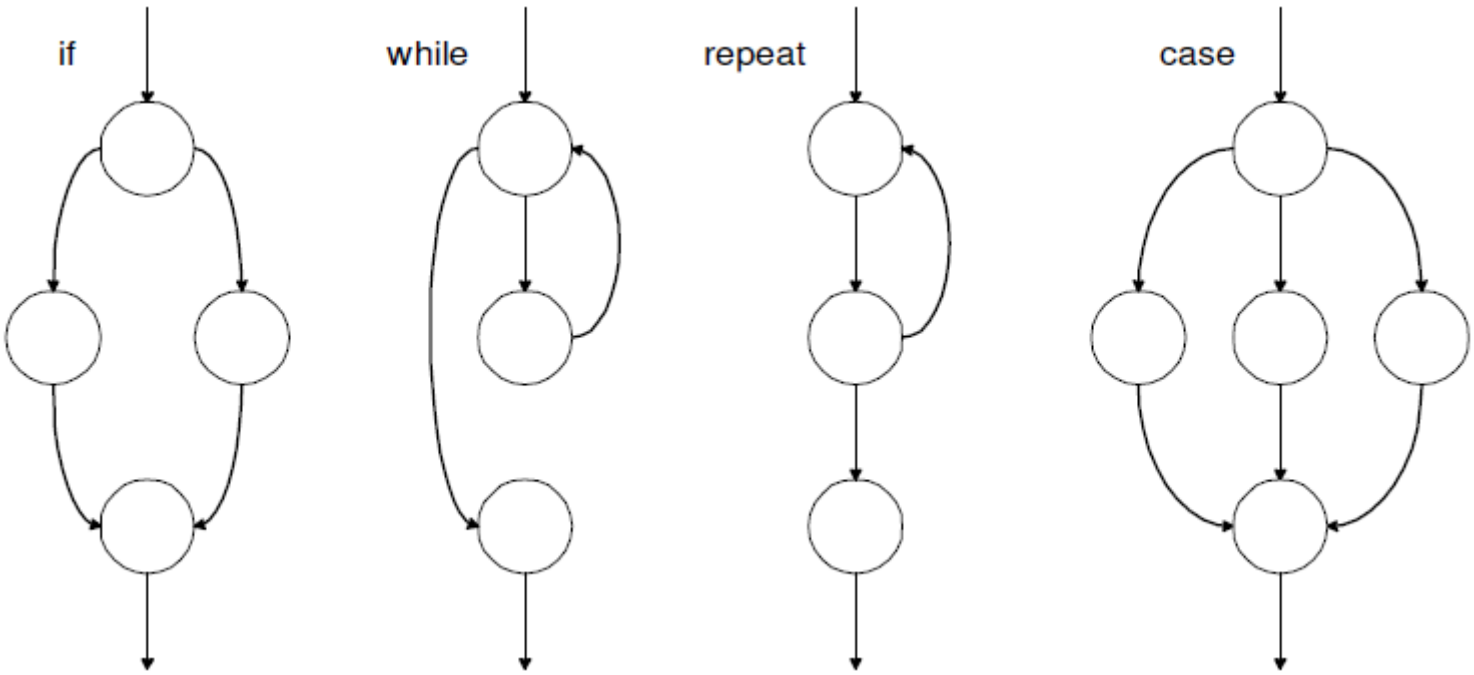
Los métodos de cobertura de caja blanca tratan de recorrer todos los caminos posibles por lo menos una vez, lo que no garantiza que no haya errores pero pretende encontrar la mayor parte.

El tipo de prueba a la cual se someterá a cada uno de los módulos dependerá de su complejidad. Recordemos que nuestro objetivo aquí es encontrar la mayor cantidad de errores posible. Si se pretende realizar una prueba estructurada, se puede confeccionar un grafo de flujo con la lógica del código a probar. De esta manera se podrán determinar todos los caminos por los que el hilo de ejecución pueda llegar a pasar, y por consecuente elaborar los juegos de valores de pruebas para aplicar al módulo, con mayor facilidad y seguridad.

Un grafo de flujo se compone de:

- Nodos (círculos), que representan una o más acciones del módulo.
- Aristas (flechas), que representan el flujo de control entre los distintos nodos.

Los nodos predicados son aquellos que contienen una condición, por lo que de ellos emergen dos o más aristas.

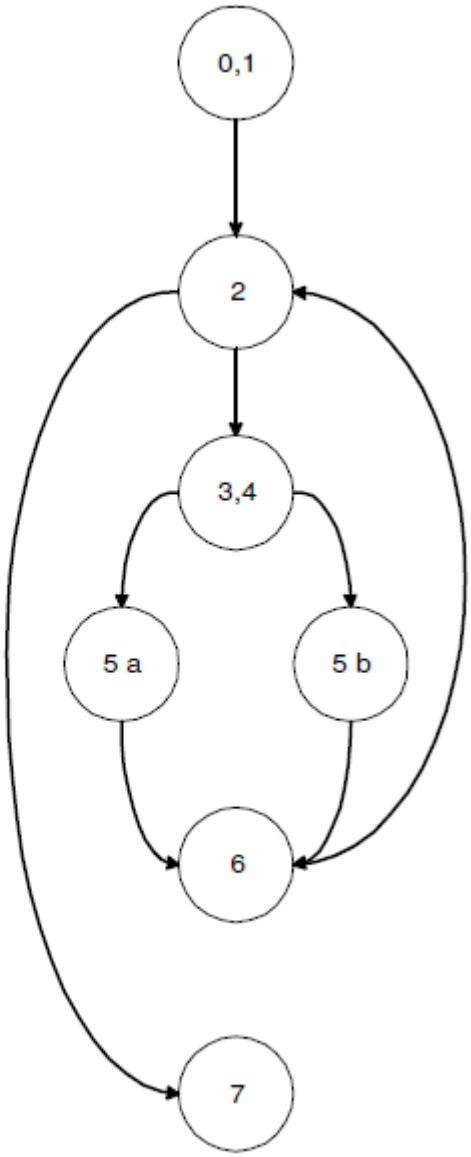


El paso de un diseño detallado o un pseudocódigo que representa una porción de programa a un grafo de flujo, requiere de las siguientes etapas:

- Señalar cada condición, tanto en sentencias *if* y *case* como en bucles *while* y *repeat*.
- Agrupar todas las secuencias siguiendo los esquemas de representación de construcciones.
- Numerar cada uno de los nodos resultantes de manera que consten de un identificador único. Las ramas de cada bifurcación pueden identificarse por el mismo número seguido de distintas letras.
- Dibujar en forma ordenada los nodos y sus aristas.

En el siguiente ejemplo, se muestra la manera de traducir un pequeño tramo de programa escrito en pseudocódigo a forma de grafo de flujo:

```
0:  Imprimir ("Ingrese 2 números")
1:  Leer (Num1 y Num2)
2:  Mientras (Num1 < Num2) hacer
3:    Num1 = Num1 + 1
4:    Si (Num1 es par) entonces
5a:       Imprimir (Num1" es par")
5b:  Si no   Imprimir (Num1" es impar")
6:    Fin Si
7:  Fin Mientras
```



Las pruebas unitarias tienen como objetivo verificar la funcionalidad y estructura de cada componente individualmente una vez que ha sido codificado.

Las pruebas unitarias constituyen la prueba inicial de un sistema y las demás pruebas deben apoyarse sobre ellas.

Existen dos enfoques principales para el diseño de casos de prueba:

- **Enfoque estructural** o de **caja blanca**. Se verifica la estructura interna del componente con independencia de la funcionalidad establecida para el mismo. Por tanto, no se comprueba la corrección de los resultados si éstos se producen. Ejemplos de este tipo de pruebas pueden ser ejecutar todas las instrucciones del programa, localizar código no usado, comprobar los caminos lógicos del programa, etc.
- **Enfoque funcional** o de **caja negra**. Se comprueba el correcto funcionamiento de los componentes del sistema de información, analizando las entradas y salidas y verificando que el resultado es el esperado. Se consideran exclusivamente las entradas y salidas del sistema sin preocuparse por la estructura interna del mismo.

El enfoque que suele adoptarse para una prueba unitaria está claramente orientado al diseño de casos de caja blanca, aunque se complemente con caja negra. El hecho de incorporar casos de caja blanca se debe, por una parte, a que el tamaño del componente

es apropiado para poder examinar toda la lógica y por otra, a que el tipo de defectos que se busca, coincide con los propios de la lógica detallada en los componentes.

Los pasos necesarios para llevar a cabo las pruebas unitarias son los siguientes:

- Ejecutar todos los casos de prueba asociados a cada verificación establecida en el plan de pruebas, registrando su resultado. Los casos de prueba deben contemplar tanto las condiciones válidas y esperadas como las inválidas e inesperadas.
- Corregir los errores o defectos encontrados y repetir las pruebas que lo detectaron. Si se considera necesario, debido a su implicación o importancia, se repetirán otros casos de prueba ya realizados con anterioridad.

La prueba unitaria se da por finalizada cuando se hayan realizado todas las verificaciones establecidas y no se encuentre ningún defecto, o bien se determine su suspensión.

Pruebas de integración

En el caso de las pruebas de integración y de sistema,dado que ya se han realizado las pruebas unitarias, se tomará a cada uno de los módulos unitarios como una caja negra.

Las pruebas de integración tienen como base las pruebas unitarias y consisten en una progresión ordenada de testeos para los cuales los distintos módulos van siendo ensamblados y probados hasta haber integrado el sistema completo. Si bien se realizan sobre módulos ya probados en forma individual, no es necesario que se terminen todas las pruebas unitarias para comenzar con las de integración. Dependiendo de la forma en que se organicen, se pueden realizar en paralelo a las unitarias.

El orden de integración de los módulos influye en:

- La forma de preparar los casos de prueba.
- Las herramientas a utilizar (módulos ficticios, muñones, impulsores o “stubs”).
- El orden para codificar y probar los módulos.
- El costo de preparar los casos.
- El costo de la depuración.

Tanto es así que se le debe prestar especial atención al proceso de elección del orden de integración que se emplee.

Existen principalmente dos tipos de integración:

- La *integración incremental*
- La integración *no incremental*.

La **integración incremental** consiste en combinar el conjunto de módulos ya probados (al principio será un conjunto vacío) con los siguientes módulos a probar. Luego se va incrementando progresivamente el número de módulos unidos hasta que se forma el sistema completo.

En la **integración no incremental** o **Big Bang** se combinan todos los módulos de una vez.

Para ambos tipos de integración se deberán preparar los datos de prueba junto con los resultados esperados. Esta tarea debe ser realizada por personas ajenas a la programación de los módulos. No es necesario que la lleven a cabo una vez codificados los módulos puesto que basta con conocer qué módulos compondrán el sistema y cuál será la interfaz entre ellos.

Si en algún momento de la prueba se detectara uno o más errores, se dejará constancia del hecho y se reenviarán los módulos afectados al responsable de la programación para que identifique la causa del problema y lo solucione. Luego se volverán a efectuar las

pruebas programadas y así sucesivamente hasta que el sistema entero esté integrado y sin errores.

Por el hecho de poder ser llevada a cabo por distintos caminos, la integración incremental brinda una mayor flexibilidad en el uso de recursos. Se puede integrar la estructura de módulos desde un extremo a otro y continuar hacia el extremo opuesto según distintas prioridades. La forma de llevar a cabo esta tarea dependerá de la naturaleza del sistema en cuestión, pero sea cual fuere el camino elegido, será de suma importancia una correcta planificación.

En la **integración incremental ascendente (De abajo arriba – bottom-up)** se comienza integrando primero los módulos de más bajo nivel. El proceso deberá seguir los siguientes pasos:

- Elegir los módulos de bajo nivel que se van a probar.
- Escribir un módulo impulsor para la entrada de datos de prueba a los módulos y para la visualización de los resultados.
- Probar la integración de los módulos.
- Eliminar los módulos impulsores y juntar los módulos ya probados con los módulos de niveles superiores, para continuar con las pruebas.

Estas tareas se pueden realizar en paralelo si es que se dispone de tres equipos de trabajo, o en serie de lo contrario. Para la prueba de cada uno de los módulos mencionados, es necesaria la preparación de un módulo impulsor. El objeto de los módulos impulsores es transmitir o impulsar los casos de prueba a los módulos testeados y recibir los resultados que estos produzcan en los casos en que sea necesario. Es decir, que deben simular las operaciones de llamada de los módulos jerárquicos superiores correspondientes. Estos módulos tienen que estar bien diseñados, para que no arrojen ni más ni menos errores que los que realmente pueden producirse. Al fin y al cabo, deben simular todas las situaciones que se van a producir en el sistema real.

La **integración incremental descendente (De arriba abajo – top-down)** parte del módulo de control principal (de mayor nivel) para luego ir incorporando los módulos subordinados progresivamente. No hay un procedimiento considerado óptimo para seleccionar el siguiente módulo a incorporar. La única regla es que el módulo incorporado tenga todos los módulos que lo invocan previamente probados.

En general no hay una secuencia óptima de integración. Debe estudiarse el problema concreto y de acuerdo a este, buscar el orden de integración más adecuado para la organización de las pruebas. No obstante, pueden considerarse las siguientes pautas:

- Si hay secciones críticas como ser un módulo complicado, se debe proyectar la secuencia de integración para incorporarlas lo antes posible.
- El orden de integración debe incorporar cuanto antes los módulos de entrada y salida.

Existen principalmente dos métodos para la incorporación de módulos:

- Primero en profundidad: primero se mueve verticalmente en la estructura de módulos.
- Primero en anchura: Primero se mueve horizontalmente en la estructura de módulos.

Etapas de la integración descendente:

- El módulo de mayor nivel hace de impulsor y se escriben módulos ficticios simulando a los subordinados, que serán llamados por el módulo de control superior.
- Probar cada vez que se incorpora un módulo nuevo al conjunto ya engarzado.
- Al terminar cada prueba, sustituir un módulo ficticio subordinado por el real que reemplazaba, según el orden elegido.

- Escribir los módulos ficticios subordinados que se necesiten para la prueba del nuevo módulo incorporado.

Los módulos ficticios subordinados se crean para permitir la prueba de los demás módulos. Pueden llevar a cabo una variedad de funciones, como por ejemplo:

- Mostrar un mensaje que demuestre que ese módulo fue alcanzado (“Módulo XX alcanzado”).
- Establecer una conversación con una terminal. De esta forma se puede permitir que la misma persona que realiza la prueba actúe de módulo subordinado.
- Devolver un valor constante, tabulado o elegido al azar.
- Ser una versión simplificada del módulo representado.
- Mostrar los datos recibidos.
- Ser un loop sin nada que hacer más que dejar pasar el tiempo.

Ventajas de la integración descendente:

- Las fallas que pudieran existir en los módulos superiores se detectan en una etapa temprana.
- Permite ver la estructura del sistema desde un principio, facilitando la elaboración de demostraciones de su funcionamiento.
- Concuerda con la necesidad de definir primero las interfaces de los distintos subsistemas para después seguir con las funciones específicas de cada uno por separado.

Desventajas de la integración descendente:

- Requiere mucho trabajo de desarrollo adicional ya que se deben escribir un gran número de módulos ficticios subordinados que no siempre son fáciles de realizar. Suelen ser más complicados de lo que aparentan.
- Antes de incorporar los módulos de entrada y salida resulta difícil introducir los casos de prueba y obtener los resultados.
- Los juegos de datos de prueba pueden resulta difíciles o imposibles de generar puesto que generalmente son los módulos de nivel inferior los que proporcionan los detalles.
- Induce a diferir la terminación de la prueba de ciertos módulos.

Ventajas de la integración incremental ascendente:

- Las entradas para las pruebas son más fáciles de crear ya que los módulos inferiores suelen tener funciones más específicas.
- Es más fácil la observación de los resultados de las pruebas puesto que es en los módulos inferiores donde se elaboran.
- Resuelve primero los errores de los módulos inferiores que son los que acostumbran tener el procesamiento más complejo, para luego nutrir de datos al resto del sistema.

Desventajas de la integración incremental ascendente:

- Se requieren módulos impulsores, que deben escribirse especialmente y que no son necesariamente sencillos de codificar.
- El sistema como entidad no existe hasta que se agrega el último módulo.

El método de **integración incremental sándwich (estrategia combinada)** combina facetas de los métodos ascendente y descendente. Consiste en integrar una parte del sistema en forma ascendente y la restante en forma descendente, provocando la unión de ambas partes en algún punto intermedio. La principal ventaja es que nos da mayor libertad para elegir el orden de integración de los módulos según las características

específicas del sistema en cuestión. De esta manera, podremos incluir y probar antes los módulos que consideremos críticos:

- Módulos dirigidos a múltiples propósitos.
- Módulos con mayor control (en general, los módulos de mayor nivel controlan a muchos otros módulos).
- Módulos con alto grado de complejidad.
- Módulos con requisitos de rendimiento muy definidos.

La **integración no incremental** puede ser beneficiosa par la prueba de sistema de pequeñísima envergadura cuya cantidad de módulos sea muy limitada y la interfaz entre los mismos clara y sencilla. Consiste en integrar todos los módulos del sistema a la vez e ingresar los valores de prueba para testear todas las interfaces.

La única ventaja es que no se necesita ningún tipo de trabajo adicional: ni planificar el orden de integración, ni crear módulos impulsores, ni crear módulos ficticios subordinados. Por otro lado, la lista de desventajas incluye:

- No se tiene noción de la comunicación de los módulos hasta el final.
- En ningún momento se dispone de un producto -siquiera parcial- para mostrar o presentar.
- El hecho de realizar todas las pruebas de una vez hace que las sesiones de prueba sean largas y tediosas.
- La cantidad de errores que arroje puede ser atemorizante.
- La tarea de encontrar la causa de los errores resulta mucho más compleja que con los métodos incrementales.
- Se corre el riesgo de que a poco tiempo de que se cumpla el plazo de entrega, haya que volver sobre el diseño y la codificación del sistema.

Pruebas de sistema

Son pruebas de integración del sistema de información completo, y permiten probar el sistema en su conjunto y con otros sistemas con los que se relaciona para verificar que las especificaciones funcionales y técnicas se cumplen. Dan una visión muy similar a su comportamiento en el entorno de producción.

Las pruebas de sistema se realizan una vez integrados todos los componentes. Su objetivo es ver la respuesta del sistema en su conjunto, frente a distintas situaciones. Se simulan varias alternativas que podrían darse con el sistema implantado y en base a ellas se prueba la eficacia y eficiencia de la respuesta que se obtiene.

Se pueden distinguir varios tipos de pruebas distintos, por ejemplo:

- **Pruebas negativas**: se trata de que el sistema falle para ver sus debilidades.
- **Pruebas funcionales**: dirigidas a asegurar que el sistema de información realiza correctamente todas las funciones que se han detallado en las especificaciones dadas por el usuario del sistema.
- **Pruebas de comunicaciones**: determinan que las interfaces entre los componentes del sistema funcionan adecuadamente, tanto a través de dispositivos remotos, como locales. Asimismo, se han de probar las interfaces hombre/máquina.
- **Pruebas de volumen**: consisten en examinar el funcionamiento del sistema cuando está trabajando con grandes volúmenes de datos, simulando las cargas de trabajo esperadas.
- **Pruebas de sobrecarga**: consisten en comprobar el funcionamiento del sistema en el umbral límite de los recursos, sometiéndole a cargas masivas. El objetivo es establecer los puntos extremos en los cuales el sistema empieza a operar por debajo de los requisitos establecidos.

- **Pruebas de disponibilidad de datos**: consisten en demostrar que el sistema puede recuperarse ante fallos, tanto de equipo físico como lógico, sin comprometer la integridad de los datos.
- **Pruebas de facilidad de uso**: consisten en comprobar la adaptabilidad del sistema a las necesidades de los usuarios, tanto para asegurar que se acomoda a su modo habitual de trabajo, como para determinar las facilidades que aporta al introducir datos en el sistema y obtener los resultados.
- **Pruebas de operación**: consisten en comprobar la correcta implementación de los procedimientos de operación, incluyendo la planificación y control de trabajos, arranque y rearranque del sistema, etc.
- **Pruebas de entorno**: consisten en verificar las interacciones del sistema con otros sistemas dentro del mismo entorno.
- **Pruebas de recuperación**: se simulan fallas de software y/o hardware para verificar la eficacia del proceso de recuperación.
- **Pruebas de rendimiento**: tiene como objeto evaluar el rendimiento del sistema integrado en condiciones de uso habitual. Consisten en determinar que los tiempos de respuesta están dentro de los intervalos establecidos en las especificaciones del sistema.
- **Pruebas de resistencia o de estrés**: comprueban el comportamiento del sistema ante situaciones donde se demanden cantidades extremas de recursos (número de transacciones simultáneas anormal, excesivo uso de las memorias, etc).
- **Pruebas de seguridad**: se utilizan para testear el esquema de seguridad intentando vulnerar los métodos utilizados para el control de accesos no autorizados al sistema. Consisten en verificar los mecanismos de control de acceso al sistema para evitar alteraciones indebidas en los datos.
- **Pruebas de instalación**: verifican que el sistema puede ser instalado satisfactoriamente en el equipo del cliente, incluyendo todas las plataformas y configuraciones de hardware necesarias.
- **Pruebas de compatibilidad**: se prueba al sistema en las diferentes configuraciones de hardware o de red y de plataformas de software que debe soportar.

Pruebas de implantación

El objetivo de las pruebas de implantación es comprobar el funcionamiento correcto del sistema integrado de hardware y software en el entorno de operación, y permitir al usuario que, desde el punto de vista de operación, realice la aceptación del sistema una vez instalado en su entorno real y en base al cumplimiento de los requisitos no funcionales especificados.

Una vez que hayan sido realizadas las pruebas del sistema en el entorno de desarrollo, se llevan a cabo las verificaciones necesarias para asegurar que el sistema funcionará correctamente en el entorno de operación. Debe comprobarse que responde satisfactoriamente a los requisitos de rendimiento, seguridad, operación y coexistencia con el resto de los sistemas de la instalación para conseguir la aceptación del usuario de operación.

Las pruebas de **seguridad** van dirigidas a verificar que los mecanismos de protección incorporados al sistema cumplen su objetivo; las de **rendimiento** a asegurar que el sistema responde satisfactoriamente en los márgenes establecidos en cuanto a tiempos de respuesta, de ejecución y de utilización de recursos, así como los volúmenes de espacio en disco y capacidad; por último con las pruebas de **operación** se comprueba que la planificación y control de trabajos del sistema se realiza de acuerdo a los procedimientos establecidos, considerando la gestión y control de las comunicaciones y asegurando la disponibilidad de los distintos recursos.

Asimismo, también son llevadas a cabo las pruebas de **gestión de copias de seguridad** y recuperación, con el objetivo de verificar que el sistema no ve comprometido su funcionamiento al existir un control y seguimiento de los procedimientos de salvaguarda y

de recuperación de la información, en caso de caídas en los servicios o en algunos de sus componentes. Para comprobar estos últimos, se provoca el fallo del sistema, verificando si la recuperación se lleva a cabo de forma apropiada. En el caso de realizarse de forma automática, se evalúa la inicialización, los mecanismos de recuperación del estado del sistema, los datos y todos aquellos recursos que se vean implicados.

Las verificaciones de las pruebas de implantación y las pruebas del sistema tienen muchos puntos en común al compartir algunas de las fuentes para su diseño como pueden ser los casos para probar el rendimiento (pruebas de sobrecarga o *stress*).

El responsable de implantación junto al equipo de desarrollo determina las verificaciones necesarias para realizar las pruebas así como los criterios de aceptación del sistema. Estas pruebas las realiza el equipo de operación, integrado por los técnicos de sistemas y de operación que han recibido previamente la formación necesaria para llevarlas a cabo.

Pruebas de aceptación

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario de dicho sistema que determine su aceptación, desde el punto de vista de su funcionalidad y rendimiento.

Las pruebas de aceptación, al igual que las de sistema, se realizan sobre el producto terminado e integrado; pero a diferencia de aquellas, están concebidas para que sea un usuario final quien detecte los posibles errores.

Se clasifican en dos tipos:

- Pruebas Alfa.
- Pruebas Beta.

Las **pruebas Alfa** se realizan por un cliente en un entorno controlado por el equipo de desarrollo. Para que tengan validez, se debe primero crear un ambiente con las mismas condiciones que se encontrarán en las instalaciones del cliente. Una vez logrado esto, se procede a realizar las pruebas y a documentar los resultados.

Cuando el software sea la adaptación de una versión previa, deberán probarse también los procesos de transformación de datos y actualización de archivos de todo tipo.

Las **pruebas Beta** se realizan en las instalaciones propias de los clientes. Para que tengan lugar, en primer término se deben distribuir copias del sistema para que cada cliente lo instale en sus oficinas, dependencias y/o sucursales, según sea el caso. Si se tratase de un número reducido de clientes el tema de la distribución de las copias no representa grandes dificultades, pero en el caso de productos de venta masiva, la elección de los *beta testers* debe realizarse con sumo cuidado. En el caso de las pruebas Beta ,cada usuario realizará sus propias pruebas y documentará los errores que encuentre, así como las sugerencias que crea conveniente realizar, para que el equipo de desarrollo tenga en cuenta al momento de analizar las posibles modificaciones.

Cuando el sistema tenga un cliente individual, las pruebas de aceptación se hacen de común acuerdo con éste, y los usuarios se determinan en forma programada, así como también se definen los aspectos a probar y la forma de informar resultados. Cuando, en cambio, se está desarrollando un producto masivo, los usuarios para pruebas de determinan de formas menos estrictas, y hay que ser muy cuidado en la evaluación del *feedback* que proveen. Por lo tanto, en este segundo caso hay que dedicar un esfuerzo considerable a la planificación de las pruebas de aceptación.

Pruebas de regresión

El objetivo de las pruebas de regresión es eliminar el efecto onda, es decir, comprobar que los cambios sobre un componente de un sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados.

Las pruebas de regresión se deben llevar a cabo cada vez que se hace un cambio en el sistema, tanto para corregir un error como para realizar una mejora. No es suficiente probar sólo los componentes modificados o añadidos, o las funciones que en ellos se realizan, sino que también es necesario controlar que las modificaciones no produzcan efectos negativos sobre el mismo u otros componentes.

Normalmente, este tipo de pruebas implica la repetición de las pruebas que ya se han realizado previamente, con el fin de asegurar que no se introducen errores que puedan comprometer el funcionamiento de otros componentes que no han sido modificados y confirmar que el sistema funciona correctamente una vez realizados los cambios.

Las pruebas de regresión pueden incluir:

- La repetición de los casos de pruebas que se han realizado anteriormente y están directamente relacionados con la parte del sistema modificada.
- La revisión de los procedimientos manuales preparados antes del cambio, para asegurar que permanecen correctamente.
- La obtención impresa del diccionario de datos de forma que se comprueba que los elementos de datos que han sufrido algún cambio son correctos.

El responsable de realizar las pruebas de regresión será el equipo de desarrollo junto al técnico de mantenimiento, quien a su vez, será responsable de especificar el plan de pruebas de regresión y de evaluar los resultados de dichas pruebas.

Relación ante los resultados de las pruebas

Las pruebas nos llevan a descubrir errores, que en la mayoría de los casos son de tipo funcional, es decir, del tipo: “el sistema debería hacer tal cosa y hace tal otra”.

En este apartado analizaremos nada más que los pasos a seguir cuando el error sólo es atribuible a la codificación.

Depuración

La depuración es la corrección de errores que sólo afectan a la programación, porque no provienen de errores previos en el análisis o en el diseño. A veces la depuración se hace luego de la entrega del sistema al cliente y es parte del mantenimiento.

En realidad, en las revisiones de código y las pruebas unitarias, encontrar un error es considerablemente más sencillo, ya que se hace con el código a mano. Aun cuando se hubiera optado por una prueba unitaria de caja negra,es sencillo recorrer el módulo que revela un comportamiento erróneo por dentro para determinar el lugar exacto del error. Existen incluso herramientas de depuración (*debugging*) de los propios ambientes de desarrollo que facilitan esta tarea, que incluso proveen recorrido paso a paso y examen de valores de datos. Y el lenguaje C traía una macro *assert* portable, que sencillamente abortaba un programa si una condición no se cumplía.

De todas maneras, es importante analizar correctamente si el error está donde parece estar o proviene de una falla oculta más atrás en el código. Para encontrar estos casos más complejos son útiles las herramientas de recorrida hacia atrás, que permiten partir del lugar donde se genera el error y recorrer paso a paso el programa en sentido inverso.

Las pruebas de integración, de sistema y de aceptación también pueden llevar a que sea necesaria una depuración, aunque aquí es más difícil encontrar el lugar exacto del error.

Por eso a menudo se utilizan *bitácoras* (*logs*), que nos permiten evaluar las condiciones que se fueron dando antes de un error mediante el análisis de un historial de uso del sistema que queda registrado en medios de almacenamiento permanente.

La depuración se hace en cuatro pasos:

- Reproducir el error.
- Diagnosticar la causa.
- Corregirla.
- Verificar la corrección.

Si el error no se repite al intentar reproducirlo es muy difícil hacer el diagnóstico. Como en casi todas las ciencias, se buscan causas y efectos, condiciones necesarias y suficientes para que se produzca el error. Luego hay que buscar el sector del código donde se produce el error, lo que nos lleva a las consideraciones hechas recientemente. La corrección del error entraña mucho riesgo, porque a menudo se introducen nuevos errores (hay quienes hablan de tasas de 0,2 a 0,5 nuevos errores por corrección). Y nunca hay que olvidarse de realizar una nueva verificación después de la corrección.

Reacción ante los errores en las pruebas de sistema y de aceptación

Hemos dicho ya que los errores que aparezcan en estos tipos de prueba van a llevar a la larga a una depuración, en la medida en que sean errores de codificación.

Para llegar a ello, no obstante, se requiere determinar el módulo donde se produjo el error. Esta tarea, en apariencia dificultosa, puede facilitarse considerablemente si trabajamos con un entorno de desarrollo que nos permita recorrer el código en modo de depuración sin necesidad de entrar en todos los módulos.

Revisión Formal

El objetivo de la revisión formal es detectar y registrar los defectos de un producto intermedio verificando que satisface sus especificaciones y que se ajusta a los estándares establecidos, señalando las posibles desviaciones.

Es un proceso de revisión riguroso en el que hay poca flexibilidad a la hora de llevarlo a cabo debido a que su objetivo es llegar a detectar lo antes posible, los posibles defectos o desviaciones en los productos que se van generando a lo largo del desarrollo. Esta característica fuerza a que se adopte esta práctica únicamente para productos que son de especial importancia, porque de otro modo podría frenar la marcha del proyecto.

En este proceso intervienen varias personas del grupo de aseguramiento de calidad, el equipo de desarrollo y según el tipo de revisión formal puede participar también el usuario.

El responsable del grupo de aseguramiento de calidad una vez que conoce los productos que se van a revisar formalmente, establece los grupos funcionales que van a llevar a cabo las revisiones, convocando a los participantes por adelantado, e informando del objetivo de la revisión, la agenda y las responsabilidades que tendrán asignadas en la revisión.

Es importante que en el transcurso de la revisión se sigan las directrices que estableció el responsable del grupo de aseguramiento de calidad, con el fin de que sea productiva y no se pierda tiempo en discusiones o ataques al responsable del producto.

Se concluye determinando las áreas de problemas y elaborando un informe de revisión formal y una lista de acciones correctivas que posee un carácter formal y vinculante.

Revisión Técnica

El objetivo de la revisión técnica es evaluar un producto intermedio del desarrollo para comprobar que se ajusta a sus especificaciones y que se está elaborando de acuerdo a las normas, estándares y guías aplicables al proyecto.

Con el fin de asegurar la calidad en el producto final del desarrollo, se deben llevar a cabo revisiones semiformales sobre los productos intermedios durante todo el ciclo de vida del software. Para ello, se fijan los objetivos de la revisión, la agenda que se podrá ir ajustando a lo largo del proyecto y el tipo de informe que se elaborará después de las revisiones.

Los participantes en una revisión técnica son el jefe de proyecto y el responsable del grupo de aseguramiento de calidad que, de forma conjunta, revisarán el producto que corresponda en cada momento.

Una vez fijado sobre qué productos intermedios se llevarán a cabo las revisiones, el responsable de aseguramiento de calidad recoge la información necesaria de cada producto para poder establecer los criterios de revisión y más adelante, evaluar si el producto cumple las especificaciones, es decir, si se ha elaborado de acuerdo a unas características concretas como pueden ser la aplicación de una técnica específica, la inclusión de algún tipo de información, etc. Además, se debe contar con la normativa y estándares aplicables al proyecto de forma que, no sólo se asegure que el producto cumpla sus especificaciones, sino también del modo adecuado.

Si se detecta alguna desviación en cuanto a sus especificaciones o a los estándares aplicados, y se considera que es necesario realizar alguna modificación, el responsable del grupo de aseguramiento de calidad elabora un informe con el que el jefe de proyecto tomará las medidas que estime convenientes. Con dichos informes de calidad, el jefe de proyecto irá confeccionando el dossier de aseguramiento de calidad, que formará parte de la documentación del proyecto al finalizar el desarrollo.

Bibliografía

- UBA (Universidad de Buenos Aires)
- PAe (Métrica 3)

La calidad del software y su medida. Modelos, métricas,normas y estándares.

Introducción a la Calidad del Software

Introducción

La **Calidad del Software** es *"la concordancia con los requerimientos funcionales y de rendimiento explícitamente establecidos con los estándares de desarrollo documentados y con las características implícitas que se esperan de todo software desarrollado profesionalmente"*. La Calidad del Software (CS) es una disciplina más dentro de la Ingeniería del Software. El principal instrumento para garantizar la calidad de las aplicaciones sigue siendo el Plan de Calidad, el cual se basa en normas o estándares genéricos y en procedimientos particulares. Los procedimientos pueden variar en cada organización, pero lo importante es que estén escritos, personalizados, adaptados a los procesos de la organización y que sean cumplidos.

Se puede decir que los requisitos del software son la base de las medidas de calidad y que la falta de concordancia con los requisitos es una falta de calidad. Los estándares o metodologías definen un conjunto de criterios de desarrollo que guían la forma en que se aplica la Ingeniería del Software. Si no se sigue ninguna metodología siempre habrá falta de calidad. Todas las metodologías y herramientas tienen un único fin, producir software de alta calidad.

A la hora de definir la calidad del software se debe diferenciar entre la calidad del Producto de software y la calidad del Proceso de desarrollo. No obstante, las metas que se establezcan para la calidad del producto van a determinar las metas a establecer para la calidad del proceso de desarrollo, ya que la calidad del producto va a estar en función de la calidad del proceso de desarrollo. Sin un buen proceso de desarrollo es casi imposible obtener un buen producto.

La calidad el producto de software se diferencia de la calidad de otros productos de fabricación industrial, ya que el software tiene ciertas características especiales:

1. El software es un producto mental, no restringido por las leyes de la Física o por los límites de los procesos de fabricación.
2. Se desarrolla, no se fabrica. El coste está fundamentalmente en el proceso de diseño, no en la producción. Y los errores se introducen también en el diseño, no en la producción.
3. El software no se deteriora con el tiempo. No es susceptible a los efectos del entorno, y su curva de fallos es muy diferente a la del hardware. Todos los problemas que surjan durante el mantenimiento estaban desde el principio, y afectan a todas las copias del mismo; no se generan nuevos errores.
4. Es artesanal en gran medida. El software, en su mayoría, se construye a medida, en vez de ser construido ensamblando componentes existentes y ya probados, lo que dificulta aún más el control de su calidad.
5. El mantenimiento del software es mucho más complejo que el mantenimiento del hardware. Cuando un componente de hardware se deteriora se sustituye por una pieza de repuesto, pero cada fallo en el software implica un error en el diseño o en el proceso mediante el cual se tradujo el diseño en código de máquina ejecutable.
6. Es engañosamente fácil realizar cambios sobre un software, pero los efectos de estos cambios se pueden propagar de forma explosiva e incontrolada.
7. El software con errores no se rechaza. Se asume que es inevitable que el software presente errores.

Es importante destacar que la calidad del software debe ser considerada en todos sus estados de evolución (especificaciones, diseño, código, etc). No basta con tener en cuenta la calidad del producto una vez finalizado, cuando los problemas de mala calidad ya no tienen solución o la solución es muy costosa.

La problemática general a la que se enfrenta el software es:

1. Aumento constante del tamaño y complejidad de los programas.
2. Carácter dinámico e iterativo a lo largo de su ciclo de vida, es decir que los programas de software a lo largo de su vida cambian o evolucionan de una versión a otra para mejorar las prestaciones con respecto a las anteriores.
3. Dificultad de conseguir productos totalmente depurados, ya que en ningún caso un programa será perfecto.
4. Se dedican elevados recursos monetarios a su mantenimiento, debido a la dificultad que los proyectos de software entrañan y a la no normalización a la hora de realizar los proyectos.
5. No suelen estar terminados en los plazos previstos, ni con los costes estipulados, ni cumpliendo los niveles deseables de los requisitos especificados por el usuario.
6. Incrementos constantes de los costes de desarrollo debido entre otros, a los bajos niveles de productividad.

7. Los clientes tienen una alta dependencia de sus proveedores por ser en muchos casos aplicaciones a “medida”.
8. Procesos artesanales de producción con escasez de herramientas.
9. Insuficientes procedimientos normalizados para estipular y evaluar la calidad, costes y productividad.

Uno de los principales problemas a los que nos enfrentamos a la hora de hablar de la calidad del software es el siguiente: ¿Es realmente posible encontrar un conjunto de propiedades en un software que nos den una indicación de su calidad? Para dar respuesta a esta pregunta aparecen los **Modelos de Calidad**. En los Modelos de Calidad, la calidad se define de forma jerárquica y tienen como objetivo resolver la complejidad mediante la descomposición.

La Calidad del Software debe implementarse en todo el ciclo de vida del mismo. Las distintas actividades para la implantación del control de calidad en el desarrollo de software son:

1. Aplicación y metodología y técnicas de desarrollo.
2. Reutilización de procesos de revisión formales.
3. Prueba del software.
4. Ajustes a los estándares de desarrollo.
5. Control de cambios, mediciones y recopilación de información.
6. Gestión de informes sobre el control de calidad.

La Calidad del Software es el conjunto de cualidades que lo caracterizan y que determinan su utilidad y existencia, la cual plantea un adecuado balanceo de eficiencia, confiabilidad, facilidad de mantenimiento, portabilidad, facilidad de uso, seguridad e integridad.

La implantación de un Modelo o Estándar requiere de una Gestión de la Calidad del Software. La Calidad se logra a través de la **Gestión de la Calidad**, la cual, según ISO 9000:2000, consiste en la realización de actividades coordinadas que permiten dirigir y controlar una organización en lo relativo a la calidad.

Gestión de la Calidad del Software

La Gestión de la Calidad del Software es una actividad esencial en cualquier empresa de software para asegurar la calidad de sus productos, y la competitividad frente a la oferta del mercado. Es un conjunto de actividades de la función general de la Dirección que determina la calidad, los objetivos y las responsabilidades. Se basa en la determinación y aplicación de las políticas de calidad de la empresa (objetivos y directrices generales). La Gestión o Administración de la Calidad se aplica normalmente a nivel empresa. También puede haber una gestión de la calidad dentro de la gestión de cada proyecto.

El propósito de la Administración de la Calidad del Software es, en primer lugar, entender las expectativas del cliente en términos de calidad y poner en práctica un plan proactivo para satisfacer esas expectativas. Dado que la calidad está definida por el cliente, podría parecer que es completamente subjetiva. De cualquier forma, hay muchas cosas acerca de la calidad que pueden hacerse objetivamente. Esto requiere examinar cada una de las características individuales del software y determinar una o más métricas que pueden recolectarse para reflejar dichas características. Por ejemplo, una característica de calidad puede ser que la solución tenga la menor cantidad de errores. Esta característica puede medirse contando los errores y defectos de la solución.

La Administración de la Calidad no es un evento, es un proceso y una forma de pensamiento. Un producto de software consistente, de alta calidad no puede producirse a partir de un proceso malo. Existe la necesidad de un ciclo constante de medir la calidad, actualizar el proceso, medir otra vez, actualizar, etc. Para hacer que la administración de calidad del software funcione, es vital recolectar métricas. Si no se capturan métricas será

difícil mejorar los procesos a partir de una iniciativa de administración de calidad. Uno de los propósitos de la administración de la calidad del software es encontrar errores y defectos en el proyecto tan pronto como sea posible. Entonces, un buen proceso de administración de calidad tomará más esfuerzo y costo. De cualquier manera, habrá una gran recompensa al tiempo que el proyecto avanza.

Por ejemplo, es mucho más fácil arreglar un problema con los requerimientos de negocio durante la fase de análisis que tener que arreglar problemas durante las pruebas. En otras palabras, el equipo de proyecto debe intentar mantener una alta calidad durante el proceso de desarrollo de los productos de software, en vez de esperar arreglar problemas durante las pruebas cercanas al final del proyecto (o en el peor de los casos, cuando el cliente encuentra el problema después que el proyecto se completó).

Desde el punto de vista de la calidad, la Gestión de la Calidad del Software está formada por 4 partes, las cuales son:

1. **Planificación** de la Calidad del Software.
2. **Control** de la Calidad del Software.
3. **Aseguramiento** de la Calidad del Software.
4. **Mejora** de la Calidad del Software.

Planificación de la Calidad del Software

Según la Norma ISO 9000:2000, la planificación de la calidad es la parte de la gestión de la calidad enfocada al establecimiento de los objetivos de la calidad y a la especificación de los procesos operativos necesarios y de los recursos relacionados para cumplir los objetivos de calidad.

La Planificación de la Calidad del Software es la parte de la Gestión de la Calidad encargada de realizar el proceso administrativo de desarrollar y mantener una relación entre los objetivos y recursos de la organización; y las oportunidades cambiantes del mercado. El objetivo es modelar y remodelar los negocios y productos de la empresa, de manera que se combinen para producir un desarrollo y utilidades satisfactorias.

Los aspectos a considerar en la Planificación de la Calidad del Software son:

- Modelos/Estándar de Calidad del Software a utilizar.
- Costos de la Calidad del Software.
- Recursos humanos y materiales necesarios.

Los factores que determinan el Modelo o Estándar de Calidad de Software a elegir son:

1. La complejidad del proceso de diseño.
2. La madurez del diseño.
3. La complejidad del proceso de producción.
4. Las características del producto o servicio.
5. La seguridad del producto o servicio.
6. Económico.

Según la Norma ISO/IEC 90003:2004 se puede decir que: “La planificación de la calidad facilita el modo de adaptar la planificación del sistema de gestión de la calidad a un proyecto especifico, producto o contrato. La planificación de la calidad puede incluir referencias genéricas y/o *proyecto/producto/contrato* específico de procedimientos, como apropiados. La planificación de la calidad debería ser revisada de nuevo junto con el progreso del diseño y desarrollo, y los elementos, en cada fase, deberían ser completamente definidos al comienzo de dicha fase.”.

La planificación de la calidad del software a nivel de proyectos debería considerar lo siguiente:

1. Inclusión de los planes de desarrollo.
2. Los requisitos de calidad relacionados con los productos y/o procesos.
3. Los sistemas de gestión de la calidad adaptando y/o identificando los procesos e instrucciones específicos, apropiados para el ámbito del manual de calidad y algunas exclusiones expuestas.
4. Los procesos de proyectos-específicos e instrucciones, tales como, especificación de pruebas del software detallando los planes, diseños, casos de pruebas y procesos para la unidad, integración, sistemas y pruebas de aceptación.
5. Los métodos, modelos, herramientas, convenios de lenguajes de programación, bibliotecas, marcos de trabajo y otros componentes reutilizables para ser usados en los proyectos.
6. Los criterios para el comienzo y el final de cada fase o etapa del proyecto.
7. Los tipos de análisis y otras verificaciones y actividades de validación para ser llevadas a cabo.
8. Los procesos de gestión de la configuración para ser llevados a cabo.
9. Las actividades de seguimiento y las medidas para ser llevadas a cabo.
10. Las personas responsables de aprobar de procesos de salida para su uso posterior.
11. La formación necesaria para el uso de herramientas y técnicas, y la organización de la formación previa a la habilidad necesaria.
12. Los registros para ser mantenidos.
13. La gestión de cambios, como por ejemplo, para recursos, escalas de tiempo y cambios de contrato.

La planificación de la calidad, sin embargo, abreviada es particularmente útil para limitar los objetivos de calidad para los software siendo designados para un propósito limitado.

Según Humphrey (1989) un plan de calidad puede tener la siguiente estructura:

1. Introducción al Producto: una descripción del producto, su objetivo en el mercado y expectativas de calidad del producto.
2. Planes del producto: Fechas críticas respecto de la liberación del producto y responsabilidades del producto respecto de su distribución y servicio.
3. Descripción del proceso: Procesos de desarrollo y servicios que serían usados en el desarrollo y en la administración.
4. Objetivos de Calidad: Objetivos y planes de calidad del producto, los cuales incluyen la identificación de los atributos de calidad del producto.
5. Manejo del riesgo: principales riesgos que pueden afectar la calidad del producto.

Esta información puede ser presentada en diferentes documentos.

El plan de calidad define los atributos de calidad más importantes del producto a ser desarrollado y define el proceso de evaluación de la calidad.

En la Planificación de la Calidad del Software se debe determinar:

1. Rol de la Planificación.
2. Requerimientos de la Calidad del Software.
3. Preparación de un Plan de Calidad del Software.
4. Implementación de un Plan de Calidad del Software.
5. Preparar un Manual de Calidad.

Control de la Calidad del Software

Según la Norma ISO 9000:2000, el control de la calidad es la parte de la gestión de la calidad orientada al cumplimiento de los requisitos de la calidad.

El Control de la Calidad del Software son las técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad, centradas en 2 objetivos fundamentales:

1. Mantener bajo control un proceso.
2. Eliminar las causas de los defectos en las diferentes fases del ciclo de vida.

Está formado por actividades que permiten evaluar la calidad de los productos de software desarrollados. El aspecto a considerar en el Control de la Calidad del Software es la “Prueba del Software”.

Las **Pruebas de Software** presentan una interesante anomalía para el Ingeniero del Software. Durante las fases anteriores de definición y de desarrollo, el Ingeniero intenta construir el software partiendo de un concepto abstracto y llegando a una implementación tangible. Luego, llegan las pruebas. El Ingeniero crea una serie de casos de prueba que intentar “demoler” el software construido. De hecho, las pruebas son uno de los pasos de la Ingeniería del Software que se puede ver como destructivo en lugar de constructivo.

Las Etapas del Desarrollo de Software son:

- Análisis
- Diseño
- Implementación
- Pruebas
- Mantenimiento

La prueba es el proceso de ejecutar un programa con intención de encontrar defectos. Es un proceso destructivo que determina el diseño de los casos de prueba y la asignación de responsabilidades.

La prueba exitosa es aquella que descubre defectos. El “caso de prueba bueno” es aquel que tiene alta probabilidad de detectar un defecto aún no descubierto. El “caso de prueba exitoso” es aquel que detecta un defecto aún no descubierto.

La prueba no es:

1. Demostración que no hay errores.
2. Demostración que el software desempeña correctamente sus funciones.
3. Establecimiento de confianza que un programa hace lo que debe hacer.

La prueba demuestra hasta qué punto las funciones del software parecen funcionar de acuerdo con las especificaciones y parecen alcanzarse los requisitos de rendimiento. Además, los datos que se van recogiendo a medida que se lleva a cabo la prueba proporcionan una buena indicación de la confiabilidad del software e indican la calidad del software como un todo. Pero, la prueba no puede asegurar la ausencia de defectos; sólo puede demostrar que existen defectos en el software.

La prueba del software es un concepto más amplio que, a menudo, es conocido como verificación y validación (V&V):

- La **verificación** se refiere al conjunto de actividades que aseguran que el software implementa correctamente una función específica. Su pregunta asociada es: ¿estamos construyendo el producto correctamente?
- La **validación** se refiere a un conjunto diferente de actividades que aseguran que el software construido se ajusta a los requisitos del cliente. Su pregunta asociada es: ¿estamos construyendo el producto correcto?. La validación del software se consigue mediante una serie de pruebas de caja negra que demuestran la conformidad respecto de los requisitos del cliente. Un plan de prueba traza las clases de pruebas

que se han de llevar a cabo, y un procedimiento de prueba define los casos de prueba específicos en un intento por descubrir errores de acuerdo con los requisitos.

Los principios básicos de la pruebas de software son:

1. A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente.
2. Las pruebas deberían planificarse mucho antes que empiecen.
3. Las pruebas deberían empezar por “lo pequeño” y progresar hacia “lo grande”.
4. Para ser más eficaces, las pruebas deberían ser realizadas por un equipo independiente.

Una estrategia de prueba de software integra las técnicas de diseño de casos de prueba en una serie de pasos bien planificados que dan como resultado una correcta construcción del software. La estrategia proporciona un mapa que describe los pasos que hay que llevar a cabo como parte de la prueba, cuándo se deben planificar y realizar esos pasos, y cuánto esfuerzo, tiempo y recursos se van a requerir. Cualquier estrategia de prueba debe incorporar la planificación de la prueba, el diseño de casos de prueba, la ejecución de las pruebas; y la agrupación y evaluación de los datos resultantes.

Las características generales de las estrategias de prueba de software son las siguientes:

1. La prueba comienza en el nivel módulo y trabaja “hacia fuera”, hacia la integración de todo el sistema basado en computadora.
2. Diferentes técnicas de prueba son apropiadas en diferentes momentos.
3. La prueba la realiza el que desarrolla el software y un grupo de prueba independiente.
4. La prueba y la depuración son actividades, pero la depuración se puede incluir en cualquier estrategia de prueba.

Para implementar con éxito una estrategia de prueba de software, se debe:

1. Especificar los requisitos del producto de manera cuantificable antes que comiencen las pruebas.
2. Especificar los objetivos de prueba de manera explícita.
3. Desarrollar un plan de prueba de haga hincapié en la prueba de ciclo rápido.
4. Construir un software robusto diseñado para probarse a sí mismo.
5. Usar revisiones de técnicas formales efectivas como filtro antes de la prueba.
6. Realizar revisiones técnicas formales para evaluar la estructura de la prueba y los propios casos de prueba.
7. Desarrollar un enfoque de mejora continua al proceso de prueba.

Existen 2 enfoques de estrategia de prueba de software: uno tradicional y otro relacionado al ambiente Cliente/Servidor.

- **Estrategia Tradicional**

Una estrategia Tradicional de prueba del software debe incluir pruebas de bajo nivel que verifiquen que todos los pequeños segmentos de código fuente se han implementado correctamente, así como pruebas de alto nivel que validen las principales funciones del sistema frente a los requisitos del cliente. Una estrategia proporciona un conjunto de hitos. Debido a que los pasos de la estrategia de prueba se dan cuando aumenta la presión de los plazos fijados, se debe poder medir el progreso y los problemas deben aparecer lo antes posible.

Inicialmente, la prueba se centra en cada módulo individualmente, asegurando que funciona adecuadamente como una unidad. La **prueba de unidad** hace un uso intensivo de las técnicas de prueba de caja blanca, ejercitando caminos específicos de la estructura de control del módulo para asegurar un alcance completo y una detección máxima de

errores. La prueba de unidad centra el proceso de verificación en la menor unidad del diseño del software: el componente de software o módulo. Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada hacia y desde la unidad de programa que está siendo probada. Se examinan las estructuras de datos locales para asegurar que los datos que se mantienen temporalmente conservan su integridad durante todos los pasos de ejecución del algoritmo. Se prueban las condiciones límite para asegurar que el módulo funciona correctamente en los límites establecidos. Se ejercitan todos los caminos independientes de la estructura de control con el el fin de asegurar que todas las sentencias del módulo se ejecutan por lo menos una vez. Y, finalmente, se prueban todos los caminos de manejo de errores. Antes de iniciar cualquier otra prueba es preciso probar el flujo de datos de la interfaz del módulo. Si los datos no entran correctamente, todas las demás pruebas no tienen sentido. Además de las estructuras de datos locales, durante la prueba de unidad se debe comprobar el impacto de los datos globales sobre el módulo.

A continuación, se deben ensamblar o integrar los módulos para formar el paquete de software completo. La **prueba de integración** es una técnica sistemática que permite construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo es juntar los módulos probados mediante la prueba de unidad y construir una estructura de programa que esté de acuerdo con lo que dicta el diseño. Se combinan todos las módulo por anticipado. Se prueba todo el programa en conjunto. Se encuentra un gran conjunto de errores. La corrección se hace difícil, ya que es complicado aislar las causas al tener el programa entero en toda su extensión. Una vez que se corrigen esos errores aparecen otros nuevos y el proceso continúa en lo que parece ser un ciclo sin fin.

Después que el software se ha integrado, se dirigen un conjunto de pruebas de alto nivel. Se deben comprobar los criterios de validación establecidos durante el análisis de requisitos. La **prueba de validación** proporciona una seguridad final que el software satisface todos los requisitos funcionales, de comportamiento y de rendimiento. Durante la validación se usan exclusivamente técnicas de prueba de caja negra.

El software, una vez validado, se debe combinar con otros elementos del sistema. La **prueba del sistema** verifica que cada elemento se ajusta de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total. La prueba del sistema está constituida por una serie de pruebas diferentes cuyo propósito primordial es ejercitar profundamente el sistema basado en computadora. Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se ha integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

La **prueba de regresión** es volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para asegurarse que los cambios no han propagado efectos colaterales no deseados. Este tipo de prueba es la actividad que ayuda a asegurar que los cambios no introduzcan un comportamiento no deseado o errores adicionales. A medida que progresa la prueba de regresión, el número de pruebas de regresión puede crecer demasiado. Por lo tanto, el conjunto de pruebas de regresión debería diseñarse para incluir sólo aquellas pruebas que traten una o más clases de errores en cada una de las funciones principales del programa. No es práctico ni eficiente volver a ejecutar cada prueba de cada función del problema después de un cambio.

Cuando se construye un software a medida para un cliente, se llevan a cabo una serie de pruebas de aceptación para permitir que el cliente valide todos los requisitos. Estas pruebas las realiza el usuario final en lugar del responsable del desarrollo de sistema. Una prueba de aceptación puede ir desde un informal paso de prueba hasta la ejecución sistemática de una serie de pruebas bien planificadas.

La **prueba alfa** la realiza el cliente en el lugar del área de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y los problemas de uso. Las pruebas alfa se realizan en un entorno controlado.

La **prueba beta** la realiza el usuario final del software en los lugares de trabajo de los clientes. La prueba beta es una aplicación en vivo del software en su entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador. Como resultado de los problemas informados durante la prueba beta, el desarrollador del software realiza modificaciones y prepara una versión del producto de software para toda clase de clientes.

- **Estrategia Cliente/Servidor**

La naturaleza distribuida de los sistemas Cliente/Servidor (C/S) plantea un conjunto de problemas específicos en la prueba del software, entre los cuales se pueden mencionar:

1. Consideraciones del GUI de cliente.
2. Consideraciones del entorno destino y de la diversidad de plataformas.
3. Consideraciones de bases de datos distribuidas.
4. Consideraciones de procesamiento distribuido.
5. Entornos destino que no son robustos.
6. Relaciones de rendimiento no lineales.

En general, las pruebas de software C/S se producen en 3 niveles:

1. Aplicaciones de cliente individuales: se prueban de modo desconectado (no se consideran el funcionamiento del servidor y de la red subyacente).
2. Aplicaciones de software de cliente y del servidor asociado: se prueban, pero no se ejercitan específicamente las operaciones de red.
3. Se prueba la arquitectura completa de C/S, incluyendo el rendimiento y funcionamiento de la red.

Los enfoques de pruebas para aplicaciones C/S son:

1. Pruebas de función de aplicación: Se prueba la funcionalidad de las aplicaciones cliente utilizando métodos. La aplicación se prueba en solitario en un intento de descubrir errores en su funcionamiento.
2. Pruebas de servidor: Se prueban la coordinación y las funciones de gestión de datos del servidor. Se considera el rendimiento del servidor (tiempo de respuesta y traspaso de datos en general).
3. Pruebas de bases de datos: Se prueba la precisión e integridad de los datos almacenados en el servidor. Se examinan las transacciones enviadas por las aplicaciones cliente para asegurar que los datos se almacenen, actualicen y recuperen adecuadamente. También se prueba el archivo de datos.
4. Pruebas de transacciones: Se crea una serie de pruebas adecuadas para comprobar que todas las clases de transacciones se procesen de acuerdo con los requisitos. Las transacciones hacen hincapié en la corrección de procesamiento y en los temas de rendimiento (tiempo de procesamiento de transacciones y comprobación de volúmenes de transacciones).
5. Pruebas de comunicaciones a través de la red: Estas prueban verifican que la comunicación entre los nodos de la red se produzca correctamente, y que el paso de mensajes, las transacciones y el tráfico de red tengan lugar sin errores. También se pueden efectuar pruebas de seguridad de red como parte de esta actividad de prueba.

Para llevar a cabo estos enfoques de prueba, se recomienda el desarrollo de perfiles operativos derivados de escenarios C/S. Un perfil operativo indica la forma en que los

distintos tipos de usuarios interactúan con el sistema C/S. Esto proporciona un patrón de uso que se puede aplicar cuando se diseñan y ejecutan las pruebas.

La estrategia para probar una arquitectura C/S comienza por comprobar una única aplicación de cliente. La integración de los clientes, del servidor y de la red se irá probando progresivamente. Finalmente, se prueba todo el sistema como entidad operativa. La integración de módulos en el desarrollo C/S puede tener algunos aspectos ascendentes o descendentes, pero la integración en proyectos C/S tiene más hacia el desarrollo paralelo y hacia la integración de módulos en todos los niveles de diseño.

La naturaleza multiplataforma en red de los sistemas C/S requiere que se preste más atención a la prueba de configuraciones y a la prueba de compatibilidades. La doctrina de prueba de configuraciones impone la prueba del sistema en todos los entornos conocidos de hardware y software en los cuales vaya a funcionar. La prueba de compatibilidad asegura una interfaz funcionalmente consistente entre plataformas de software y hardware.

Respecto de la organización de las pruebas del software, se puede decir que en cualquier proyecto de software existe un conflicto de intereses inherente que aparece cuando comienza la prueba. Se pide a la gente que ha construido el software que lo pruebe. Esto parece totalmente inofensivo; después de todo, ¿quién podría conocer mejor un programa que los que lo han desarrollado? El que desarrolla el software siempre es responsable de probar los módulos del programa, asegurándose que cada uno lleva a cabo la función para la que fue diseñada. En muchos casos se encargará de la prueba de integración, el paso de las pruebas que lleva a la construcción (y prueba) de la estructura total del sistema.

Las consideraciones para generar un plan de pruebas son:

1. Métodos de prueba.
2. Infraestructura (para desarrollo de software de prueba y ejecución de las pruebas).
3. Automatización de las pruebas.
4. Software de apoyo.
5. Administración de la configuración.
6. Riesgos.

Se requiere un plan global y uno detallado para cada actividad de prueba (pruebas unitarias, de integración, de facilidad de uso, funcionales, de sistema y de aceptación).

El formato del plan de pruebas es:

1. Propósito: Prescribir el ámbito, enfoque, recursos y plazos de las actividades de prueba.
2. Esquema: Identificador, Introducción, Items de prueba, Características a ser probadas, Características a no ser probadas, Enfoque, Criterios de aprobación por Item, Criterios de suspensión y requerimientos de continuación, Entrega de la prueba, Tareas de la prueba, Necesidades ambientales, Necesidades de personal y entrenamiento, Calendario, Riesgos y Contingencias, Aprobaciones.
3. Especificación de Prueba (complemento): Especificación de arquitectura de prueba, Implementación de las pruebas, Especificación de diseño de pruebas, Especificación de caso de prueba, Especificación de procedimiento de prueba, Ejecución de las pruebas, Evaluación de las pruebas.

El **diseño de casos de prueba** para el software o para otros productos de ingeniería puede requerir tanto esfuerzo como el propio diseño inicial del producto. Sin embargo, los Ingenieros de Software tratan las pruebas como algo sin importancia, desarrollando casos de prueba que “parezcan adecuados”, pero que tienen poca garantía de ser completos. Se deben diseñar pruebas que tengan la mayor probabilidad de encontrar el mayor número

de errores con la mínima cantidad de esfuerzo y tiempo posible. Cualquier producto de ingeniería puede probarse de una de estas 2 formas:

- **Prueba de caja negra**:
  Cuando se considera el software de computadora, la prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. Los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. Este tipo de prueba examina algunos aspectos del modelo fundamental del sistema sin tener mucho encuentra la estructura lógica interna del software. Los métodos de prueba de la caja negra se centran en los requisitos funcionales de software. La prueba de la caja negra permite al Ingeniero del Software obtener conjuntos de condicionales de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La prueba de la caja negra no es una alternativa a las técnicas de prueba de la caja blanca. La prueba de la caja negra intenta encontrar errores de las siguientes categorías:
  1) Funciones incorrectas o ausentes.
  2) Errores de interfaz.
  3) Errores en estructuras de datos o en acceso a bases de datos externas.
  4) Errores de rendimiento.
  5) Errores de inicialización y terminación.

- **Prueba de caja blanca**
  La prueba de caja blanca del software se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o mencionado. Para este tipo de prueba, se deben definir todos los caminos lógicos y desarrollar casos de prueba que ejerciten la lógica del programa. La prueba de caja blanca no se debe desechar como impracticable. Se pueden elegir y ejercitar una serie de caminos lógicos importantes. Se pueden comprobar las estructuras de datos para verificar su validez. Se pueden combinar los atributos de la prueba de caja blanca y de caja negra, para llegar a un método que valide la interfaz del software y asegure el correcto funcionamiento interno del software. La prueba de caja blanca, denominada prueba de caja de cristal, es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba.

Una “buena prueba” tiene los siguientes atributos:

1. Una alta probabilidad de encontrar un error – Para alcanzar este objetivo, el responsable de la prueba debe entender el software e intentar desarrollar una imagen mental de cómo podría fallar el software.
2. No debe ser redundante – El tiempo y los recursos para las pruebas son limitados. No hay motivo para realizar una prueba que tiene el mismo propósito que otra. Todas las pruebas deberían tener un propósito diferente.
3. Debería ser la mejor de la cosecha – En un grupo de pruebas que tienen un propósito similar, las limitaciones de tiempo y recursos pueden abogar por la ejecución de sólo un subconjunto de estas pruebas. En tales casos, se debería emplear la prueba que tenga la más alta probabilidad de descubrir una clase entera de errores.
4. No debería ser ni demasiado sencilla ni demasiado compleja – Es posible combinar una serie de pruebas en un caso de prueba, los posibles efectos secundarios de este enfoque puede enmascarar errores. En general, cada prueba debería realizarse separadamente.

El uso de herramientas de prueba puede hacer la prueba más fácil, efectiva y productiva.

Existen distintos tipos de herramientas por actividad:

1. Para revisiones e inspecciones: Análisis de complejidad, comprensión de código, análisis sintáctico y semántico.
2. Para planificación de pruebas: Modelos (templates) para documentación de planes de prueba, estimación de esfuerzo y calendarización de pruebas, analizador de complejidad.
3. Para diseño y desarrollo de pruebas: Generador de casos de prueba, diseño de prueba basado en requerimientos, captura y análisis de cobertura.
4. Para ejecución de pruebas: Captura, análisis de cobertura, pruebas de memoria, administración de los casos de prueba, simuladores y rendimiento.
5. Para soporte: Administración de problemas, administración de la configuración.

Aseguramiento de la Calidad del Software

Según la Norma ISO 9000:2000, el aseguramiento de la calidad es la parte de la gestión de la calidad orientada a proporcionar confianza en que se cumplirán los requisitos de calidad.

El Aseguramiento de Calidad del Software es el conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza que el software satisfará los requisitos dados de calidad. Este aseguramiento se diseña para cada aplicación antes de comenzar a desarrollarla y no después. El aseguramiento de la calidad del software engloba:

1. Un enfoque de gestión de calidad.
2. Métodos y herramientas de Ingeniería del Software.
3. Revisiones técnicas formales aplicables en el proceso de software.
4. Una estrategia de prueba multiescala.
5. Procedimientos para ajustarse a los estándares de desarrollo del software.
6. Mecanismos de medición y de generación de informes.

Las revisiones del software son un “filtro” para el proceso de Ingeniería del Software. Esto es, las revisiones se aplican a varios momentos del desarrollo del software y sirven para detectar errores y defectos que pueden ser eliminados. Las revisiones del software sirven para “purificar” las actividades de la Ingeniería del Software que suceden como resultado del análisis, diseño y codificación.

La revisión técnica formal (RTF), a veces llamada inspección, es el filtro más efectivo desde el punto de vista del aseguramiento de la calidad y es un medio efectivo para mejorar la calidad del software.

El defecto se define como una anomalía del producto. Dentro del contexto del proceso del software, los términos defecto y fallo son sinónimos. Ambos implican un problema de calidad que es descubierto después de entregar el software a los usuarios finales.

El objetivo principal delas RTF es encontrar errores durante el proceso, de forma que se conviertan en defectos después de la entrega del software. El beneficio de estas RTF es el descubrimiento de errores al principio para que no se propaguen al paso siguiente del proceso de software.

Los objetivos de la RTF son:

1. Descubrir errores en la función, la lógica o la implementación de cualquier representación del software.
2. Verificar que el software bajo revisión alcanza sus requisitos.
3. Garantizar que el software ha sido representado de acuerdo con ciertos estándares predefinidos.
4. Conseguir un software desarrollado en forma uniforme.

5. Hacer que los proyectos sean más manejables.

El aseguramiento de calidad se refiere a validar los procesos usados para crear los productos. Es una herramienta especialmente útil para administradores y patrocinadores, ya que permite discutir los procesos usados para crear los productos para determinar si son razonables. Este aseguramiento tiene asociado 2 constitutivos diferentes: los Ingenieros de Software que realizan el trabajo técnico y un grupo de SQA (Software Quality Assurance) que tiene la responsabilidad de la planificación de aseguramiento de la calidad, supervisión, mantenimiento de registros, análisis e informes.

Las Actividades del SQA son:

1. Establecimiento de un plan de SQA para un proyecto.
2. Participación en el desarrollo de la descripción del proceso de software del proyecto.
3. Revisión de las actividades de Ingeniería del Software para verificar su ajuste al proceso de software definido.
4. Auditoría de los productos de software designados para verificar el ajuste con los definidos como parte del proceso del software.
5. Asegurar que las desviaciones del trabajo y los productos del software se documentan y se manejan de acuerdo con un procedimiento establecido.
6. Registrar lo que no se ajuste a los requisitos e informar a sus superiores.

Además de estas actividades, el grupo de SQA coordina el control y la gestión de cambios y; ayuda a recopilar y analizar las **métricas del software**.

Las métricas son escalas de unidades sobre las cuales puede medirse un atributo cuantificable. Cuando se habla de software nos referimos a la disciplina de recopilar y analizar datos basándonos en mediciones reales de software, así como a las escalas de medición. Los atributos son características observables del producto o del proceso de software, que proporciona alguna información útil sobre el estado del producto o sobre el progreso del proyecto. El término producto se utiliza para referirse a las especificaciones, a los diseños y a los listados del código. Los valores de las métricas no se obtienen sólo por mediciones. Algunos valores de métricas se derivan de los requisitos del cliente o de los usuarios y, por lo tanto, actúan como restricciones dentro del proyecto.

Los principios básicos de la medición son:

1. Los objetivos de la medición deberían establecerse antes de empezar la recopilación de datos.
2. Todas las técnicas sobre métricas deberían definirse sin ambigüedades.
3. Las métricas deberían obtenerse basándose en una teoría válida para el dominio de aplicación.
4. Hay que hacer las métricas a medida para acomodar mejor los productos y procesos específicos.
5. Siempre que sea posible, la recopilación de datos y el análisis debería automatizarse.
6. Se deberían aplicar técnicas estadísticas válidas para establecer las relaciones entre los atributos internos del producto y las características externas de la calidad.

Las Razones que justifican la Medición del Software son:

1. Para indicar la calidad del producto.
2. Para evaluar la productividad de la gente que desarrolla el producto.
3. Para evaluar los beneficios (en términos de productividad y calidad) derivados del uso de nuevos métodos y herramientas de Ingeniería de Software.
4. Para establecer una línea base de estimación.
5. Para ayudar a justificar el uso de nuevas herramientas o formación adicional.

Las actividades del proceso de medición son:

1. Formulación: Obtención de medidas y métricas apropiadas para la representación del software.
2. Colección: Mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
3. Análisis: Cálculo de las métricas y aplicación de herramientas matemáticas.
4. Interpretación: La evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la presentación.
5. Retroalimentación: Recomendaciones obtenidas de la interpretación de métricas y técnicas transmitidas al equipo de desarrollo de software.

Los sistemas métricos necesitan tres tipos de valores:

1. Objetivos: Se basan habitualmente en consideraciones comerciales.
2. Predicciones: Indican la viabilidad de los objetivos. Se basan en las características del producto con el que tratamos.
3. Valores Reales: Pueden ser comparados con los objetivos para supervisar la progresión del proyecto. Son mediciones discretas de los atributos del software. Es preferible utilizar mediciones objetivas basadas en reglas. Algunas mediciones se basan en estimaciones donde un valor más que medirse se evalúa.

Las medidas de Calidad del Software deben comenzar desde la especificación y terminar con la implementación, implantación y mantenimiento o post-implantación. Debe aplicarse a lo largo de todo el proceso de Ingeniería de Software. Básicamente, la medición es una fase normal de cualquier actividad industrial. Sin mediciones es imposible perseguir objetivos comerciales normales de una manera racional.

Existen métricas a nivel Proyecto, Proceso y Producto respectivamente.

Métricas de Proyecto

Las métricas de Proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software. El uso de métricas para el Proyecto tiene 2 aspectos fundamentales:

1. Minimizar la planificación del desarrollo haciendo los ajustes necesarios que eviten retrasos y reducir problemas/riesgos potenciales.
2. Evaluar la calidad de los productos en el momento actual y cuando sea necesario, modificando el enfoque técnico que mejore la calidad.

Los indicadores de proyecto permiten al gestor de proyectos de software:

1. Evaluar el estado del proyecto.
2. Seguir la pista de los riesgos potenciales.
3. Detectar las áreas de problemas antes de que se conviertan en “críticas”.
4. Ajustar el flujo y las tareas del trabajo.
5. Evaluar la habilidad del equipo del proyecto en controlar la calidad de los productos de trabajo del software.

Métricas de Proceso

Las métricas del Proceso se recopilan de todos los proyectos y durante un largo período de tiempo. Su intento es proporcionar indicadores que lleven a mejorar los procesos de software a largo plazo. Se tendrán métricas asociadas a cada proceso del software (p.e. métricas de implementación). Estos indicadores de proceso permiten que una organización de Ingeniería de Software pueda tener una visión más profunda de la eficacia de un proceso ya existente y permiten que los gestores evalúen lo que funciona y lo que no.

En realidad, las medidas que recopila un equipo de proyecto y las convierte en métricas para utilizarse durante un proyecto, también pueden trasmitirse a los que tienen la responsabilidad de mejorar el proceso de software. Por esta razón, se utilizan muchas de las mismas métricas tanto en el dominio del proceso como en el del proyecto.

La única forma racional de mejorar cualquier proceso es medir atributos del proceso, desarrollar un juego de métricas significativas según estos atributos y utilizar las métricas para proporcionar indicadores que conducirán a una estrategia de mejora.

Las métricas del proceso se caracterizan por:

1. El control y ejecución del proyecto.
2. Medición de tiempos del análisis, diseño, implementación, implantación y post-implantación.
3. Medición de las pruebas (errores, cubrimiento, resultado en número de defectos y número de éxito).
4. Medición de la transformación o evolución del producto.

Métricas de Producto

Las métricas de Producto son privadas para un individuo y a menudo se combinan para desarrollar métricas del proyecto que sean públicas para un equipo de software. Están enfocadas a predecir y controlar:

1. El tamaño (líneas de código, bytes de código, operadores y operandos).
2. La estructura (control de flujo, relación entre componentes, cohesión y acoplamiento).
3. La complejidad (combinación de tamaño y estructura).
4. Los índices para controlar la documentación.
5. La calidad (independencia, completo, entendible, aumentado).
6. La estabilidad (los cambios aumentan el número de fallas, los cambios se pueden dar por definición de requerimientos o por cambios del entorno).

Las métricas del software deberían tener las siguientes características:

1. Simple y fácil de calcular.
2. Empírica e intuitivamente persuasiva: Debe satisfacer las nociones intuitivas del desarrollador sobre el atributo del producto en evaluación.
3. Consistente y objetiva: Presentar resultados sin ambigüedad.
4. Consistente en el empleo de unidades y tamaños: Deben emplearse medidas que no conduzcan a extrañas combinaciones de unidades.
5. Independiente del lenguaje de programación.
6. Mecanismo para retroalimentación de calidad: Debe proporcionar información para obtener un producto final de mayor calidad.

Las métricas a recabar dependen de los objetivos del negocio en particular. Los desarrolladores tienen a la vez objetivos comunes como, respetar el presupuesto y respetar los plazos, minimizar las tasas de defectos antes y después de la entrega del producto e intentar mejorar la calidad y la productividad. Las métricas deben ayudar a la evaluación de las representaciones del modelo lógico y físico, deben tener la capacidad de intuir sobre la complejidad del diseño y construcción; y deben ayudar en el diseño de casos de prueba.

Mejora en la Calidad del Software

Según la Norma ISO 9000:2000, la mejora de la calidad es la parte de la gestión de la calidad orientada a aumentar la capacidad de cumplir con los requisitos de la calidad. Los

requisitos pueden estar relacionados con cualquier aspecto tal como la eficacia, la eficiencia o la trazabilidad.

La Mejora de la Calidad del Software es la parte de la Gestión de la Calidad que contribuye, por medio de las mediciones, a los análisis de los datos y auditorias, a efectuar mejoras en la calidad del software.

Una Auditoria de Calidad tiene como objetivo mostrar la situación real para aportar confianza y destacar las áreas que pueden afectar adversamente esa confianza. Otro objetivo consiste en suministrar una evaluación objetiva de los productos y procesos para corroborar la conformidad con los estándares, las guías, las especificaciones y los procedimientos.

Las razones para realizar una auditoria son:

1. Establecer el estado del proyecto.
2. Verificar la capacidad de realizar o continuar un trabajo específico.
3. Verificar qué elementos aplicables del programa o Plan de Aseguramiento de la Calidad han sido desarrollados y documentados.
4. Verificar la adherencia de esos elementos con el programa o Plan de Aseguramiento de la Calidad.

El propósito y la actividad de la auditoria es recoger, examinar y analizar la información necesaria para tomar las decisiones de aprobación. La auditoria es realizada de acuerdo con los planes y procedimientos documentados. El plan de auditoria establece un procedimiento para dirigir la auditoria y para las acciones de seguimiento sobre las recomendaciones de la auditoria. Al realizar la auditoria, el personal de la misma evalúa los elementos del software y los procesos para contrastarlos con los objetivos y criterios de las auditorias, tales como contratos, requerimientos, planes, especificaciones o procedimientos, guías y estándares.

Los resultados de la auditoria son documentados y remitidos al director de la organización auditada, a la entidad auditora, y cualquier organización externa identificada en el plan de auditoria. El informe incluye la lista de elementos no conformes u otros aspectos para las posteriores revisiones y acciones. Cuando se realiza el plan de auditoria, las recomendaciones son informadas e incluidas en los resultados de la auditoria.

La auditoria puede traer como consecuencia la Certificación. Dicho Proceso de Certificación comienza con la emisión de una Solicitud de Certificación y culmina con la Concesión del Certificado. Un sistema de certificación de calidad permite una valoración independiente que debe demostrar que la organización es capaz de desarrollar productos y servicios de calidad.

En un software se tienen las siguientes visiones de la calidad:

1. Necesaria o Requerida: La que quiere el cliente.
2. Programada o Especificada: La que se ha especificado explícitamente y se intenta conseguir.
3. Realizada: La que se ha conseguido.

El objetivo es conseguir que las tres visiones coincidan. A la intersección entre la calidad Requerida y la calidad Realizada se le llama Calidad Percibida, y es la única que el cliente valora. Toda aquella calidad que se realiza pero no se necesita es un gasto inútil de tiempo y dinero.

La calidad, como sistema de gestión de una organización, necesita definir estos procesos y medirlos, para poder gestionarlos, es decir, para tener la capacidad de proponer mejoras y reconocerlas.

Para implementar un programa de mejoras es necesario definir procesos, decidir qué se quiere mejorar, definir qué medidas serán necesarias recoger, cómo y dónde tomarlas, gestionarlas mediante herramientas, utilizarlas para la toma de decisiones y reconocer las mejoras. Cuando el proceso a mejorar es el de desarrollo del software, es importante definir qué objetivos se quieren alcanzar, para reducir el número de medidas y, en consecuencia, el coste de recopilarlas y el impacto sobre la actividad de producción de software.

La calidad ha dejado de ser un tópico y es necesario que forme parte de los productos o servicios que comercializamos para nuestros clientes. El cliente es el mejor auditor de la calidad, él exige el nivel que está dispuesto a pagar por ella, pero no más. Por tanto, debemos de cuantificar cuál es el nivel de calidad que nos exige para poder planificar la calidad de los productos que se generen a lo largo de la producción del producto o servicio final. Al analizar las necesidades de nuestros clientes, deberemos tener en cuenta la previsible evolución de sus necesidades y tendencias en cuanto a características. Deberemos tener en cuenta la evolución tecnológica del entorno del producción de nuestros productos para suministrarlos con el nivel tecnológico adecuado. No debemos olvidar el nivel de calidad de nuestros competidores, debiendo elaborar productos cuyas características y funcionalidades sean competitivas con las de nuestros competidores.

La Calidad del Software es resultado del movimiento global dentro del proceso de mejoramiento continuo de los modelos y/o estándares de producción en todos los sectores industriales, en particular, cuando éste se concentra en la producción de sistemas de información y software especializado.

Calidad de los Datos

Dado que la calidad tiene componentes objetivos y subjetivos es necesario catalogar los requisitos de calidad de datos de los usuarios según unas determinadas dimensiones de calidad. Se intenta definir el concepto de calidad de datos y catalogar las dimensiones de calidad en función de unos determinados criterios, como pueden ser el ciclo de vida de los datos o los tipos de investigación realizadas o simplemente la forma en la que se usan los datos. Pero todos están de acuerdo en que la calidad de datos es un concepto multidimensional que comprende distintos aspectos según las necesidades de los consumidores de datos o de los diseñadores de sistemas, y que se justifica por el hecho de la concepción de calidad que aporta ISO.

Algunas de las dimensiones de calidad de datos más importantes son:

Dimensión	Definición
Facilidad de acceso	Los datos están disponibles o bien son fácil o rápidamente recuperables.
Cantidad apropiada de datos	El volumen de datos es adecuado para la tarea que se está realizando.
Compleción	Los datos son completos y suficientes para la tarea que se está desarrollando.
Facilidad de comprensión	Los datos son fácilmente comprensibles.
Credibilidad	Los datos pueden ser considerados como creíbles y verdaderos.
Disponibilidad temporal	Los datos están lo suficientemente actualizados para la tarea que se está desarrollando.
Facilidad de manipulación	Los datos son fácilmente aplicables y manipulables en diferentes tareas.
Facilidad de interpretación	Los datos están representados en el idioma apropiado, con una simbología correcta y adecuada y con la definición apropiada.
Libres de error	Los datos son correctos y fiables
Objetividad	Los datos son imparciales, sin prejuicios y connotaciones.
Relevancia	Los datos son útiles y aplicables en la tarea que se está desarrollando.
Representación concisa	Los datos están representados de una forma compacta.
Representación consistente	Todos los datos se representan en el mismo formato, que además es el más adecuado para la tarea que se está desarrollando.
Reputación	Los datos están altamente relacionados en términos de sus fuentes o contenidos.
Seguridad	El acceso a los datos está restringido apropiadamente para garantizar su seguridad
Valor añadido	Los datos son beneficiosos y ofrecen ventajas al usarlos.

Existe una metodología para la medición de la calidad de los datos guardados en un almacén de datos. Se parte de la idea de almacenar información referente a la calidad de los datos en el mismo almacén de datos, dicha metodología propone una serie de pasos bien estructurados y definidos que, partiendo de los requisitos de calidad de datos de los usuarios, trata de identificar las dimensiones de calidad de datos de los usuarios, trata de identificar las dimensiones de calidad que mejor describen esos requisitos, para después obtener métricas a partir de ese conjunto de dimensiones; después se realiza el proceso de medición propiamente dicho, que consiste en generar un valor numérico como resultado de un juicio de un determinado valor del dato con respecto a la dimensión elegida; posteriormente los resultados se guardan en el mismo almacén de datos, para después analizar los resultados. La forma de guardar los datos depende fuertemente del modelo de datos elegido para el almacén de datos.

Minería de datos. Aplicación a la resolución de problemas de gestión. Tecnología y algoritmos. Procesamiento analítico en línea (OLAP). Big data. Bases de datos NoSQL.

La minería de datos: Data Mining

El término minería o recopilación de datos (data mining) hace referencia al proceso de análisis semiautomático de BD de gran tamaño para hallar estructuras útiles. Al igual que la búsqueda de conocimiento en la inteligencia artificial o el análisis estadístico, la minería de datos intenta descubrir reglas y estructuras a partir de los datos. Es decir, la minería de datos trata de la búsqueda del conocimiento en las BD.

Los almacenes de datos guardan todos los datos relevantes para una organización, estando estructurados para que se pueda extraer información a partir de dichos datos. En este tema vamos a ver como la minería de datos permite sacar el máximo provecho del almacén de datos, ofreciendo una serie de técnicas y herramientas que automatizan (o semiautomatizan) el proceso de extracción de información y significado a partir de los datos que éste contiene.

El nombre de minería de datos (data mining) deriva de las similitudes entre buscar valiosa información de negocios en grandes BD y minar una montaña para encontrar una veta de metales preciosos. Ambos procesos requieren examinar una inmensa cantidad de material, o investigar inteligentemente hasta encontrar exactamente dónde residen los valores.

El proceso de descubrimiento de conocimiento en Bases de Datos

El descubrimiento de conocimiento en las BD es el proceso no trivial de identificación de patrones válidos, potencialmente útiles y comprensibles en los datos. El objetivo es la extracción de conocimiento de los datos, en el contexto de las BD de gran tamaño.

El proceso es iterativo, consta de unos pasos básicos e involucra decisiones por parte del usuario, siendo interactivo. Esto quiere decir que el proceso requiere el entendimiento del

dominio de la aplicación por parte del usuario. Se han identificado los siguientes pasos como componentes del proceso:

- Selección de un conjunto de datos objetivo.
- Preprocesamiento y limpieza de los datos.
- Transformación y reducción en la dimensión de los datos.
- Selección del método de minería de datos y de la técnica (algoritmo) de minería de datos e implementación de la técnica para realizar la extracción de patrones.
- Interpretación o evaluación de los patrones extraídos.
- Consolidación del conocimiento descubierto.

Por otro lado, el consorcio Cross-Industry Standard Process for Data Mining propuso un modelo estándar y de acceso público del proceso. El modelo es jerárquico y consta de cuatro niveles de abstracción:

- El primer nivel está constituido por una serie de fases las cuales se dividen en tareas generales.
- El segundo nivel se conoce como genérico, ya que, trata de cubrir todas las posibles situaciones de minería de datos.
- El tercer nivel es más especializado, describiendo particularmente qué acciones deben llevarse a cabo dependiendo de situaciones específicas.
- El cuarto nivel es la instanciación del proceso, como un registro de acciones, decisiones y resultados del proceso completo de minería de datos.

Definiciones de Minería de Datos

Veamos ahora una serie de definiciones de minería de datos, que ayudan a entender mejor en qué consiste:

- La minería de datos pretende obtener visiones en profundidad de los datos corporativos que no son fácilmente detectables. De hecho, más que analizar los resultados de la actividad, permite modelizarla construyendo patrones o categorías que la identifiquen, respondiendo a las necesidades de información del tipo ¿qué hay en los datos de interés?, o ¿qué podría ocurrir en un futuro?, en base al descubrimiento de tendencias o agrupaciones interesantes de datos. De hecho, las herramientas enmarcadas bajo la denominación de Minería de Datos (MD), permiten no sólo el análisis de información que tradicionalmente ha venido siendo realizado por los Sistemas de Soporte a la Decisión (DSS), sino, y esto es lo realmente importante y diferencial, el planteamiento y descubrimiento automáticos de hechos e hipótesis, ya sean patrones, reglas, grupos, funciones, modelos, secuencias, relaciones, correlaciones, etc. Una cualidad que resalta es la posibilidad de anticiparse a las variaciones del entorno, lo que facilitará darles una mejor y más rápida respuesta.
- La extracción de información oculta y predecible de grandes BD, es una nueva y poderosa tecnología con gran potencial para ayudar a las compañías a concentrarse en la información más importante de sus Bases de Información (Data Warehouse). Las herramientas de Data Mining predicen futuras tendencias y comportamientos, permitiendo en los negocios tomar decisiones proactivas y conducidas por un conocimiento acabado de la información. Los análisis prospectivos automatizados ofrecidos por un producto así van más allá de los eventos pasados provistos por herramientas retrospectivas típicas de sistemas de soporte de decisión. Las herramientas Data Mining pueden responder a preguntas de negocios que tradicionalmente consumen demasiado tiempo para poder ser resueltas y a los cuales los usuarios de esta información casi no están dispuestos a aceptar. Estas herramientas exploran las BD en busca de patrones ocultos, encontrando información predecible que un experto no puede llegar a encontrar porque se encuentra fuera de sus expectativas.

- La minería de datos consiste en la búsqueda de relaciones y patrones globales que se hallan presentes en las grandes BD pero que están “ocultos” entre el gran volumen de datos existente. Estas relaciones representan un conocimiento útil sobre los objetos de la BD y la realidad que representan.

Los puntos en común que observamos en las definiciones anteriores son:

- Es necesario disponer de unas BD o, mejor aún, de un almacén de datos, sobre los cuales realizar el proceso de minería.
- El proceso de minería debe ser automático en la mayor medida posible, debido a los grandes volúmenes de datos que se analizan.
- Los resultados obtenidos deben representar conocimiento útil y no evidente a primera vista.

Después de estudiar el concepto y definiciones de la minería de datos, terminamos este punto poniendo de manifiesto que las aplicaciones de MD extraen conocimiento escondido, patrones de comportamiento no explícitos, relaciones ocultas o información predictiva del almacén, sin necesidad de preguntas o peticiones específicas sino utilizando distintas técnicas, tales como algoritmos matemáticos, métodos estadísticos, modelos lógicos borrosos, algoritmos genéticos, inducciones de reglas, sistemas expertos y sistemas basados en el conocimiento y redes neuronales.

Fundamentos de la Minería de Datos

Las técnicas de Minería de Datos son el resultado de un largo proceso de investigación y desarrollo de productos. Esta evolución comenzó cuando los datos de negocios fueron almacenados por primera vez en computadoras, y continuó con mejoras en el acceso a los datos, y más recientemente con tecnologías generadas para permitir a los usuarios navegar a través de los datos en tiempo real.

La Minería de Datos está lista para su aplicación en la comunidad de negocios porque está soportada por tres tecnologías que ya están suficientemente maduras:

- Recolección masiva de datos.
- Potentes computadoras con multiprocesadores.
- Algoritmos de Data Mining.

Los componentes esenciales de la tecnología de Data Mining han estado bajo desarrollo durante décadas, en áreas de investigación como la estadística, la inteligencia artificial y el aprendizaje de máquinas. Hoy, la madurez de estas técnicas, junto con los motores de BD relacionales de alto rendimiento, han hecho que estas tecnologías sean prácticas para los entorno de data warehouse actuales.

Fases del proceso de Minería de Datos

Para alcanzar buenos resultados es necesario comprender que la minería de datos no se basa en una metodología estándar y genérica que resuelve todo tipo de problemas, sino que consiste en una metodología dinámica e iterativa que va a depender del problema planteado, de la disponibilidad de la fuente de datos, del conocimiento de las herramientas necesarias, de la metodología desarrollada, y de los requerimientos y recursos de la empresa.

El procedimiento para resolver un problema a través de la minería de datos se divide en dos grandes etapas: la preparación de los datos y la minería de datos propiamente dicha.

Pasos en la fase de preparación de los datos

- *Planteamiento del problema*: Definir de manera objetiva cuál es el problema a resolver, determinar con qué recursos humanos y tecnológicos se cuenta, cuáles son las fuentes de información y cuál es la disponibilidad de la información.
- *Selección de los datos*: De todas las fuentes de información disponibles se debe establecer cuáles son las que se van a considerar. Es decir, se decide sobre qué datos se va a trabajar, tanto desde el punto de vista físico, como desde el punto de vista lógico. Se debe realizar un tratamiento y estructuración de la información con el objetivo de presentarla de la mejor manera posible para posteriores análisis.
- *Limpieza y preprocesamiento de los datos*: En esta fase se analizan los datos con la finalidad de reorganizar la información eliminando aquella que es poco útil o completando la que nos falta. Se eliminan los datos irrelevantes, se unifican los criterios de representación que pueden no ser los mismos en todas las fuentes de datos y se eliminan redundancias y duplicados.
- *Reducción y proyección de datos*: Consiste en encontrar las características útiles que representan las dependencias de los datos en el objetivo del proceso.

Pasos en la Minería de Datos

- *Selección de técnicas de minería de datos*.
- *Selección de los algoritmos de minería de datos*. En él son seleccionados los métodos para que sean usados en la búsqueda de patrones de los datos. Esto incluye decidir qué modelos y parámetros son más apropiados para la adquisición del tipo de conocimiento deseado. A través de la entrega de los datos para los algoritmos de minería de datos seleccionados se llega al conocimiento.
- *Extracción del conocimiento. Búsqueda de patrones*: Es esta fase donde se escogen y se aplican las técnicas de minería de datos para la determinación de patrones de interés en los datos. Para ello se interpretan los resultados obtenidos a lo largo del proceso para la construcción de modelos o se buscan estructuras subyacentes dentro de la información. Las herramientas de minería de datos, analizan los datos ya preparados para extraer significado e información.
- *Construcción del modelo. Interpretación y evaluación*: Con los resultados obtenidos en la fase anterior se lleva a cabo el análisis, interpretación y evaluación para la determinación de un modelo eficiente que sea útil en la toma de decisiones.
- *Validación del modelo*: Implementar el modelo desarrollado en el proceso real y determinar su efectividad en diferentes casos de aplicación. Si las pruebas arrojan resultados satisfactorios, el modelo queda comprobado y garantizado para su uso regular. Sin embargo, si los resultados son poco satisfactorios, se debería regresar a las fases anteriores y fortalecer el análisis para mejorar el modelo final.

Elementos o Técnicas de la Minería de Datos

La aplicación ideal de la MD se llevaría a cabo sobre las BD corporativas, que como ya hemos comentado pueden ser un Almacén de Datos, o sobre otras específicas de propósito departamental (o Data Marts), contemplando elementos o técnicas como los siguientes:

- *Agentes inteligentes*: Se encargan de analizar la información para detectar patrones y relaciones, ya sea de forma automática, o bien interactuando con el analista. Las técnicas que utilizan les permiten identificar grupos, comportamientos y reglas cuyo descubrimiento habría supuesto un enorme esfuerzo de trabajo metódico. Son tomados del campo de la inteligencia artificial y entre ellos destacan los sistemas expertos, el aprendizaje automático, la visión por ordenador o la teoría de juegos. Utilizan estructuras de datos y algoritmos basados en árboles de decisión, redes neuronales, técnicas de agrupamiento y lógica difusa. Estas técnicas son especialmente adecuadas para herramientas de minería que utilizan los modelos predictivo y de descubrimiento, ya que son muy buenas en la detección de patrones.

- *Detección de alarmas*: Consiste en la ejecución periódica o permanente de ciertos agentes para detectar acciones o situaciones susceptibles de desencadenar una acción extraordinaria o fuera del ciclo ordinario, pudiéndose activar en tiempo real, o detectarse y almacenarse para su posterior análisis y tratamiento.
- *Análisis multidimensional*: Se basa en la estructuración y presentación de la información bajo aquellas perspectivas, ejes o dimensiones de interés. Las técnicas multidimensionales son muy buenas para cruzar los datos de múltiples formas y con distintos niveles de agregación. Se basan en la utilización de BD multidimensionales. Los estudiaremos con detalle en el apartado dedicado a OLAP.
- *Consultas e informes*: Ésta es la forma tradicional de obtener información a partir de BD. Las plataformas suelen incorporar herramientas de consulta (lenguaje SQL) con interfaces gráficas muy avanzados, intuitivos y fáciles de usar, cierto grado de análisis multidimensional y agentes inteligentes. Adicionalmente pueden utilizar técnicas matemáticas y estadísticas para analizar los datos obtenidos. Estas técnicas son muy apropiadas si se va a utilizar el modelo de Verificación. Su principal ventaja es que son de eficiencia probada, trabajan sobre las BD relacionales ya existentes y además es muy sencillo encontrar herramientas amigables al usuario que las soporten.
- *Tratamiento de datos*: Los datos suelen estar almacenados en los formatos más adecuados para su gestión por parte de los sistemas existentes, pero pueden no ser los más adecuados para su procesamiento por parte de la MD, de ahí que muchos desarrollos de MD incorporen módulos de tratamiento de datos con el objeto de simplificar al máximo las interfaces de datos e información.

Aplicación a la Resolución de Problemas de Gestión

Planteamiento Inicial del Problema

El desarrollo tecnológico ha aumentado considerablemente la mejora de los sistemas de almacenamiento de datos de las empresas. El problema es que, a medida que aumenta nuestra capacidad para almacenar y acceder a la información, más problemas tenemos para tratarla. Un ejemplo claro lo podemos ver en la “revolución” que ha supuesto internet y en cómo la información que se genera dentro de cualquier campo de nuestro interés aumenta considerablemente cada año, mientras que a su vez, cada vez nos vemos más incapaces de asimilarla.

En la industria, igualmente, la preocupación de las empresas por producir “mejor y más barato”, la búsqueda constante de reducir “incertidumbre” en el proceso de fabricación y el aumento creciente de la información que se tiene que los procesos productivos, hace que crezca, cada vez más, la necesidad por analizarla. Bien es cierto, que esta necesidad solo aparece cuando la empresa tiene un volumen de históricos del proceso realmente importante.

El Análisis de la Información

También la evolución de la tecnología ha facilitado y automatizado en gran medida las tareas de análisis de información. Cada paso en esta evolución se apoya en los anteriores, y cada uno de ellos ha supuesto un avance significativo para el usuario, que ha visto como cada progreso le abría nuevas posibilidades de análisis y aumentaba el nivel de abstracción de las consultas.

Para decidir cuál es la técnica más adecuada para una determinada situación, es necesario distinguir el tipo de información que se desea extraer de los datos. Según su nivel de abstracción, el conocimiento contenido en los datos puede clasificarse en distintas categorías y requerirá una técnica más o menos avanzada para su recuperación. Éstas son las tres categorías de conocimiento con las que nos podemos encontrar.

Conocimiento Evidente

Se trata de la información fácilmente recuperable con una simple consulta (por ejemplo con un lenguaje como el SQL). Un ejemplo de este tipo de conocimiento es una pregunta como “¿Cuáles fueron las ventas en España el pasado marzo?”.

Conocimiento Multidimensional

El siguiente nivel de abstracción consiste en considerar los datos con una cierta estructura. Por ejemplo, en vez de considerar cada transacción individualmente, las ventas de una compañía pueden organizarse en función del tiempo y de la zona geográfica, y analizarse con diferentes niveles de detalle (país, región, localidad, …).

Técnicamente, se trata de reinterpretar una tabla con n atributos independientes como un espacio n-dimensional, lo que permite detectar algunas regularidades difíciles de observar con la representación monodimensional clásica.

Este tipo de información es la que analizan las herramientas OLAP, que estudiaremos más adelante y que resuelven de forma automática cuestiones como “¿Cuáles fueron las ventas en España el pasado marzo? aumentando el nivel de detalle: mostrar las de Madrid”.

Conocimiento Oculto

Se trata de la información no evidente, desconocida a priori y potencialmente útil, que puede recuperarse mediante técnicas de minería de datos, como reconocimiento de regularidades. Esta información es de gran valor, puesto que no se conocía y se trata de un descubrimiento real de nuevo conocimiento, del que antes no se tenía constancia, y que abre una nueva visión del problema. Un ejemplo de este tipo sería “¿Qué tipos de clientes tenemos? ¿Cuál es el perfil típico de cada clase de usuario?”.

La Minería de Datos resuelve el problema

Como se ha visto en el punto anterior, las técnicas disponibles para extraer la información contenida en los datos son muy variadas y cada una de ellas es complementaria del resto, no excluyentes entre sí. Cada técnica resuelve problemas de determinadas características y para extraer todo el conocimiento oculto, en general será necesario utilizar una combinación de varias.

La mayor parte de la información de interés contenida en una BD, aproximadamente el 80% corresponde a conocimiento superficial, fácilmente recuperable mediante consultas sencillas con SQL. El 20% restante corresponde a conocimiento oculto que requiere técnicas más avanzadas de análisis para su recuperación. Estas cifras pueden dar la false impresión de que la cantidad de información recuperable mediante técnicas de minería de datos es despreciable. Sin embargo, se trata precisamente de información que puede resultar de vital importancia para la empresa y que no se puede desdeñar.

Básicamente, y como ya hemos comentado, la clave que diferencia la minería de datos respecto de las técnicas clásicas es que el análisis que realiza es exploratorio, no corroborativo. Se trata de descubrir conocimiento nuevo, no de confirmar o desmentir hipótesis. Con cualquiera de las otras técnicas es necesario tener una idea concreta de lo que se está buscando y, por tanto, la información que se obtiene con ellas está condicionada a la idea preconcebida con que se aborde el problema. Con la minería de datos es el sistema y no el usuario el que encuentra la hipótesis, además de comprobar su validez.

Por lo tanto, queda claro que el descubrimiento de esta información “oculta” es posible gracias a la minería de datos, que entre otras sofisticadas técnicas aplica la inteligencia

artificial para encontrar patrones y relaciones dentro de los datos permitiendo la creación de modelos, es decir, representaciones abstractas de la realidad.

La obtención de un buen modelo permitirá una buena comprensión del funcionamiento de una empresa, y será una base idónea para la toma de decisiones. Es decir, dado que el objetivo último de la gestión de los datos corporativos es ofrecer información de calidad a la dirección, cuanto más eficiente sea el proceso de minería, mayor será en cantidad y en calidad la información disponible para soportar la toma de decisiones.

Mediante éstas herramientas y técnicas se pueden obtener patrones y estructuras de información muy valiosas para la industria que pueden ayudar, mediante el análisis de los grandes volúmenes de datos de históricos almacenados, a mejorar la calidad y reducir los costes de los procesos productivos así como comprender mejor las causas que generan fallos en los mismos.

Los beneficios de la utilización de las técnicas de minería de datos en diversas organizaciones son enormes, de forma que las empresas más innovadoras, las están incorporando con gran éxito de forma extensiva.

Aplicaciones de la Minería de Datos

La información hallada a través de las técnicas de minería de datos tiene numerosas aplicaciones en el mundo empresarial.

Las aplicaciones más usadas son las que necesitan algún tipo de predicción. Por ejemplo, cuando una persona solicita una tarjeta de crédito, la compañía emisora quiere predecir si la persona constituye un buen riesgo de crédito. La predicción tiene que basarse en los atributos conocidos de la persona, como edad, sus ingresos, sus deudas, etc. Las reglas para realizar la predicción se deducen de los mismos atributos de titulares de tarjetas de crédito pasados y actuales, junto con su conducta observada.

Otra clase de aplicaciones busca asociaciones. Por ejemplo, los libros que se suelen comprar juntos. Si un cliente compra un libro, puede que la librería en línea le sugiera otros libros asociados. Puede que otros tipos de asociación lleven al descubrimiento de relaciones causa-efecto. Por ejemplo, el descubrimiento de asociaciones inesperadas entre un medicamento recién introducido y los problemas cardíacos llevó al hallazgo de que el medicamento podía causar problemas cardíacos en algunas personas. El medicamento se retiró del mercado.

Las asociaciones son un ejemplo de patrones descriptivos. Las agrupaciones son otro ejemplo de este tipo de patrones.

Algunos ejemplos de campos de aplicación de la minería de datos en el mundo empresarial son:

- Gestión de mercados y de riesgos.
- Diseño de estrategias competitivas.
- Ingeniería financiera y promoción comercial.
- Detección de fraudes.

Al igual que en el mundo empresarial, en el medio científico es muy habitual la recolección de gran cantidad de datos, de los que resulta muy difícil extraer conocimiento. Por ello, la minería de datos se está aplicando en campos como:

- Diagnóstico médico.
- Clasificación y estudio de señales biomédicas.
- Detección de patrones en imágenes astronómicas.
- Análisis de biosecuencias en biomedicina.
- Técnicas documentales.

Estudiamos ahora con más profundidad algunas de las aplicaciones más concretas de la minería de datos dentro de las organizaciones en campos como: marketing, predicción, reducción de riesgos, detección de fraudes y control de calidad.

Marketing

Éste es uno de los campos donde los éxitos de la minería de datos son más conocidos. Cuanto más precisa sea la información que tengamos sobre los clientes, mayores posibilidades tendremos de aumentar nuestros ingresos y rentabilizar al máximo nuestras acciones. El objetivo fundamental puede resumirse en determinar quién comprará qué, cuándo y dónde. Veamos tres aplicaciones concretas dentro del marketing:

- Targeting: Podemos aumentar espectacularmente el porcentaje de respuesta a una campaña de marketing si se dirige a los objetivos adecuados. La minería de datos permite detectar entre los potenciales clientes los que presentan una mayor probabilidad de responder a la campaña y dirigirla a ellos específicamente, con lo cual se consigue reducir drásticamente los costes.
- Fidelización de clientes: Conseguir un nuevo cliente o recuperar uno perdido resulta mucho más costoso que mantener uno que ya lo es. De ahí la rentabilidad de las campañas de fidelización de clientes, que detectan aquéllos que parece más probable que se vayan a perder, permitiendo llevar a cabo iniciativas que eviten dicha pérdida.
- La minería de datos también permite detectar nuevas oportunidades de mercado, comparando hábitos de consumo de diferentes clientes, por ejemplo, determinando la ubicación más conveniente para un determinado negocio.

Predicción

Conocer a priori cómo evolucionará una variable en el futuro constituye una información muy valiosa y supone una indudable ventaja competitiva. Se trata de una herramienta de evidente interés tanto desde el punto de vista comercial, como en gestión o control de procesos.

A partir de los datos históricos almacenados y utilizando técnicas de minería de datos pueden elaborarse modelos que permitan estimar con precisión la evolución de una variable de futuro. Disponer de esta información con tiempo suficiente permite adecuar la respuesta de forma óptima. Esto puede resulta útil en los campos más diversos:

- Detección de oportunidades.
- Prevención de problemas.
- Gestión óptima del personal.
- Optimización de stocks.

Reducción de Riesgos

La minería de datos permite construir sistemas de evaluación automática de riesgos, basados en la experiencia previa. Estos sistemas resultan de gran utilidad cuando la cantidad de casos a evaluar es excesiva para su procesamiento manual. El empleo de técnicas de minería de datos ha aumentado la eficacia y fiabilidad de dichos sistemas, logrando un comportamiento más similar al de los expertos humanos.

Detección de Fraudes

Aplicando técnicas de minería de datos, pueden obtenerse modelos que permitan descubrir posibles fraudes, basándose en la detección de comportamientos anómalos, en comparación con los datos registrados anteriormente.

Podemos encontrar aplicaciones concretas en operadores de telefonía o empresas de gestión de tarjetas de crédito. Estas compañías analizan el uso que los clientes hacen de sus servicios y pueden localizar, de manera muy rápida, un uso fraudulento de los mismos.

Control de Calidad

Existen numerosos ejemplos en los que se han aplicado técnicas de minería de datos para desarrollar sistemas automáticos de control de calidad. Estos sistemas suponen un considerable ahorro en el proceso productivo, puesto que facilitan:

- Detección más precisa de productos defectuosos: A menudo el control de calidad se realiza de forma manual y, por tanto, depende de una evaluación subjetiva por parte del personal encargado del mismo. El principal problema de este método es que el criterio de calidad no es estable sino que depende de la persona que realiza el análisis. La minería de datos permite desarrollar sistemas automáticos de control de calidad que discriminan los productos defectuosos con un alto grado de precisión y fiabilidad, según un criterio objetivo.
- Localización precoz de defectos: El control de calidad no sólo debe realizarse al final del proceso. Cuanto antes se detecte un fallo, menor será su impacto. A menudo no resulta fácil medir la variable que determina la calidad del producto en tiempo real o en la cadena de producción. En estos casos, es imprescindible utilizar técnicas de minería de datos para descubrir posibles relaciones que permitan detectar los fallos utilizando las variables disponibles durante el proceso.
- Identificación de causas de fallos: La minería de datos no sólo resulta útil para discriminar los productos defectuosos. También ayuda a determinar los fallos más frecuentes así como identificar las causas de los mismos. Esto permite adoptar medidas para evitarlos en el futuro.
- Análisis no destructivo: A menudo, para obtener la información que se necesita, hay que realizar un análisis destructivo. Un ejemplo típico es la evaluación de la resistencia de un material, medida que se establece forzándolo hasta que se rompe. Utilizando minería de datos es posible estimar con bastante exactitud el valor de este tipo de parámetros en función de otras características que sí pueden medirse sin destruir el producto. Esto permite controlar la calidad de todos los productos fabricados y no sólo de una pequeña muestra, ya que no se destruyen con el examen.

Tecnología y Algoritmos

Antes de estudiar las técnicas y algoritmos principales, vamos a ver los modelos que a lo largo del tiempo han ido apareciendo y en los que se apoya la minería de datos.

Modelos de la Minería de Datos

Modelo de Verificación

Este es el modelo más parecido al proceso tradicional de extracción de información basado en lenguajes de consulta a BD (por ejemplo SQL). Su principal característica es que no extrae información nueva, sino que, basándose en los datos del almacén, verifica la validez de las afirmaciones que se le presentan.

El proceso comienza por el establecimiento de una hipótesis por parte del usuario. Este, a continuación, solicita a la herramienta que verifique su validez. Una vez recibida la respuesta, el usuario puede refinar o detallar la hipótesis, preparar unas preguntas más específicas y solicitar una nueva verificación. De esta manera se consigue un proceso iterativo dirigido por un operador humano.

La desventaja de este modelo es, que si al usuario no se le ocurre realizar una pregunta clave, o no ve una relación importante entre diferentes elementos de la BD, la herramienta por sí sola carece de iniciativa para investigar por su propia cuenta.

Los nuevos Modelos Automáticos

La minería de datos ha dado lugar a una paulatina sustitución del análisis de datos dirigido a la verificación, por un enfoque de análisis de datos dirigido al descubrimiento del conocimiento. La principal diferencia entre ambos se encuentra en que en el último, se descubre información sin necesidad de formular previamente una hipótesis. La aplicación automatizada de algoritmos de minería de datos permite detectar fácilmente patrones en los datos, razón por la cual esta técnica es mucho más eficiente que el análisis dirigido a la verificación cuando se intenta explorar datos procedentes de repositorios de gran tamaño y complejidad elevada. Dichas técnicas emergentes se encuentran en continua evolución como resultado de la colaboración entre campos de investigación tales como BD, reconocimiento de patrones, inteligencia artificial, sistemas expertos, estadística, visualización, recuperación de información, y computación de altas prestaciones.

Los algoritmos de minería de datos se clasifican en dos grandes categorías de modelos con distintas denominaciones:

- Modelos predictivos, también llamados:
  - Modelos supervisados.
  - Modelos basados en la memoria.
  - Minería de datos dirigida.
- Modelos de descubrimiento del conocimiento, también llamados:
  - Modelos no supervisados.
  - Modelos descriptivos.
  - Minería de datos no dirigida.

Por lo tanto con los nuevos modelos usamos la minería de datos para:

- *Predecir*: Utilizar algunas variables o campos en un BD para predecir valores desconocidos o futuros.
- *Describir*: Encontrar patrones que describan la información (interpretables por el hombre).

Modelos Predictivos

Los algoritmos supervisados o predictivos predicen el valor de un atributo (etiqueta), de un conjunto de datos, conocidos otros atributos (atributos descriptivos). A partir de datos cuya etiqueta se conoce, se induce una relación entre dicha etiqueta y otra serie de atributos. Esas relaciones sirven para realizar la predicción en datos cuya etiqueta es desconocida. Esta forma de trabajar se conoce como aprendizaje supervisado y se desarrolla en dos fases:

- Entrenamiento: Construcción de un modelo usando un subconjunto de datos con etiqueta conocida.
- Prueba: Prueba del modelo sobre el resto de los datos.

El usuario indica sobre qué variables se quiere obtener la predicción y el sistema proporciona la respuesta. Esta respuesta la puede proporcionar explicando cómo la consiguió, lo cual a su vez puede ser una información tan valiosa como la respuesta en sí misma, o sin explicarlo.

Cuando una aplicación no es lo suficientemente madura no tiene el potencial necesario para una solución predictiva fiable. En este caso se puede optar por diversos caminos alternativos:

- Modelo predictivo restringido: No se obtiene predicción alguna.
- Modelo predictivo no restringido: Se obliga a la realización de una predicción de menor fiabilidad.
- Modelos de descubrimiento del conocimiento: Que descubren patrones y tendencias en los datos actuales (no utilizan datos históricos).

Ejemplos: ¿Cuál es el riesgo de este cliente?, ¿Se quedará el cliente?

Algunas técnicas asociadas a los modelos predictivos:

- Clasificación: Clasificar datos en clases predefinidas.
- Estimación: A diferencia de la clasificación (que trata con resultados discretos), la estimación trata con valores numéricos continuos. A partir de un conjunto de valores de entrada, la estimación obtiene un valor para cierta variable continua, como puede ser una renta, la altura, etc.
- Predicción de valores: Una predicción no es más que un tipo de clasificación o estimación.
- Regresión: función que convierte datos en valores de una función de predicción.
- Árboles de decisión: Son estructuras en forma de árbol que representan conjuntos de decisiones. Estas decisiones generan reglas para la clasificación de un conjunto de datos.
- Redes neuronales artificiales: Modelos predecibles no lineales que aprenden a través del entrenamiento y semejan la estructura de una red neuronal biológica.
- Series temporales.

Modelos de Descubrimiento del Conocimiento

El objetivo de estos modelos es establecer algún tipo de relación entre todas las variables.

En estos modelos se utiliza la herramienta de minería para descubrir nueva información que no estaba anteriormente en el almacén de forma explícita. Según este modelo es la propia herramienta quien se plantea sus propias preguntas, sin necesidad de que el usuario establezca hipótesis o realice preguntas concretas, aunque, éste puede intervenir para guiar los caminos a explorar.

Habitualmente esta búsqueda se dirige hacia la categorización de los registros en grupos para detectar patrones aplicables o extraer relaciones implícitas en los datos. También es común la búsqueda de elementos extraños o fuera de lo normal.

Ejemplo: Un cliente que compra productos dietéticos es tres veces más probable que compre caramelos.

Algunas técnicas asociadas a los modelos de descubrimiento del conocimiento:

- Asociación: Permite establecer las posibles relaciones entre acciones o sucesos aparentemente independientes.
- Reconocimiento de patrones: Permite la asociación de una señal o información de entrada con aquella o aquellas con las que guarda mayor similitud, y que ya están catalogadas en el sistema.
- Segmentación o agrupamiento: Esta herramienta posibilita la identificación de tipologías o grupos en los cuales los elementos guardan similitud entre sí y se diferencian de los de otros grupos.

- Clustering: Es la tarea de segmentar un grupo diverso en un número de subgrupos más similar (denominados clusters). Lo que distingue el clustering de la clasificación es que éste no requiere un conjunto predefinido de clases.
- Reglas de asociación: Se trata del agrupamiento por afinidad que tiene como objetivo determinar qué cosas van juntas.
- Detección de desviaciones.

Clasificación

Dentro de los modelos de predicción, una de las técnicas más importantes es la clasificación. En este apartado vamos a describir qué es la clasificación, a estudiar técnicas para la creación de un tipo de clasificadores, denominados clasificadores de árboles de decisión y se analizarán otras técnicas de predicción.

De manera abstracta, el problema de la clasificación es el siguiente: dado que los elementos pertenecen a una de las clases y dados los casos pasados de los elementos junto con las clases a las que pertenecen, el problema es predecir la clase a la que pertenece un elemento nuevo.

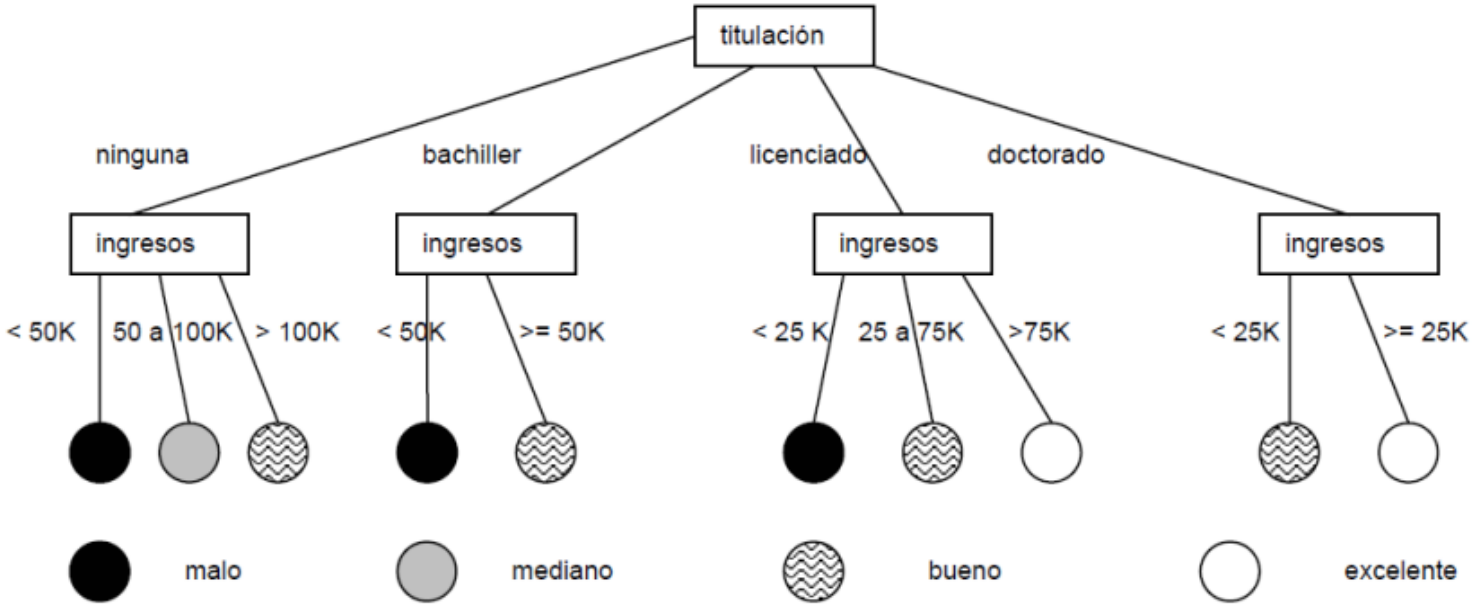
La clasificación se puede llevar a cabo hallando reglas que dividan los datos dados en grupos disjuntos. Continuando con el ejemplo de un banco tiene que decidir si debe conceder una tarjeta de crédito a un solicitante. El banco tiene diversa información sobre esa persona, la cual puede utilizar para adoptar una decisión. Para adoptar la decisión el banco asigna un nivel de valor de crédito de: excelente, bueno, mediano o malo a cada integrante de un conjunto de muestras de clientes actuales según su historial de pagos. Luego, el banco intenta hallar reglas que clasifiquen a sus clientes como excelentes, buenos, medianos o malos.

El proceso de creación de clasificadores comienza con un muestra de los datos, denominada *conjunto de formación*. Para cada tupla del conjunto de formación ya se conoce la clase a la que pertenece. Existen diversas maneras de crear clasificadores. Una de las técnicas más utilizadas para este fin son los clasificadores de árboles de decisión.

Clasificadores de árboles de decisión

Los clasificadores de árboles de decisión son una técnica muy utilizada para la clasificación. Como sugiere su nombre estos clasificadores utilizan un árbol. Cada nodo hoja tiene una clase asociada, y cada nodo interno tiene un predicado o función asociado.

Continuando con el ejemplo, para concretar las reglas que clasifican los clientes en excelentes, buenos, medianos o malos, vamos a considerar dos atributos: titulación e ingresos. En la siguiente figura se muestra un árbol de decisión que establece las reglas concretas de clasificación.



Para clasificar un nuevo caso se empieza por la raíz y se recorre el árbol hasta alcanzar una hoja. En los nodos internos se evalúa el predicado o función, para hallar a que nodo hijo hay que ir. El proceso continúa hasta llegar a un nodo hoja.

Creación de Clasificadores de árboles de decisión

La pregunta que se plantea es el modo de crear un clasificador de árboles de decisión, dado un conjunto de casos de formación. La manera más frecuente de hacerlos es utilizar un algoritmo *impaciente*, que trabaja de manera recursiva, comenzando por la raíz y construyendo el árbol hacia abajo. Inicialmente solo hay un nodo, la raíz, y todos los casos de formación están asociados con este nodo.

En cada nodo, si todos o casi todos los ejemplos de formación asociados con el nodo pertenecen a la misma clase, el nodo se convierte en un nodo hoja a esa clase. En caso contrario, hay que seleccionar un *atributo de partición* o *condiciones de partición* para crear nodos hijos. En el ejemplo elegido, se escoge el atributo titulación y se crean cuatro hijos, uno por cada valor de la titulación.

Las particiones en menor número de conjuntos son preferibles a las particiones en muchos conjuntos, ya que llevan a árboles de decisión más sencillos y significativos.

Hay que averiguar el modo de hallar la mejor partición para un atributo. El modo de dividir un atributo depende del tipo de atributo. Los atributos pueden tener dos tipos de valores:

- *Valores continuos*: Los valores se pueden ordenar de manera significativa para la clasificación, como la edad o los ingresos.
- *Valores categóricos*: No tienen ningún orden significativo para la clasificación, como los nombres de los departamentos o de los paises.

Generalmente los atributos que son números se tratan como valores continuos, y los atributos de cadenas de caracteres se tratan como categóricos. En el ejemplo escogido se ha tratado el atributo titulación como categórico y el atributo ingresos como valor continuo.

En primer lugar se considera el modo de hallar las mejores particiones para los atributos continuos. Por sencillez solo se consideran *particiones binarias* de los atributos con valores continuos, es decir, particiones que den lugar a dos hijos. En caso de las *particiones múltiples* ya es más complicado y se pueden dar con valores continuos o categóricos.

Para los atributos categóricos se pueden tener particiones múltiples, con un hijo para cada valor del atributo. Esto funciona muy bien para los atributos categóricos con pocos valores diferentes, como la titulación o el sexo.

La idea principal de construcción de árboles de decisión es la evaluación de los diferentes atributos y de las distintas condiciones de partición y la selección del atributo y de la condición de partición que generen el índice máximo de ganancia de información. El mismo procedimiento funciona de manera recursiva en cada uno de los conjuntos resultantes de la partición, lo que hace que se construya de manera recursiva el árbol de decisión.

Otros Tipos de Clasificadores

Hay varios tipos de clasificadores a parte de los clasificadores de árbol. Dos tipos que han resultado bastante útiles son:

- *Clasificadores de redes neuronales*: Utilizan los datos de formación para adiestrar redes neuronales artificiales.
- *Clasificadores bayesianos*: Hallan la distribución de los valores de los atributos para cada clase de los datos de formación.

Regresión

La regresión trata la predicción de valores, no de clases. Dados los valores de un conjunto de variables, X1, X2, …, Xn se desea predecir el valor de una variable Y. Por ejemplo se puede tratar el nivel educativo con un número y los ingresos con otro número, y con base a estas dos variables, querer predecir la posibilidad de impago, que podría ser un porcentaje de probabilidad de impago o el importe impagado.

Asociaciones

Como ya se dijo, las asociaciones permiten establecer las posibles relaciones entre acciones o sucesos aparentemente independientes. Así, se puede reconocer cómo la ocurrencia de un determinado suceso puede inducir la aparición de otro u otros. Este tipo de herramientas son particularmente útiles, por ejemplo, para comprender los hábitos de compra de los clientes y para la concepción de ofertas, de ventas cruzadas y del “merchandising”.

Los comercios en general suelen estar interesados en las asociaciones entre los diferentes artículos que compra la gente. Ejemplos de estas asociaciones son:

- Alguien que compra pan es bastante probable que compre también leche.
- Una persona que compró un libro X es bastante probable que también compre el libro Y.

Reglas de Asociación

Un ejemplo de regla de asociación es: *pan* => *leche*. En el contexto de las compras de alimentación, la regla dice que los clientes que compran pan también tienden a comprar leche con una probabilidad elevada.

Una regla de asociación debe tener una población asociada: la población consiste en un conjunto de casos. En el ejemplo de la tienda de alimentación, la población puede consistir en todas las compras en la tienda de alimentación, cada compra es un caso.

Las reglas tienen un soporte, así como una confianza asociados. Los dos se definen en el contexto de la población:

- *El soporte*: Es una medida de la fracción de la población que satisface tanto el antecedente como el consecuente de la regla. Por ejemplo, supongamos que solo el 0,001% de todas las compras incluyen leche y clavos. El soporte de la regla *leche => clavos* es bajo. Las empresas no suelen estar interesadas en reglas que tienen un soporte bajo, ya que afectan a pocos clientes y no merece la pena prestarles atención.
- *La confianza*: Es una medida de la frecuencia con que el consecuente es cierto, cuando lo es el antecedente. Por ejemplo la regla *pan => leche* tiene una confianza del 80% si el 80% de las compras que incluyen pan incluyen también leche. Hay que tener en cuenta que la confianza de *pan => leche* puede ser muy diferente de la confianza *leche => pan*, aunque las dos tiene el mismo soporte.

Otros Tipos de Asociación

El uso de meras reglas de asociación tiene varios inconvenientes. Uno de los principales es que muchas asociaciones no son muy interesantes, ya que pueden predecirse. Por ejemplo, si mucha gente compra cereales y mucha gente compra pan, se puede predecir que un número bastante grande de personas comprará las dos cosas, aunque no haya ninguna relación entre las dos compras. Lo que resultaría interesante es una desviación de la ocurrencia conjunta de las dos compras. O dicho en términos estadísticos, lo que se busca son *correlaciones* entre los artículos.

Otro tipo importante son las asociaciones de secuencias. Las series de datos temporales, como las cotizaciones bursátiles en una serie de días, constituyen un ejemplo de datos de secuencias.

Bases de Datos Multidimensionales

La idea básica empleada por las BD multidimensionales (BDM) es muy sencilla: en lugar de utilizar tablas bidimensionales para almacenar los datos, como se hace en una BD relacional (BDR), emplea tablas n-dimensionales (o hipercubos). Es algo parecido a utilizar una hoja de cálculo para el tratamiento de datos, solo que, se podrán utilizar más de dos dimensiones y se dispondrá de otras capacidades adicionales.

Una BDM está diseñada para los sistemas de soporte de decisiones en la cual los datos tienen una estructura matricial (multidimensional) para su almacenamiento. Este tipo de organización admite consultas más complejas.

Análisis Multidimensional

El análisis multidimensional consiste en analizar hechos económicos o, de otros tipos, desde la perspectiva de sus dimensiones, abarcando los diferentes niveles de éstas.

Con el análisis multidimensional se da respuesta a las consultas complicadas de los usuarios, que reflejan los diversos componentes que tienen sus organizaciones. Estos componentes puedes ser de dos tipos: cuantitativos y cualitativos.

A estos componentes también se les llama dimensiones, y a los valores de los componentes (o dimensiones) se les llama atributos. Además, el detalle con el que se muestran los atributos puede variar, cada dimensión se puede descomponer en diferentes niveles de detalle, y éstos dependen de las necesidades del usuario.

Las dimensiones definen dominios como geografía, producto, tiempo, cliente, …

Los miembros de una dimensión se agrupan de forma jerárquica (dimensión geográfica: ciudad, provincia, autonomía, país, …).

El Esquema Multidimensional

La realización del análisis multidimensional a partir de trozos de información no sería nada práctica, lo que se pretende es tener disponible toda la información formando un solo conjunto, al que llamaremos esquema multidimensional.

Una de las características principales del esquema multidimensional es la agregabilidad, gracias a la cual se pueden presentar los valores de una determinada dimensión según sus distintos niveles de detalle. Como es lógico para poder realizar agregación es necesario tener datos en el nivel más bajo de cada dimensión, y los niveles superiores se calcularán a partir de éstos.

Para un óptimo análisis este esquema se soporta en las BBDD multidimensionales, éstas almacenan los datos en estructuras llamadas hipercubos (más de tres dimensiones). En la práctica estos hipercubos no son grandes matrices, sino que son matrices más reducidas que aparecen como una sola matriz. Esto reduce el espacio de índice requerido.

El esquema multidimensional puede ser soportado encima de un SGBD relacional (ROLAP: OLAP sobre BD Relacionales). Para ello el esquema multidimensional deberá ser transformado para poder implementarse sobre un SGBD relacional (que solo soporta tablas planas). Una de las formas de hacer esta transformación es utilizar el “esquema en estrella”, que estudiaremos más adelante.

Características del Análisis Multidimensional

- Navegabilidad: Cuando se habla de navegar se refiere a que se puede pasar de un punto a otro del esquema multidimensional. Estos movimientos son:
  - Perforación (drill-down): Consiste en variar el nivel de detalle de los datos, desde los datos más resumidos a los más detallados. Se dice que drill-down es desagregar y Roll-up es agregar.
  - Segmentación (slice and dice): Consiste en “recortar” un subconjunto de los datos moviéndose por los distintos datos de una misma dimensión o cambiando de dimensión. Es decir, es la capacidad de ver la BD desde diferentes puntos de vistas. El corte suele hacerse a lo largo del eje del tiempo para analizar tendencias. Se dice que *slice* es proyección y que *dice* es selección.
- Visualización: La presentación de los resultados se suele hacer en forma de cuadros o tablas de dos dimensiones, con el cálculo de totales parciales y generales. Se suelen fijar un conjunto de valores de dimensiones y mostrar en la tabla de dos dimensiones los valores en función de esas dimensiones.
- Representación gráfica: Suele ser un gráfico de dos dimensiones, donde los valores de las dimensiones fijadas aparecen como comentarios y las dimensiones variables son los ejes de coordenadas. Con este tipo de representaciones se suele perder una dimensión.
- Representación mediante mapas: Muy utilizada para dimensiones geográficas, donde se realizan perforaciones seleccionando la zona deseada.
- Cálculos dinámicos.

Modelo de Datos Multidimensional (MDM)

Se define un modelo de datos multidimensional como la disciplina específica para modelizar datos que es una alternativa a la modelización E/R. Es un modelo de datos (estático y dinámico) basado en estructuras multidimensionales.

Un modelo multidimensional contiene la misma información que un modelo E/R pero agrupa la información en un formato simétrico cuyos objetivos serían:

- Que el usuario entienda mejor el modelo.
- Que el rendimiento y tiempo de respuesta de las consultas sea el óptimo.
- Que los cambios en el modelo se hagan con menos impacto y mayor facilidad.

Veamos ahora los elementos que componen la visión estática de un modelo de datos multidimensional:

- Esquema de hecho (esquema de cubo): Es el objeto a analizar. Ejemplos: empleados, ventas, stocks, …
- Atributos de hecho o de medida: Atributos de tipo cuantitativo cuyos valores (cantidades) se obtienen generalmente por aplicación de una función estadística que resume un conjunto de valores en un único valor. Ejemplos: nº de empleados, cantidad vendida, precio medio, …
- Funciones resumen: Funciones de tipo estadístico que se aplican a los atributos de hecho. Ejemplos: frecuencia, suma, media, máximo, etc.
- Dimensiones: Cada uno de los ejes en un espacio multidimensional. Ejemplos: tiempo, espacio, productos, intervalos del nº de empleados, departamentos, etc.
- Atributos de dimensión o de clasificación: Atributos de tipo cualitativo (sus valores son modalidades) que suministran el contexto en el que se obtienen las medidas en un esquema de hecho. Ejemplos: días, semanas, ciudades, provincias, etc.
- Jerarquías: Varios atributos de dimensión unidos mediante una relación de tipo jerárquico. Ejemplos: día -> semana -> mes -> año.
- Series temporales: Una de las dimensiones más habituales de cualquier BDM es el tiempo. Para guardar datos en función del tiempo, se utilizan las series temporales, que son tratadas como una dimensión más.

Vamos a estudiar ahora con más detalle dos de los elementos fundamentales en las BDM: las dimensiones y las jerarquías. Utilizaremos para ello una serie de ejemplos que nos van a ayudar a entender mejor estos dos elementos.

Dimensiones

Ejemplo 1

Supongamos que queremos implementar una sencilla BD para almacenar la cantidad de dinero que se gasta en el pago de las pensiones atendiendo al tipo de pensión y a la comunidad autónoma en que se paga.

En el caso de que hubiera dos tipos de pensiones, se podría establecer una BDM con una estructura similar a la de una hoja de cálculo, empleando tantas filas como tipos de pensiones y tantas columnas como comunidades. El gasto correspondiente a cada comunidad y pensión se almacenaría en la celdilla correspondiente, tal como se muestra a continuación:

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17
P1																	
P2																	

El equivalente relacional sería una tabla de 34 filas y 3 columnas: tipo de pensión, comunidad autónoma y gasto.

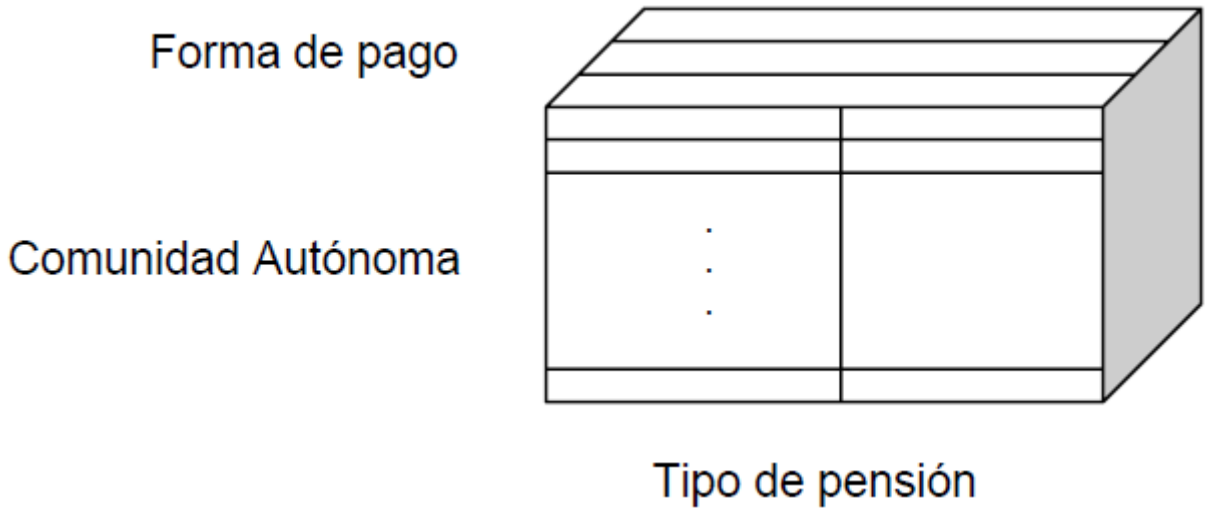
En este ejemplo sencillo, el espacio de almacenamiento utilizado en ambos casos es el mismo, pero, ¿qué ocurre con los tiempos de acceso a la información?

Si se quiere acceder al gasto en un tipo de pensión y una comunidad determinados (una sola fila), el tiempo de acceso será similar, siempre que la tabla relacional esté ordenada o tenga definido un índice por tipo de pensión y comunidad autónoma.

Si se quiere obtener el gasto en pensiones de tipo 1 (P1) para todas las comunidades, entonces el tiempo de respuesta de la BDM es mejor, ya que solo tiene que sumar una fila de la matriz (17 sumas). En cambio en la BDR se debe recorrer todos los registros de la tabla para localizar aquellos que cumplan la condición definida (34 registros) o crear más índices.

Ejemplo 2

Supongamos ahora, que también es necesario almacenar la forma de pago de las pensiones y que dicha forma de pago puede ser en efectivo, por talón o transferencia. La BDM tendría el aspecto siguiente:



En esta estructura se emplea cada una de las tres dimensiones del cubo para representar cada uno de los campos que se utilizarían en el modelo relacional. Las celdas resultantes se emplean para almacenar el gasto para cada tripleta (CA, TP, FP).

El equivalente relacional sería una tabla con 102 filas y 4 columnas: tipo de pensión, comunidad autónoma, forma de pago y gasto.

De nuevo, las consultas de agregados (totales) serían más costosas en la BDR que en la BDM.

Jerarquías

Otro aspecto fundamental de las BDM es la posibilidad de jerarquizar las dimensiones. Vamos a ver esto con otro ejemplo.

Ejemplo 3

Supongamos que, además de conocer el gasto por comunidades, se quiere saber también el gasto por localidades dentro de cada comunidad.

La manera inmediata de representar esto consiste en añadir una nueva dimensión para crear un hipercubo de cuatro dimensiones. Sin embargo esta solución no es eficiente, ya que para cada fila de cada localidad, solo una de las celdillas contendrá el valor. Dicha celdilla será la correspondiente a la comunidad a la que pertenece la localidad.

Con esta estructura se gasta mucho espacio de almacenamiento en celdillas que jamás van a contener datos, por lo tanto hay que buscar otro mecanismo que lo evite. La

solución a este problema es crear una jerarquía de niveles en cada dimensión para representar los diversos grados de detalle.

Si se dispone de este mecanismo, la solución al caso de las localidades sería tan simple como jerarquizar la dimensión de las comunidades autónomas, estableciendo las localidades como escalón inferior en la jerarquía.

Para ofrecer esta alternativa el gestor debe ser capaz, de operar con las celdillas, de reconocer si el valor almacenado corresponde a una comunidad o a una localidad, de forma que al hallar totales o realizar cualquier otro tipo de operación no mezcle valores correspondientes a diferentes niveles jerárquicos.

Por lo tanto, una celda es una posición formada por la intersección de cada uno de los elementos de las dimensiones que forman el cubo. La celda puede contener cero, uno o varios datos (cantidades).

Este concepto de jerarquía es extensible a más de dos niveles, por lo que se puede afinar el grado de detalle obtenido al realizar las consultas.

BD Multidimensionales vs BD Relacionales

Terminamos este apartado de BDM realizando una comparación entre estas BD y las BDR que son más conocidas.

La utilización de BDM ofrece ventajas sobre las BDR siempre que se vaya a trabajar sobre datos agregados, totales, subtotales, etc. También son superiores a la hora de trabajar con series temporales, obtener vistas de unos datos en función de otros (vistas bidimensionales del hipercubo que forma la BDM) y manejar diversos grados de detalle. En resumen son unas BD adecuadas para el estudio de alto nivel de los datos, al ofrecer una mayor flexibilidad y rapidez de acceso para el análisis de los mismos.

Por otra parte, si lo que se quiere es acceder a un dato individual básico, la ventaja de las BDM desaparece a favor de las BDR. Estas son capaces de recuperar un dato individual con la misma eficiencia que las multidimensionales, suelen ser capaces de almacenar mayor cantidad de información y además, dada su utilización masiva en sistemas OLTP, están optimizadas para la inserción de registros y el control concurrente de usuarios.

La utilización de ambos tipos de BD no es excluyente. De hecho es frecuente utilizar una BDR para almacenar los datos de nivel más bajo de la jerarquía de una BDM, de forma que si se desea obtener un dato básico, se excava a través de la jerarquía multidimensional hasta acceder a la BDR.

Procesamiento Analítico en Línea (OLAP)

Dado que el volumen de datos almacenados en las BD suele ser elevado, hay que resumirlos de algún modo si se quiere obtener información que puedan utilizar los usuarios. Las herramientas OLAP soportan el análisis interactivo de la información de resumen.

Definición de OLAP

El acrónimo OLAP significa Procesamiento Analítico en Línea (On-Line Analytical Processing), y se utiliza para hacer referencia a sistemas y herramientas de minería de datos que usan técnicas multidimensionales para la extracción y el análisis de los datos.

Según E.F. Codd, que fue quién acuñó el término, OLAP es: la síntesis, el análisis y la consolidación dinámica de grandes volúmenes de datos multidimensionales.

Según otra definición de OLAP: se trata de un término inventado para describir una aproximación dimensional interactiva al soporte de toma de decisiones (análisis desde la perspectiva de sus componentes o dimensiones, contemplando también los distintos niveles o jerarquías que éstas poseen).

Siempre que se habla de tecnología OLAP el adjetivo más utilizado es “multidimensional”, ya sea para referirse a los datos, a su estructura, a la BD que se emplea o a casi cualquier otro aspecto del OLAP. Esta caracterización llega hasta el punto de identificar el OLAP y las BD multidimensionales como una misma cosa. Aunque indudablemente ambas tecnologías están relacionadas, la utilización de OLAP no implica necesariamente la utilización de BD multidimensionales.

La pregunta que debemos respondes es, ¿qué requiere el usuario de OLAP? La respuesta es:

- Conceptos familiares para el usuario final: Dimensiones, medidas y jerarquías.
- Acceso inmediato a los datos.
- Información consistente.
- Navegación y consulta sencillas.
- Capacidades de generación de informes.
- Datos precalculados.
- Soporte de grandes volúmenes de datos.
- Flexibilidad de manejo y presentación.
- Potentes capacidades de análisis: Agregaciones, comparaciones, ratios, correlaciones, análisis de situaciones, contraste de hipótesis, descubrimiento de patrones y tendencias, previsiones, series temporales, etc.

Características de los Sistemas OLAP

Las características básicas de los sistemas OLAP son las siguientes:

- Ofrecen una visión multidimensional y jerarquizada de los datos.
- Son capaces de analizar tendencias a lo largo del tiempo.
- Pueden presentar vistas de un número reducido de dimensiones elegido por el usuario.
- Permiten ahondar en la jerarquía de los datos para acceder a los de más bajo nivel.
- Son interactivos y soportan múltiples usuarios concurrentes.

Resulta ahora claro, vistas sus características, como los sistemas OLAP pueden beneficiarse de las funcionalidades de una BDM:

- La visión multidimensional y la jerarquizada van explícitas en la propia estructura de la BD. La herramienta OLAP, que posiblemente esté integrada en la BDM, solo tiene que ocuparse del manejo del cubo hiperdimensional para extraer los datos conforme a los criterios establecidos por el usuario.
- El estudio de tendencias se puede realizar aprovechando las series temporales de la BDM o, si no se dispone de dicho tipo de datos, realizando las operaciones y conversiones necesarias para manejar el tiempo como una dimensión adicional de la BD.
- La presentación de vistas se conoce en la jerga OLAP como “slice and dice” (cortar y trocear) y se podría traducir en algo así como segmentación. Esta característica de una herramienta OLAP consiste en la capacidad de extraer “rodajas” del hipercubo que forma la BDM. Estas rodajas se extraen dado un valor fijo para una o varias dimensiones y tomando el hipercubo resultante.
- La capacidad de perforar en los niveles de jerarquía se realiza, de nuevo, aprovechando la propia estructura de la BDM subyacente. En el caso de que se

utilice una BDR como escalón inferior de la jerarquía, la herramienta OLAP debe ocuparse de que el acceso a dicho nivel sea transparente para el usuario.
- La interactividad y el soporte de múltiples usuarios simultáneos son capacidades que dependen en gran medida de los tiempos de respuesta del gestor de BD empleado, por lo que se puede utilizar como criterio orientativo a la hora de elegir el producto que se va a adquirir para construir el sistema.

Implementación de Sistemas OLAP

Como ya hemos comentado, debido a su orientación hacia el manejo de los datos organizados en dimensiones, el entorno natural de trabajo de los sistemas OLAP son las bases de datos multimensionales. No obstante también pueden trabajar sobre BD Relacionales, aunque en este caso sus prestaciones se ven disminuidas. Atendiendo a este criterio, los sistemas OLAP se pueden dividir en tres tipos principales, que estudiamos a continuación.

MOLAP (Multidimensional-OLAP)

Los primeros sistemas OLAP utilizaban arrays de memoria multidimensionales para almacenar los cubos de datos y se denominan OLAP multidimensional (MOLAP).

Por lo tanto, funcionan sobre BD multidimensionales. Requieren un esfuerzo previo de modelización y construcción de la BD multidimensional y de otro continuo consistente en migrar los datos en formato relacional al nuevo formato multidimensional. A cambio ofrecen un rendimiento muy superior a la hora de realizar la extracción y el análisis de los datos, puesto que los datos a los que acceden están organizados en dimensiones y jerarquías.

Los datos se almacenan en un sistema de matrices (hipercubo) en donde cada eje es una dimensión.

ROLAP (Relational-OLAP)

Posteriormente, los servicios OLAP se integraron en los sistemas relacionales y los datos se almacenaron en las BD relacionales. Estos sistemas se denominan sistemas OLAP relacionales (ROLAP).

Estos sistemas permiten trabajar sobre las BD corporativas ya establecidas, ahorrando así el trabajo de crear y mantener nuevas BD multidimensionales. A cambio deben ocuparse de realizar la conversión entre la visión relacional de los datos mantenida por el SGDBR y el manejo multidimensional y jerárquico que debe ofrecer al usuario, lo cual acarrea un coste en tiempo y recursos de máquina.

El almacenamiento se suele realizar en un esquema en estrella (no normalizado) o copo de nieve (normalizado), que vamos a estudiar posteriormente con detalle.

Las tendencias actuales en estos sistemas ROLAP son:

- Desarrollo de técnicas específicas para el almacenamiento (índices join, bitmap, …) y optimización de consultas.
- Crear servidores SQL ampliado especializados en funcionar como Almacén de Datos.

A su vez, estas tendencias dan lugar a dos tipos de modelos ROLAP:

- SGBD especializados de SQL: Proporcionan un lenguaje de consulta avanzado y soporte para el proceso de consultas SQL sobre esquemas en estrella y copo de nieve en entornos de solo lectura.

- Servidores ROLAP: Servidores intermedios que se sitúan entre el SGBDR y las herramientas cliente. Este middleware está especializado en el soporte de consultas OLAP multidimensionales que se optimizan para servidores relacionales específicos.

Respecto a la elección entre MOLAP y ROLAP, en la práctica resulta mucho más habitual encontrar sistemas de almacén de datos, junto con sus correspondientes herramientas OLAP y de minería de datos, implementadas mediante BD relacionales. Esto es debido a la mayor experiencia de que se dispone para trabajar sobre BD Relacionales, a la gran cantidad de productos ya disponibles en el mercado y a la confianza que las organizaciones tienen en este tipo de BD.

HOLAP (Hybrid-OLAP)

Además de los dos sistemas descritos, aparecen los sistemas híbridos, que almacenan algunos resúmenes en la memoria y los datos básicos y otros resúmenes en las BD Relacionales, se denominan sistemas OLAP híbridos (HOLAP).

Dicho de otra forma, los sistemas HOLAP proporcionan análisis multidimensional accediendo indistintamente a BD Multidimensionales o Relacionales.

Muchos sistemas OLAP se implementan como sistemas cliente-servidor. El servidor contiene la BD Relacional y los cubos de datos MOLAP. Los sistemas clientes obtienen vistas de los datos comunicándose con el servidor.

ROLAP: Tipos de Diseño

Nos detenemos ahora en los sistemas ROLAP, que a pesar de no ser los que mejor se adaptan a una herramienta OLAP, si son muy utilizados. Veamos los diferentes tipos de diseño que se deben realizar para que estos sistemas puedan dar una respuesta eficiente.

Esquema en Estrella

Esquema relacional adaptado a la representación de datos multidimensionales. Se basa en una serie de tablas que representan dimensiones unidas mediante claves ajenas, a una principal que actúa como nexo llamada tabla de hechos y que almacena datos agregados y precalculados (tablas no normalizadas).

Tabla de Hechos

El contenido de una tabla de hechos está formado por:

- Clave principal: Concatenación de las claves de todas las tablas de dimensión asociadas a la tabla de hechos.
- Claves ajenas: Que referencian a las claves de las correspondientes dimensiones.
- Atributos de Hecho: atributos de tipo cuantitativo cuyos valores (cantidades) se obtienen generalmente por aplicación de una función estadística que resume un conjunto de valores en un único valor. Ejemplos: nº de empleados, cantidad vendida, precio medio, et.

Por otro lado, las características principales de una tabla de hechos son:

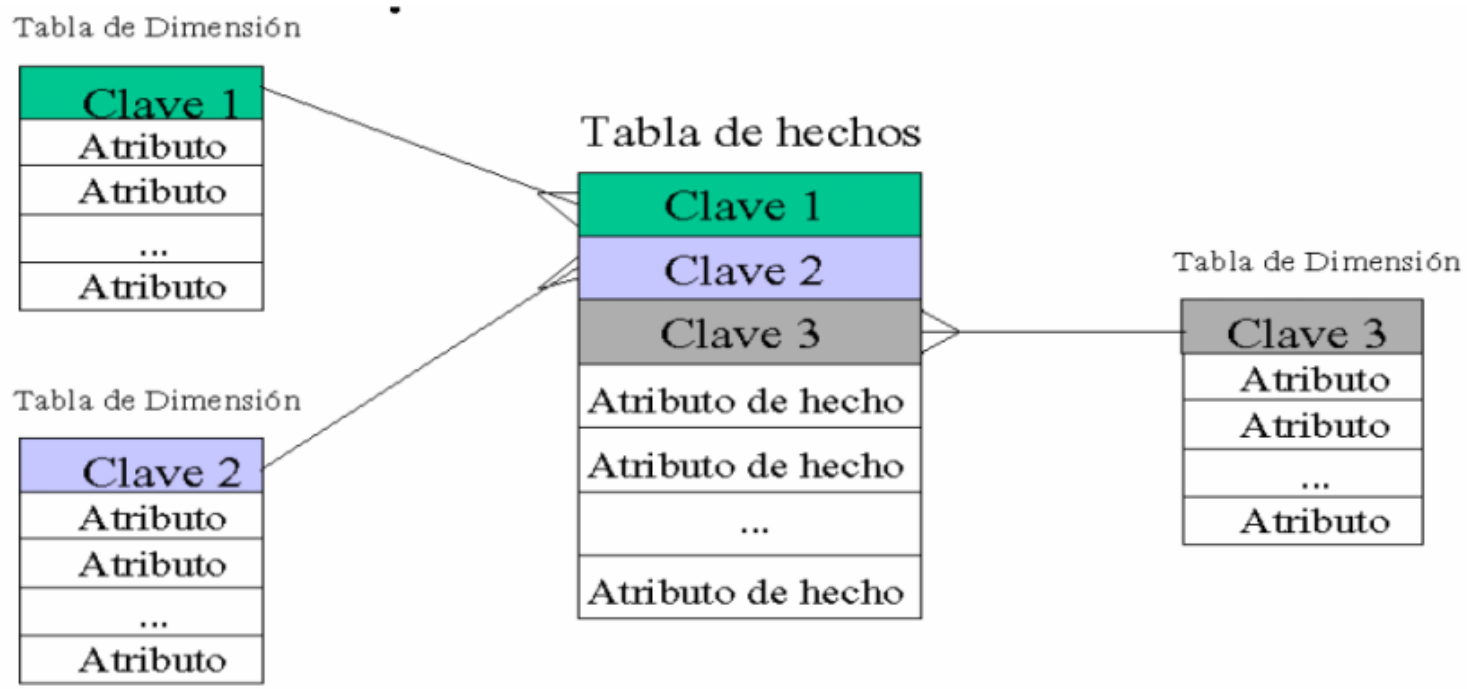
- Filas con pocas columnas (pocos atributos).
- Nº de filas: Desde millones a más de miles de millones (tantas como celdas tenga el cubo).
- Acceso, en general, vía dimensiones.

Tablas de Dimensión

Las características de las tablas de dimensión son:

- Definen las dimensiones de negocio en términos familiares para los usuarios.
- Filas con numerosas columnas de texto, altamente descriptivas.
- Normalmente menos de un millón de filas.
- Combinadas con las tablas de hecho mediante claves ajenas.
- Altamente indexadas.
- No están relacionadas entre sí.
- Se utilizan como puntos de acceso a los datos detallados de la tabla de hechos.
- A veces se tienen que desnormalizar.

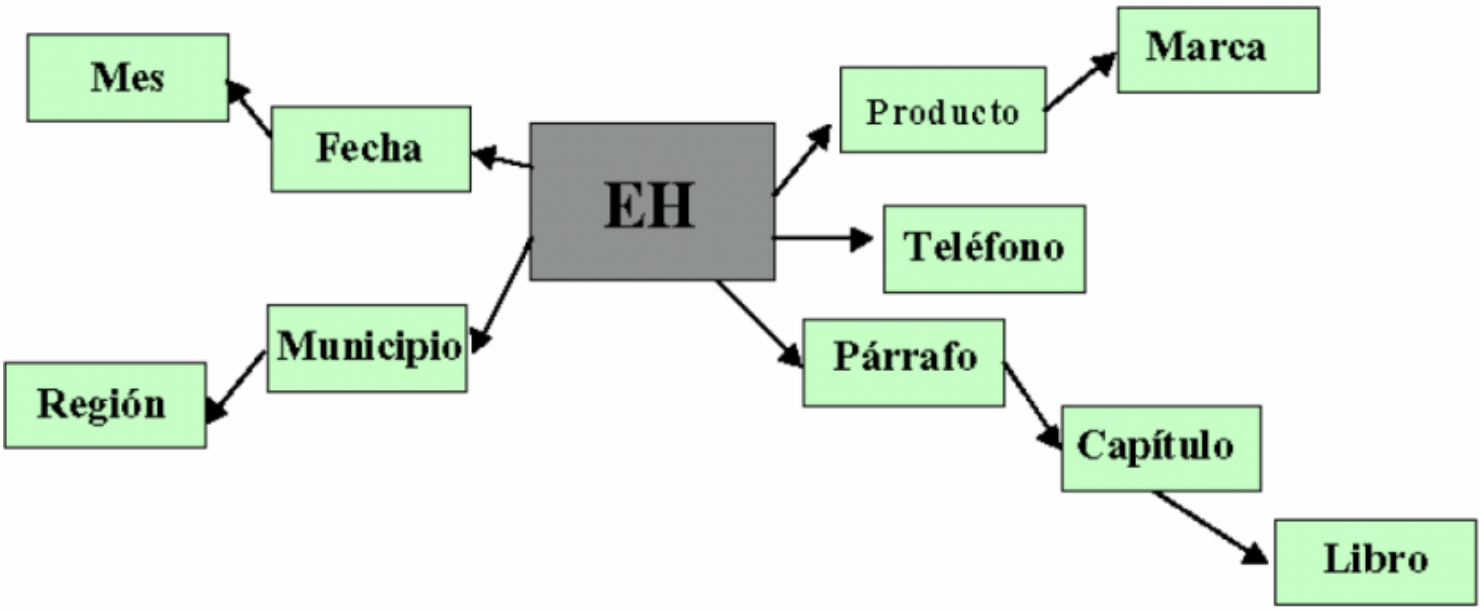
Figura de Tabla de Hechos con Tabla de Dimensiones



Al igual que sucede al manejar un hipercubo multidimensional, las consultas típicas en un esquema en estrella consisten en fijar un valor o rango de ellos para las dimensiones y, a continuación, obtener la información solicitada. La respuesta se encuentra realizando operaciones de unión natural (join) entre tablas de dimensiones y la tabla de hechos. Para optimizar las consultas, el gestor de BD debe ser capaz de reconocer que está trabajando con un esquema en estrella y hacer en primer lugar los join entre las tablas de dimensiones y, con el resultado, hacer un único join con la tabla de hechos, minimizando el número de accesos físicos.

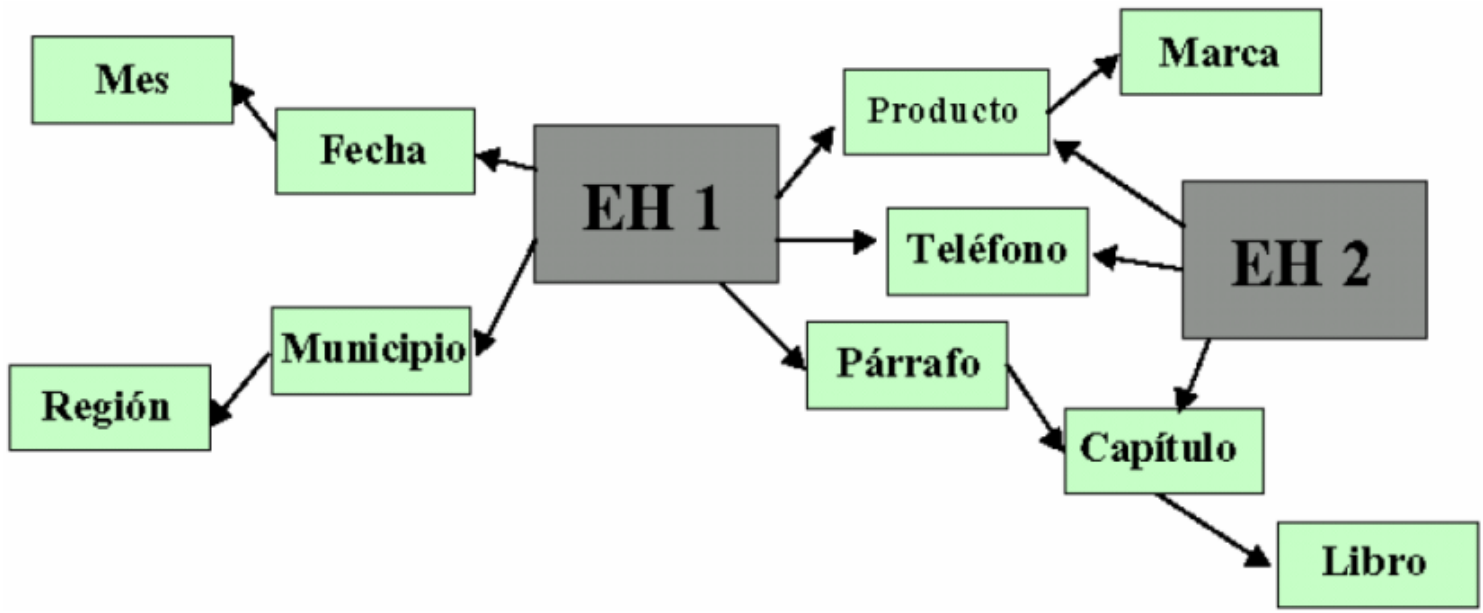
Esquema en Copo de Nieve

El esquema en copo de nieve es una variante del esquema en estrella que presenta las tablas de dimensión estructuradas a más de un nivel (tablas normalizadas). Se utiliza cuando hay jerarquías en las dimensiones, lo que supone más claves ajenas. Ejemplo:



Constelación de Estrellas

La constelación de estrellas la forman varios esquemas en estrella y/o en copo de nieve que comparten dimensiones. Ejemplo:



Índices Bitmap

Para poder conseguir una cierta eficiencia en los accesos, hay que considerar una serie de aspectos en el diseño físico, tales como:

- Estructuras de índices (mapas de bits, índices de combinación, índices textuales).
- Vistas materializadas:
  - Identificación de las vistas a materializar.
  - Explotación de la vista materializada durante la consulta.
  - Actualización de las vistas materializadas durante la carga y refresco.

En este apartado vamos a estudiar cómo es la estructura de los índices bitmap.

Los índices bitmap son un tipo especial de índice que almacena la información en bits en vez de múltiplos de bit (byte, doble byte) y que sirve para acelerar el acceso a filas con atributos de baja cardinalidad.

Se dice que un atributo es de baja cardinalidad si su dominio está formado por pocos elementos. Ejemplo: el atributo sexo (H o M).

Se trata de guardar un mapa de bits para cada posible valor del atributo, por lo que, como se dijo anteriormente, no es eficiente usar estos índices para valorares de alta cardinalidad. Ejemplo: el índice para sexo tendrá dos bitmaps.

Para responder a consultas que se realicen sobre esquemas relacionales con índices bitmap, basta con hacer las operaciones lógicas apropiadas (AND, OR, NOT) sobre los bits de cada índice implicado en la consulta, lo cual es una operación muy rápida, mucho más que la comparación de cadenas o números que implica la utilización de índices de otro tipo.

Este tipo de índices son útiles para indexar las tablas de dimensiones en esquemas en estrella o en copo de nieve, ya que muchas de estas dimensiones suelen tener su clave principal formada por un atributo de baja cardinalidad. Ejemplo: código de provincia, sexo, estado civil, etc.

Elección de una Herramienta OLAP

A la hora de elegir una herramienta OLAP hay que tener en cuenta, entre otros, los puntos siguientes:

- Si obliga a trabajar con una BD multidimensional (MOLAP), relacional (ROLAP) o si soporta ambas.
- En el caso de herramientas MOLAP es conveniente estudiar las capacidades de la BDM subyacente. Además hay que fijarse en su capacidad de aceptar accesos concurrentes y la carga de usuarios que admite, ya que el objetivo del OLAP es permitir el análisis interactivo.
- En el caso de herramientas ROLAP, la penalización en que se incurre al no utilizar BD multidimensional, y las facilidades que ofrece la herramienta para ofrecer una vista multidimensional de los datos (optimización de accesos a esquemas en estrella, en copo de nieve e índices bitmap).
- El límite en cuanto al número de dimensiones y de celdillas que puede manejar, sea o no multidimensional la BD subyacente. También la profundidad de los niveles de jerarquías y el manejo de series temporales.
- La capacidad de cálculo y la facilidad para especificar qué métodos y operaciones hay que aplicar a los datos. También debe disponer de herramientas y presentación de informes.
- El mantenimiento de las dimensiones y las jerarquías mediante herramientas automatizadas. Facilidad a la hora de modificar cualquiera de ambos elementos.

Comparativa de OLAP y Otros Sistemas

Terminamos el estudio de los sistemas OLAP haciendo una comparativa de los mismos frente a otros. Por un lado sistemas muy relacionados, como son los Sistemas de Soporte a las Decisiones y la propia Minería de Datos, y por otros, sistemas antagónicos como los sistemas OLTP.

Minería de Datos frente a OLAP y DSS

Los sistemas de ayuda a la decisión (DSS) son herramientas sobre las que se apoyan los responsables de una empresa, directivos y gestores, en la toma de decisiones. Para ello, utilizan:

- Un Data Warehouse, en el que se almacena la información de interés para la empresa.

- Herramientas de análisis multidimensional (OLAP).

Los DSS permiten al responsable de la toma de decisiones consultar y utilizar de manera rápida y económica las enormes cantidades de datos operacionales y de mercado que se generan en una empresa. Gracias al análisis OLAP, pueden verificarse hipótesis y resolverse consultas complejas. Además, en el curso del análisis, la interpretación de los datos puede dar lugar a nuevas ideas y enfoques del problema, sugiriendo nuevas posibilidades de análisis.

Sin embargo, el análisis OLAP depende de un usuario que plantee una consulta o hipótesis. Es el usuario el que lo dirige y, por tanto, el análisis queda limitado por las ideas preconcebidas que aquél pueda tener.

La minería de datos constituye un paso más en el análisis de los datos de la empresa para apoyar la toma de decisiones. No se trata de un técnica que sustituya los DSS ni el análisis OLAP, sino que los complementa, permitiendo realizar un análisis más avanzado de los datos y extraer más información de ellos.

Como ya se ha comentado anteriormente, utilizando minería de datos es el propio sistema el que descubre nuevas hipótesis y relaciones. De este modo, el conocimiento obtenido con estas técnicas no queda limitado por la visión que el usuario tiene del problema.

Las diferencias entre minería de datos y OLAP radican esencialmente en que el enfoque desde el que se aborda el análisis con cada una de ellas es completamente distinto. Fundamentalmente:

- El análisis que realizan las herramientas OLAP es dirigido por el usuario, deductivo, parte de una hipótesis o de una pregunta del usuario y se analizan los datos para resolver esa consulta concreta. Por el contrario, la minería de datos permite razonar de forma inductiva a partir de los datos para llegar a una hipótesis general.
- Además, las aplicaciones OLAP trabajan generalmente con datos agregados, para obtener una visión global del negocio. Por el contrario, la minería de datos trabaja con datos individuales, concretos, descubriendo las regularidades y patrones que presentan entre sí y generalizando a partir de ellos.

Un ejemplo clarificará la diferencia entre ambas técnicas es el siguiente:

Una pregunta típica de un sistema OLAP/DSS sería: “El año pasado, ¿se compraron más furgonetas en Cataluña o en Madrid?”. La respuesta de sistema sería del tipo “En Cataluña se compraron 12.000 furgonetas, mientras que, durante el mismo intervalo, en Madrid se compraron 10.000”. Obviamente es una información interesante y útil, pero restringida por la hipótesis realizada a priori.

En cambio, un problema típico para resolver utilizando minería de datos sería, por ejemplo: “Hallar un modelo que determine las características más relevantes de las personas que compran furgonetas”. A partir de los datos del pasado, el sistema de minería de datos proporcionaría una respuesta del tipo: “Depende de la época del año y la situación geográfica. En invierno, los habitantes de Madrid que pertenecen a un cierto grupo de edad y nivel de ingresos probablemente comprarán más furgonetas que gente de las mismas características en Cataluña”.

Como puede verse, se trata de problemas distintos, de modo que según los objetivos perseguidos deberá utilizarse una técnica u otra. Además, puesto que sus conclusiones son complementarias, en general será conveniente combinar ambas para obtener los mejores resultados.

Sistemas OLTP vs Sistemas OLAP

Como ya sabemos OLAP (On-Line Analytical Processing) se define como análisis rápido de información multidimensional compartida. El término OLAP aparece en contraposición al concepto tradicional OLTP (On-Line Transactional Processing), de designa el procesamiento operacional de los datos, orientado a conseguir la máxima eficacia y rapidez en las transacciones (actualizaciones) individuales de los datos, y no su análisis de forma agregada.

Existen, por lo tanto, dos grupos de aplicaciones que se realizan en una empresa:

- Aplicaciones que ejecutan operaciones del día a día (compra, inventario, nóminas, …). Son los Sistemas de Procesamiento de transacciones en línea (OLTP).
- Aplicaciones que se encargan de analizar el negocio, interpretar lo que ha ocurrido y tomar decisiones (para mejorar los servicios al cliente, incrementar ventas,…). Son los Sistemas de Procesamiento analítico en línea (OLAP).

Los dos son sistemas de procesamiento muy diferentes. Veamos las diferencias principales entre los dos sistemas:

- OLAP permite que una compañía decida qué debe hacer y OLTP ayuda a llevar a cabo la decisión.
- OLTP representa una “imagen” de los asuntos de la organización que se actualiza constantemente (con cada operación realizada). Los sistemas OLAP son estáticos, refrescándose periódicamente (cada semana, cada mes, …) a partir de las fuentes OLTP.
- El diseño de los sistemas OLTP elimina redundancias, y se piensa más en la eficiencia (transacciones rápidas) que en el usuario (dificultad para navegar). Los sistemas OLAP almacenan datos redundantes para conseguir un acceso sencillo al usuario y buenos tiempos de respuesta.
- OLTP proporciona capacidades muy limitadas para la toma de decisiones (los usuarios examinan la BD registro a registro). OLAP trabaja con un resumen de miles de registros “condensados” en una respuesta.
- Los sistemas transaccionales (u operacionales) automatizan el día a día del negocio, buscando la eficiencia. Los sistemas analíticos se centran en la estrategia a largo plazo y están dirigidos por el negocio.
- En cuanto a la implementación de OLTP y OLAP:
  - Surgen los sistemas EIS y DSS (basados en OLAP) para soportar la toma de decisiones. Presentan problemas para recuperar datos de la BD Operacionales.
  - No se puede implementar OLTP y OLAP en una sola BD. Actuando el SGBD como interfaz entre datos y usuarios.
  - Se necesita una arquitectura dual de BD.

En el siguiente cuadro, se observa de forma resumida, las características de los sistemas OLTP y OLAP, quedando así más claras sus diferencias.

	OLTP	**OLAP**
Usuario típico	*Administrativo*	Gestor o Directivo
Usuario del sistema	*Ejecución del negocio*	*Análisis del negocio*
Interacción de usuario	*Predeterminado*	*Ad-hoc*
Unidad de trabajo	*Transacción*	*Consulta*
Característica de trabajo	*Lectura/Escritura*	*Principalmente lectura*
Registros accedidos	*Cientos*	*Millones*
Número de usuarios	*Miles*	*Cientos*
Focos	*Entrada de datos*	*Salida de información*

Big Data

Con la irrupción de internet, llegaron nuevos conceptos que con el tiempo se han vuelto de uso cotidiano y que nos acompañan en nuestro día a día. Han repercutido para bien en nuestras vidas y casi no podemos entender las nuevas tecnologías sin estas geniales ideas. Uno de estos conceptos que han resonado mucho últimamente es **Big Data**; aunque como ya ha pasado en anteriores ocasiones, el halo de escepticismo y desconfianza ha planeado en torno a todo lo que lo rodea. Hay muchas dudas (fundadas) en cuanto a su concepto, uso y alcance; de esta manera se crea un ambiente de recelo aparejado a algo que parece intangible, incontrolable y sobre todo, que puede atentar nuestra privacidad.

Qué significa Big Data

Big Data (*datos masivos* en español, aunque apenas se utiliza la traducción) es el proceso de recolección de grades cantidades de datos y su inmediato análisis para encontrar información oculta, patrones recurrentes, nuevas correlaciones, etc; el conjunto de datos es tan grande y complejo que los medios tradicionales de procesamiento son ineficaces. Y es que estamos hablando de desafíos como analizar, capturar, recolectar, buscar, compartir, almacenar, transferir, visualizar, etc, ingentes cantidades de información, obtener conocimiento en tiempo real y poner todos los sentidos en la protección de datos personales. El tamaño para albergar todo el proceso ha ido aumentando constantemente para poder recopilar e integrar toda la información.

La recolección de datos ha existido casi desde siempre, cuando en el amanecer el hombre hacía muescas en piedras o huesos para hacer seguimiento de las actividades cotidianas o de los suministros esenciales para subsistir. La invención de ábaco supuso un determinante empuje al cálculo y análisis que tanto necesitábamos cuando los dedos y la memoria no eran suficientes, y las primeras bibliotecas representaron además un primer intento de almacenar datos. En la época actual, todo lo que hacemos está continuamente dejando un rastro digital que se puede utilizar y analizar; los avances en tecnología, junto a la expansión de Internet y el almacenamiento en la nube, han provocado que crezca la cantidad de datos que podemos almacenar.

Para resumir, se puede utilizar **5 V’s** como definición de **Big Data** (empezaron siendo 3), que es lo que caracteriza al sistema y al mismo tiempo explica sus ventajas:

1. **Volumen**. La más evidente y la que hace honor al nombre; captar y organizar absolutamente toda la información que nos llega es esencial para tener registros completos e insesgados, y que las conclusiones que obtengamos sirvan eficientemente a la hora de la toma de decisiones. Es el Business Intelligence que todos conocemos, pero a lo grande; aunque la diferencia con la clásica inteligencia de negocio viene marcada por el resto de V’s.
2. **Velocidad**. Siempre es importante el tiempo si afrontamos tanto la necesidad de generar información (y recordemos que estamos hablando de muchos datos) como de analizarla, pero lo es más si necesitamos reaccionar inmediatamente; todo el proceso pide agilidad para extraer valor de negocio a la información que se estudia y que no se pierda la oportunidad.
3. **Variedad**. Hay que dar uniformidad a toda la información, que tendrá su origen en datos de lo más heterogéneos, tal como veremos en el siguiente apartado. Una de las fortalezas del **Big Data** reside en poder conjugar y combinar cada tipo de información y su tratamiento específico para alcanzar un todo homogéneo.
4. **Veracidad**. Se refiere a la calidad del dato y su disponibilidad; en un entorno descrito por la anterior *V, Variedad*, hay que encontrar herramientas para comprobar la información recibida; las tecnologías creadas al servicio del **Big Data** se muestran imprescindibles y eficientes para afrontar los retos.

5. **Valor**. Trabajar con **Big Data** tiene que servier para aportar valor a la sociedad, las empresas, los gobiernos, en definitiva, a las personas; todo el proceso tiene que ayudar a impulsar el desarrollo, la innovación y la competitividad, pero también mejorar la calidad de vida de las personas.

Tipos de datos en Big Data

Para aclarar qué es lo que se recoge para el análisis, podemos dividirlos en dos grandes categorías:

- **Datos estructurados**. Aquellos que tienen longitud y formato (por ejemplo fechas) y que pueden ser almacenados en tablas (como las BDR). En esta categoría entran los que se compilan en los censos de población, los diferentes tipos de encuestas, los datos de transacciones bancarias, las compras en tiendas online, etc.
- **Datos no estructurados**. Son los que carecen de un formato determinado y no pueden ser almacenados en una tabla. Pueden ser de *tipo texto* (los que generan los usuarios de los foros, redes sociales, documentos de Word), y los de *tipo no-texto* (cualquier fichero de imagen, audio, vídeo). Dentro de esta categoría, podemos añadir los **Datos semiestructurados**, que son los que no pertenecen a BDR ya que no se limitan a campos determinados, aunque poseen organización interna o marcadores que facilita el tratamiento de sus elementos; estaríamos hablando de documentos XML, HTML o los datos almacenados en BD NoSQL.

El uso del análisis de datos

Para poder analizar todo esto, se precisa de técnicas potentes y avanzada; las clásicas medias o varianzas no son por sí solas suficientes para extraer toda esa cantidad de información, ni para entender los diferentes tipos de datos que hemos descrito.

Antes de la irrupción **Big Data**, ya existían algoritmos matemáticos que nos facilitaban descubrir información oculta en los datos, como todos los que engloban el **Data Mining** (minería de datos): k-medias, árboles de decisión, redes neuronales, etc, que con la llegada de la potencia de cálculo de los ordenadores permitieron acortar el tiempo que se tardaba en obtener resultados. Aunque no se pensó para ser en tiempo real si no a posteriori, permite analizar datos para encontrar correlaciones entre ellos y de este modo desarrollar por ejemplo una estrategia de marketing adaptada a las conclusiones.

Por eso el análisis de datos siempre ha tenido un gran peso en el marketing, un mejor conocimiento del consumidor y sus necesidades propicia saber cómo aumentar las ventas; el análisis de datos nos permite establecer relaciones entre variables, predecir comportamientos, realizar agrupaciones (clustering) de grupos homogéneos, e incluso analizar textos para extraer información. Ahora con **Big Data**, todo esto se consigue en tiempo real y con cada nueva actualización de nuestro repositorio de datos es posible ver los cambios en las estadísticas inmediatamente.

Qué utilidad puede tener

Como todas las cosas en esta vida, puede tener un buen uso o usarse para propósitos “malvados”. Lo primero que llama la atención es el tema de la privacidad, ya que cada vez más detalles de nuestras vidas son almacenados y analizados por empresas y gobiernos; por supuesto, no es algo que nos debamos tomar a la ligera, pero a medida que siga avanzando la tecnología, habrá que ir adaptando las leyes y regulaciones para proteger a las personas. Por ahora, no hay más rastro de nosotros que los que ya estamos dejando día a día, y que ya están siendo analizados por terceros; a partir de este momento, todos esos registros se unen para formar un todo. Sí, podemos hablar de una representación de nosotros, pero no deja de ser un número entre millones de números, sin cara ni alma. Lo

único que va a contar para estudiar es el comportamiento de grupos homogéneos tratados como tendencias en un segundo, para que al siguiente empiece de nuevo el proceso.

En cambio los beneficios son muchos, y muy importantes. Veamos ejemplos.

- Una eCommerce puede optimizar el stock de sus almacenes a través de la información extraída de lo que busca la gente en su web o analizando las tendencias en redes sociales y foros; también fijar precios dinámicos en sus productos extrayendo datos de múltiples fuentes (las acciones de los clientes, preferencias de los proveedores o recopilación de precios de la competencia).
- El sector de las telecomunicaciones es una industria privilegiada, gracias a sus redes y a la proliferación de dispositivos móviles; la oportunidad más evidente es extraer información de la experiencia del usuario gracias al tráfico de voz y datos, y así poder ofrecer altas en contratos personalizados, ampliar la batalla por la competencia e incluso crear nuevas fuentes de ingresos.
- La banca tiene ante si un reto, y una oportunidad, de poner medios para luchar contra el fraude, los delitos financieros y las brechas de seguridad, mediante **Big Data**. Las entidades financieras están invirtiendo enormes cantidades de dinero en perfeccionar algoritmos y la tecnología de análisis para minimizar riesgos y fortalecer su imagen de cara al cliente.
- La Federación Alemana de Fútbol empezó a usar el análisis de grandes volúmenes de datos para mejorar el rendimiento de sus jugadores, y con los deberes bien hechos se presentaron en el Mundial de Brasil 2014.
- Si piensas que todo lo que puede dar de sí **Big Data** es sólo aprovechable por grandes corporaciones, vas mal encaminado; por ejemplo, las fuerzas de seguridad utilizan estas herramientas para perseguir criminales y luchar contra el terrorismo de cualquier tipo. En materia de sanidad, el cruce de información de historiales clínicos, antecedentes familiares, clima y entorno, junto a los hábitos de consumo, permitirá un modelo predictivo personal para cada paciente, y de esta manera ayudar en la detención precoz de enfermedades y estrategias más efectivas para combatirlas. En muchas ciudades, ya se usa el análisis de datos para transformarse en más modernas e inteligentes: transportes públicos interconectados para minimizar los tiempos de espera, o semáforos que ante la previsión de un aumento del tráfico e regulan para minimizar los atascos.
- Y por supuesto, las pymes también pueden subirse al carro del **Big Data**, ya que no es necesaria una gran inversión. Es suficiente con tener un CRM y a un analista de datos para extraer conclusiones de la información que utiliza una pyme, aunque siempre cabe la posibilidad de externalizarlo.

Big Data, modelando el futuro

Todo el mundo habla cada día más, es una tendencia en aumento y ha llegado para quedarse. A medida que las herramientas se hagan más accesibles, se integrará poco a poco en nuestras vidas y pasará de ser algo desconocido o temido, a una forma más de comprender el comportamiento humano y nuestra relación con el entorno.

Es como el Social Media, al principio las empresas lo veía como algo ajeno a ellas, que no debían destinar recursos porque creían que no reportaría ningún beneficio; ahora, lo más normal es hacer *Social Marketing* y elaborar informes exhaustivos con las estadísticas derivadas de su presencia online. Pues ahora es el momento de cruzar esos datos con el resto de aspectos de la organización, como ventas, tráfico web, interacción con distribuidores, etc, para encontrar nuevas vías de negocio y crear nuevas estrategias.

Y por supuesto, para analizar toda esta información, es necesario contar con profesionales que tengan parte analista y parte creativa; estos "*científicos de datos*" serán muy demandados por las empresas y organizaciones, por lo que se abre un interesantísimo campo laboral para los amantes de los números.

Bibliografía

- Scribd (Roger Fabian Molina)
- MiBloguel