



Eusko Jurlaritzaren
Informatika Elkarte

Sociedad Informática
del Gobierno Vasco

ARINbide

Conceptos Básicos y Técnicas



[ARINbide](#) by [EJIE](#) is licensed under a [Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported License](#).

Versión	Fecha	Resumen de cambios	Elaborado por:	Aprobado por:
1.0	3/11/2016	Se crea este documento independiente del módulo de ARINbide.	CAC	

Índice

1	Introducción	1
2	Técnicas	2
2.1	Sesiones de trabajo	2
2.2	Casos de uso	5
2.3	Diagrama de paquetes.....	8
2.4	Diagrama de secuencia	9
2.5	Diagrama de clases	11
2.6	Prototipado.....	15
2.7	Análisis de consistencia	17
2.8	Diagrama de representación.....	17
2.9	Diagrama de estructura	18
2.10	Reglas de transformación	21
2.11	Caminos de acceso.....	23
2.12	Diagrama de componentes	23
2.13	Diagrama de despliegue.....	24
2.14	Identificación de la configuración	25
2.15	Control de cambios	30
2.16	Gestión de bibliotecas	30

1 Introducción

ARINbide es una metodología de ingeniería de software adaptada a los servicios que EJIE S.A presta al Gobierno Vasco. Está basada en MÉTRICA Versión 3 para desarrollos orientados a objetos pero simplificada en base al principio de que el resultado obtenido recoja toda la información necesaria y solo la suficiente para el adecuado diseño y desarrollo del sistema.

En general, las fases y actividades descritas pueden ser ampliadas con el material metodológico de MÉTRICA Versión 3, para lo cual se ha conservado en esta adaptación la codificación original de las mismas.

2 Técnicas

2.1 Sesiones de trabajo

Las sesiones de trabajo tienen diversos objetivos. Dependiendo del tipo de sesión que se realice, los objetivos pueden ser: obtener información, comunicar resultados, reducir el tiempo de desarrollo, activar la participación de usuarios y directivos o aumentar la calidad de los productos.

Las sesiones de trabajo pueden ser de varios tipos en función de las personas que participen en ellas, el objetivo que se persiga y el modo de llevarlas a cabo.

Dentro de estas sesiones de trabajo se encuentran algunas técnicas como son el JAD (Joint Application Design) y el JRP (Joint Requirements Planning), y otras prácticas como las entrevistas y las reuniones, en las que se pueden dar algunas orientaciones y recomendaciones para su realización.

A continuación se explica brevemente el objetivo principal de cada una de ellas, antes de describir más en detalle la forma de llevarlas a cabo.

- Las entrevistas son un tipo de sesiones de trabajo dirigidas a obtener la información de una forma individual dónde aparecen los perfiles de entrevistado y entrevistador.
- Las reuniones pueden tener el mismo objetivo, pero la información está dispersa entre varias personas y únicamente trabajando en grupo, se conseguirá extraer y depurar toda la información de forma global.
- Las sesiones JAD y JRP son reuniones en las que se potencia el trabajo en equipo entre el cliente o usuario y el proveedor, con una participación más activa del cliente en los diferentes procesos del ciclo de vida que va a permitir identificar las necesidades planteadas, proponer soluciones, negociar enfoques diferentes y especificar el conjunto preliminar de requisitos que debe cumplir la solución para llegar al objetivo que se propone. Estas técnicas surgieron en el ámbito de un ciclo de vida de desarrollo rápido, pero en MÉTRICA Versión 3, se proponen independientemente del ciclo de vida, como medio para alcanzar una mayor productividad en las sesiones de trabajo.

2.1.1 Entrevistas

Las entrevistas constituyen un medio para obtener la información que se necesita sobre un determinado tema, como puede ser, el establecer el alcance de un problema, identificar los requisitos a cubrir por un sistema de información y analizar el funcionamiento de un sistema actual, entre otros, a partir de las personas que tienen conocimiento sobre el mismo.

Se entiende por entrevista el encuentro que se realiza “cara a cara” entre un usuario y la persona responsable de obtener la información.

Para realizar la entrevista solo es necesario designar a las personas que deben participar en ella y determinar el lugar en el que poder llevarla a cabo. Es importante identificar a qué tipo de perfil va dirigida la entrevista, a quiénes se va a entrevistar y cuál es el momento más oportuno, con el fin de evitar situaciones embarazosas y conseguir que la entrevista sea eficaz y productiva.

Como paso previo a la realización de la entrevista se deben tener en cuenta una serie de reglas generales o directrices básicas:

- Desarrollar un plan global de la entrevista.
- Asegurarse de que se cuenta con la aprobación para hablar con los usuarios.
- Preparar la entrevista previamente.
- Realizar la entrevista.
- Consolidar el resultado de la entrevista.

Además, es conveniente planificar las entrevistas estudiando la secuencia en que se van a llevar a cabo, en función de los distintos perfiles implicados y las relaciones existentes entre los entrevistados. Según la información a obtener y dependiendo de las distintas fuentes que pueden proporcionarla, puede ser necesario realizar una entrevista conjunta con varias personas.

Durante la preparación de la entrevista es imprescindible remitir al usuario un guión previo sobre los puntos a tratar, para que pueda estudiarlo con tiempo y solicitar la información que estime conveniente para la entrevista. Se debe pensar bien el tipo de guión, según el perfil y las responsabilidades del entrevistado y su extensión, de

forma que se pueda conseguir la suficiente información, sin provocar rechazo en el entrevistado. Si se considera apropiado se pueden utilizar herramientas automatizadas.

Una vez que se dispone de la aprobación para hablar con los usuarios, se hace la convocatoria de la entrevista enviando la información oportuna y fijando los objetivos, el método de trabajo que se va a seguir y el tiempo del que se dispone.

Para realizar la entrevista, es importante hacer un resumen general de los temas a tratar, utilizar un estilo apropiado y crear desde su inicio un clima de confianza entre los asistentes. Es posible que el entrevistado se resista a aportar información, siendo útil en estos casos utilizar técnicas específicas de comunicación.

Antes de finalizar la entrevista es importante que el entrevistador sintetice las conclusiones y compruebe que todos los asistentes están de acuerdo, dejando siempre abierta la posibilidad de volver a contactar para aclarar temas que surjan al estudiar la información recopilada.

Finalmente, el responsable depura y consolida el resultado de las entrevistas, elaborando un informe de conclusiones. En algunos casos puede ser conveniente elaborar un acta que refleje estas conclusiones y remitirla a los entrevistados con el objetivo de asegurar que se han comprendido bien las especificaciones dadas.

2.1.2 Reuniones

Las reuniones tienen como objetivo obtener información que se encuentra repartida entre varias personas, tomar decisiones estratégicas, tácticas u operativas, transmitir ideas sobre un determinado tema, analizar nuevas necesidades de información, así como comunicar los resultados obtenidos como consecuencia de un estudio.

Para realizar una reunión es necesario designar a las personas que deben participar en ella y determinar el lugar en el que poder llevarla a cabo. Las directrices básicas de una reunión son:

- Preparar y convocar la reunión (orden del día).
- Realizar la reunión.
- Consolidar el resultado de la reunión.
- Elaborar el acta de reunión.

Previamente a la convocatoria de la reunión, se definen los objetivos, se planifica el método de trabajo que se va a seguir y el tiempo del que se dispone, se eligen los participantes y se prepara el material necesario.

Después de la preparación, es imprescindible enviar al usuario la convocatoria con el orden del día de la reunión. Este orden incluye la fecha, hora de inicio, hora de finalización prevista, lugar, asistentes y los puntos a tratar, detallando, entre otros, el tiempo que se dedicará a cada tema y la persona responsable de exponerlo. Dicha convocatoria se envía con antelación suficiente para que los asistentes puedan organizar su agenda y prepararse para la reunión con tiempo.

Al inicio de la reunión, es importante hacer un resumen general de los temas a tratar, los objetivos que se persiguen, el método de trabajo y la agenda de la reunión. Si se considera oportuno se puede utilizar la técnica de presentación. Desde su inicio se debe crear un clima de confianza entre los asistentes.

Uno de los primeros puntos recomendables es la revisión del acta anterior y del estado de las tareas señaladas, si existen.

La persona responsable de la reunión ejercita la dinámica de dirección de grupos, estimulando la participación, controlando el ritmo de la sesión y centrando o clarificando el tema cuando sea necesario. Al finalizar, se sintetizan las conclusiones, se comprueba si hay acuerdo o si quedan puntos pendientes de reflexión y se propone fechas para próximas reuniones.

El responsable de tomar las notas en la reunión, levanta el acta y la remite a los asistentes que deben confirmar su recepción.

Cuando el número de tareas sea muy elevado puede ser conveniente llevar un documento paralelo para el control de las tareas pendientes. Este documento no debe sustituir la incorporación en el acta de las actividades acordadas.

También es conveniente que se cree una entrada en el cuaderno de bitácora del proyecto con los temas más relevantes y un vínculo al acta, de forma que pueda localizarse y referenciarse el documento correspondiente.

2.1.3 JAD (Joint Application Design)

Las sesiones JAD tienen como objetivo reducir el tiempo de desarrollo de un sistema manteniendo la calidad del mismo. Para ello se involucra a los usuarios a lo largo de todo el desarrollo del sistema, es decir, desde la

identificación de la necesidad, la propuesta de alternativas de solución y sobre todo en la especificación de los requisitos que debe cubrir el sistema y en la validación de prototipos.

- Las características de una sesión de trabajo tipo JAD se pueden resumir en los siguientes puntos:
- Se establece un equipo de trabajo cuyos componentes y responsabilidades están perfectamente identificados y su fin es conseguir el consenso entre las necesidades de los usuarios y los servicios del sistema en producción.
- Se llevan a cabo pocas reuniones, de larga duración y muy bien preparadas.
- Durante la propia sesión se elaboran los modelos empleando diagramas fáciles de entender y mantener, directamente sobre herramientas CASE.
- Al finalizar la sesión se obtienen un conjunto de modelos que deberán ser aprobados por los participantes.

Es importante definir claramente el perfil y las responsabilidades de los participantes de una sesión JAD. Se pueden distinguir los siguientes perfiles:

- Moderador (líder JAD) con amplios conocimientos de la metodología de trabajo, dinámica de grupos, psicología del comportamiento, así como de los procesos de la organización objeto del estudio.
- Promotor, persona que ha impulsado el desarrollo.
- Jefe de proyecto, responsable de la implantación del proyecto.
- Especialista en modelización, responsable de la elaboración de los modelos en el transcurso de la sesión.
- Desarrolladores, aseguran que los modelos son correctos y responden a los requisitos especificados.
- Usuarios, responsables de definir los requisitos del sistema y validarlos.

La sala en la que se llevarán a cabo este tipo de sesiones juega un papel muy importante debido a que, en las reuniones largas donde se requiere una alta concentración de todos los integrantes del equipo, es necesario prestar especial atención a aspectos relacionados con la temperatura, los medios audiovisuales, la buena visibilidad de los distintos participantes, etc.

Para llevar a cabo una sesión JAD, es necesario realizar una serie de actividades antes de su inicio, durante el desarrollo y después de su finalización. Estas actividades se detallan a continuación:

- Inicio: se define el ámbito y la estructura del proyecto, los productos a obtener, se prepara el material necesario para la sesión, se determina el lugar donde se van a llevar a cabo, se seleccionan los participantes y se sugiere una agenda de trabajo.
- Desarrollo: se identifican las salidas del proyecto y se debe conseguir el consenso entre los participantes de modo que se materialice en los modelos.
- Finalización: se valida la información de la sesión y se generan los productos de la metodología de trabajo propuesta. Si fuera necesario se integran los productos de salida.

En las sesiones de trabajo tipo JAD se distinguen dos tipos de productos:

- De preparación donde se incluye, entre otros, la historia y contexto del proyecto, los objetivos y límites, las actividades del entorno del negocio que pueden afectar al éxito del proyecto y los beneficios.
- De resultado de las sesiones de trabajo, que se establecen con anterioridad al inicio de las reuniones.

2.1.4 JRP (Joint Requirements Planning)

Las sesiones JRP tienen como objetivo potenciar la participación activa de la alta dirección como medio para obtener los mejores resultados en el menor tiempo posible y con una mayor calidad de los productos.

Las características de las sesiones JRP y JAD son comunes en cuanto a la dinámica del desarrollo de las sesiones y la obtención de los modelos con el soporte de las herramientas adecuadas. La diferencia radica en los productos de salida y en los perfiles de los participantes.

En JRP son del nivel más alto en la organización en cuanto a visión global del negocio y capacidad de decisión.

Los perfiles implicados en una sesión JRP son los siguientes:

- Moderador (líder JRP), debe tener una gran capacidad de relación, habilidades de negociación y de gestión de dinámica de grupos, así como un alto nivel de conocimiento de los procesos de la organización afectados por el Plan de Sistemas de Información (PSI).
- Promotor, persona que ha impulsado el Plan de Sistemas de Información y tiene un compromiso económico.

- Director de proyecto, responsable de que el proyecto llegue a buen fin.
- Consultores, responsable de traducir los requisitos especificados por el usuario en información estructurada, de tal forma, que los usuarios puedan entender y discutir los resultados.
- Especialista en modelización, responsable de la elaboración de los modelos en el transcurso de la sesión.
- Usuarios de alto nivel, responsables de definir los procesos de la organización y los sistemas de información afectados por el Plan de Sistemas de Información así como las prioridades para su implantación a largo o medio plazo en la organización.

La sala de reuniones juega un papel muy importante, siendo necesario acondicionarla con el fin de facilitar el desarrollo de la sesión. Se debe prestar especial atención a aspectos como la temperatura, la luz, su orientación, la distribución de los participantes en la sala, etc. Hay que asegurar que se cuenta con los medios audiovisuales adecuados como pueden ser cámara de vídeo y apuntadores láser, entre otros.

Para llevar a cabo una sesión JRP, es necesario realizar una serie de actividades:

- Iniciación, se establece la necesidad del Plan de Sistemas de Información (PSI), su alcance, los procesos de negocio implicados, las unidades organizativas afectadas, así como los usuarios clave y los perfiles del equipo JRP.
- Búsqueda, se identifican los objetivos del Plan de Sistemas de Información, se estudia la situación actual y se busca la información relevante, que pueda ser útil.
- Preparación, se seleccionan los participantes, se prepara el material necesario, acondicionando también la sala, y se establece la agenda de JRP.
- Realización: se introduce la reunión y se empieza a trabajar en la consecución de los objetivos marcados en la agenda, elaborando los productos objeto de la sesión.
- Finalización: se completan los productos y se presenta a los participantes que corresponda.

La información de salida que se obtiene al finalizar una sesión JRP, dependerá de la actividad que se esté realizando, como por ejemplo:

- Modelos de procesos de la organización.
- Modelo de información.
- Modelo de sistemas de información, etc.

2.2 Casos de uso

Los objetivos de los casos de uso son los siguientes:

- Capturar los requisitos funcionales del sistema y expresarlos desde el punto de vista del usuario.
- Guiar todo el proceso de desarrollo del sistema de información.

Los casos de uso proporcionan, por tanto, un modo claro y preciso de comunicación entre cliente y desarrollador. Desde el punto de vista del cliente proporcionan una visión de “caja negra” del sistema, esto es, cómo aparece el sistema desde el exterior sin necesidad de entrar en los detalles de su construcción. Para los desarrolladores, suponen el punto de partida y el eje sobre el que se apoya todo el desarrollo del sistema en sus procesos de análisis y diseño.

2.2.1 Descripción

Un caso de uso es una secuencia de acciones realizadas por el sistema, que producen un resultado observable y valioso para un usuario en particular, es decir, representa el comportamiento del sistema con el fin de dar respuestas a los usuarios.

Aquellos casos de uso que resulten demasiado complejos se pueden descomponer en un segundo nivel, en el que los nuevos casos de uso que intervengan resulten más sencillos y manejables.

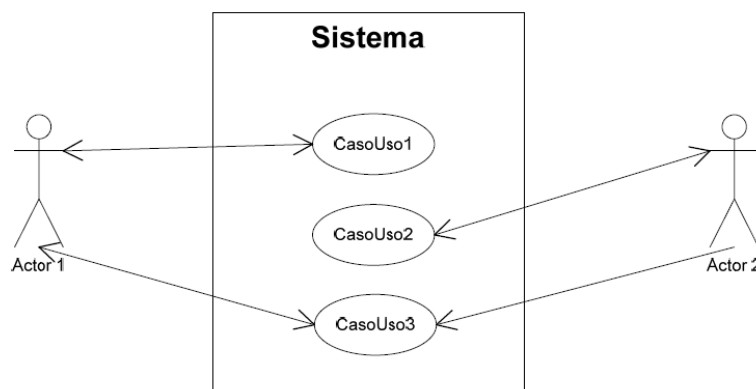
Para especificar este comportamiento existen una serie de recomendaciones o técnicas que se aplican dependiendo del momento del desarrollo que se esté y de la complejidad del caso de uso. Puede ser desde una simple descripción textual que recoja un requisito funcional a una especificación del caso de uso, e incluso un conjunto de diagramas:

2.2.2 Especificación de un caso de uso

Un caso de uso recoge, en un primer momento, una descripción general. Esta descripción reflejará posiblemente uno o varios requisitos funcionales del sistema o formará parte de algún requisito.

Se puede completar la descripción definiendo cuáles son las precondiciones y postcondiciones del sistema, es decir, qué condiciones deben cumplirse para que se realice un caso de uso y cuáles son aquellas condiciones que se cumplen posteriormente al caso de uso.

También se pueden enumerar los diferentes escenarios del caso de uso si los tuviese y dar una breve descripción de ellos. Los escenarios son los distintos caminos por los que puede evolucionar un caso de uso, dependiendo de las condiciones que se van dando en su realización.



2.2.3 Diagrama de casos de uso

Estos diagramas presentan dos tipos de elementos fundamentales:

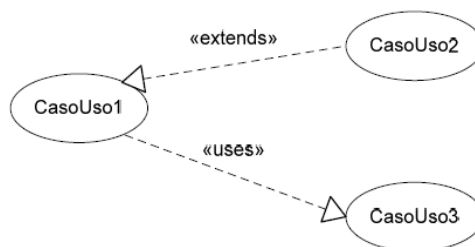
- **Actores.** Un actor es algo o alguien que se encuentra fuera del sistema y que interactúa con él. En general, los actores serán los usuarios del sistema y los sistemas externos al que se esté desarrollando. Si se habla de usuarios, un actor es el papel que puede llevar a cabo en cuanto a su forma de interactuar con el sistema, es decir, un único actor puede representar a muchos usuarios diferentes y de la misma forma, un usuario puede actuar como actores diferentes.
- **Casos de uso.** Un caso de uso representa el comportamiento que ofrece el sistema de información desde el punto de vista del usuario. Típicamente será un conjunto de transacciones ejecutadas entre el sistema y los actores. Para facilitar la comprensión de los casos de uso del sistema de información en el análisis, es posible agruparlos en paquetes según funcionalidades semejantes o relacionadas.

Además de estos elementos, un diagrama de casos de uso presenta **relaciones**. Las relaciones pueden tener lugar entre actores y casos de uso o entre casos de uso.

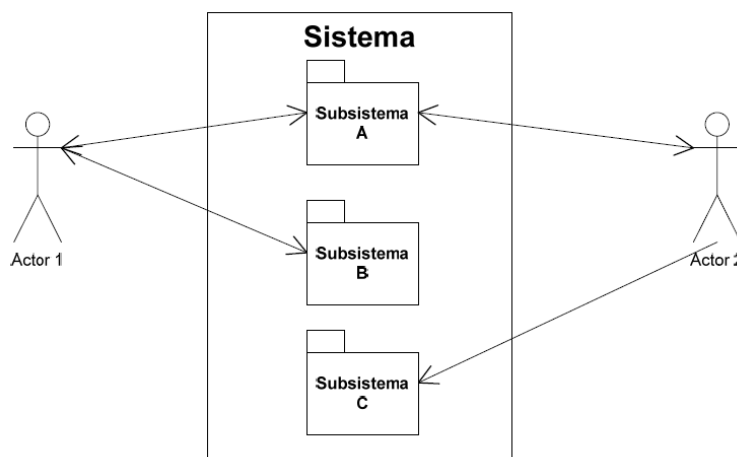
La relación entre un actor y un caso de uso es una relación de comunicación, que indica que un actor interviene en el caso de uso. Normalmente, el actor aporta información para la realización de un caso de uso o recibe información como resultado de la realización del mismo, por ello, esta relación puede ser unidireccional o bidireccional, aunque generalmente se muestra como bidireccional, ya que no es necesario especificar en detalle estas relaciones.

La relación entre casos de uso es una relación unidireccional. Esta relación puede presentar uno de los dos siguientes tipos: "usa" (también denominada "uses") y "extiende" (o "extends").

- La relación **"usa"** se utiliza cuando se quiere reflejar un comportamiento común en varios casos de uso. Es decir, si los casos de uso A y B presentan una parte común, ésta se puede sacar a un tercer caso de uso C. Entonces, habrá una relación "usa" del caso de uso A al C y otra del B al C.
- La relación **"extiende"** se utiliza cuando se quiere reflejar un comportamiento opcional de un caso de uso. Por ejemplo, se tiene el caso de uso A que representa un comportamiento habitual del sistema. Sin embargo, dependiendo de algún factor, este caso de uso puede presentar un comportamiento adicional o ligeramente diferente, que se podría reflejar en un caso de uso B. En este caso, habrá una relación "extiende" del caso de uso B al A.



A la hora de realizar una descripción general, suele ser necesario “empaquetar” los casos de uso en subsistemas para facilitar su comprensión. Cada paquete puede representar parte del sistema o un sistema relacionado. A su vez, cada paquete puede desplegarse en nuevos diagramas de casos de uso.



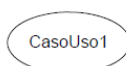
2.2.4 Notación

El diagrama de casos de uso es un grafo de actores, casos de uso y las relaciones entre estos elementos.

Opcionalmente, los casos de uso se pueden enmarcar en un cuadrado que representa los límites del sistema.

Caso de Uso

Un caso de uso se representa mediante una elipse con el nombre del caso de uso dentro o debajo



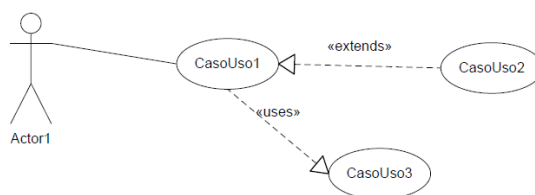
Actor

Un actor se representa con una figura de ‘hombre de palo’ con el nombre del actor debajo de la figura.



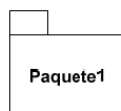
Relación

Dependiendo del tipo de relación, la representación en los diagramas será distinta. Así pues, las relaciones entre un actor y un caso de uso se representan mediante una línea continua entre ellos. Las relaciones entre casos de uso se representan con una flecha discontinua con el nombre del tipo de relación como etiqueta. En las relaciones “extensión” la flecha parte del caso de uso con el comportamiento adicional hacia aquel que recoge el comportamiento básico y en las relaciones “usa” desde el caso de uso básico hacia el que representa el comportamiento común.



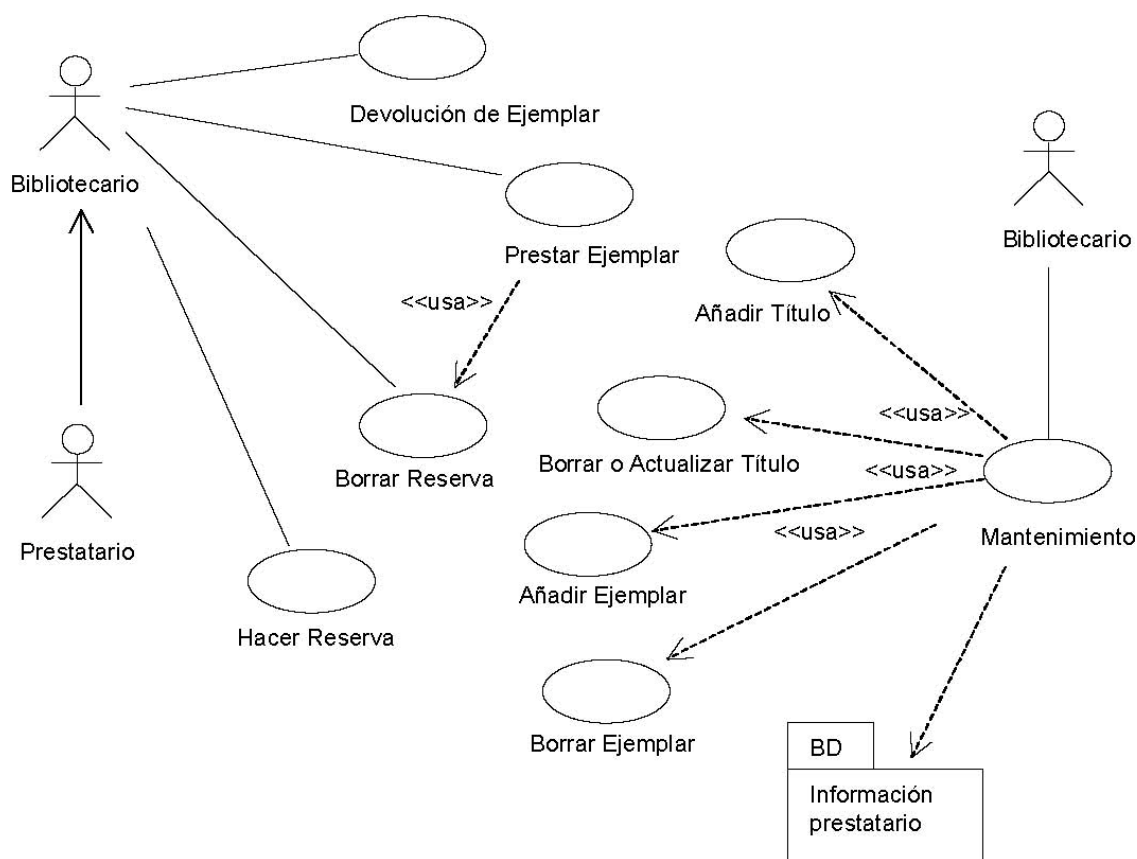
Paquete

Un paquete se representa con un icono con forma de carpeta y con el nombre colocado en su interior.



Ejemplo:

Estudio de una aplicación que se encarga de la gestión de los préstamos y reservas de libros y revistas en una biblioteca:



2.3 Diagrama de paquetes

El objetivo de estos diagramas es obtener una visión más clara del sistema de información orientado a objetos, organizándolo en subsistemas, agrupando los elementos del análisis, diseño o construcción y detallando las relaciones de dependencia entre ellos. El mecanismo de agrupación se denomina Paquete.

Estrictamente hablando, los paquetes y sus dependencias son elementos de los diagramas de casos de uso, de clases y de componentes, por lo que se podría decir que el diagrama de paquetes es una extensión de éstos. En MÉTRICA Versión 3, el diagrama de paquetes es tratado como una técnica aparte, que se aplica en el análisis para la agrupación de casos de uso o de clases de análisis, en el diseño de la arquitectura para la agrupación de clases de diseño y en el diseño detallado para agrupar componentes.

2.3.1 Descripción

Estos diagramas contienen dos tipos de elementos:

- **Paquetes:** Un paquete es una agrupación de elementos, bien sea casos de uso, clases o componentes. Los paquetes pueden contener a su vez otros paquetes anidados que en última instancia contendrán alguno de los elementos anteriores.
- **Dependencias** entre paquetes: Existe una dependencia cuando un elemento de un paquete requiere de otro que pertenece a un paquete distinto. Es importante resaltar que las dependencias no son transitivas.

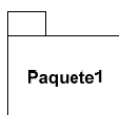
Se pueden optimizar estos diagramas teniendo en cuenta cuestiones como: la generalización de paquetes, el evitar ciclos en la estructura del diagrama, la minimización de las dependencias entre paquetes, etc.

Para completar la descripción de cada paquete, deberá relacionarse el contenido de cada uno (por ejemplo, los casos de uso que están incluidos en cada paquete).

2.3.2 Notación

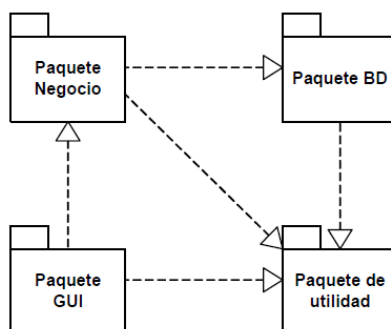
Paquete

Un paquete se representa con un icono con forma de carpeta y con el nombre colocado en su interior.



Dependencia

Las dependencias se representan con una flecha discontinua con inicio en el paquete que depende del otro.



2.4 Diagrama de secuencia

El diagrama de secuencia es un tipo de diagrama de interacción cuyo objetivo es describir el comportamiento dinámico del sistema de información haciendo énfasis en la secuencia de los mensajes intercambiados por los objetos.

2.4.1 Descripción

Un diagrama de secuencia tiene dos dimensiones, el eje vertical representa el tiempo y el eje horizontal los diferentes objetos. El tiempo avanza desde la parte superior del diagrama hacia la inferior. Normalmente, en relación al tiempo sólo es importante la secuencia de los mensajes, sin embargo, en aplicaciones de tiempo real se podría introducir una escala en el eje vertical. Respecto a los objetos, es irrelevante el orden en que se representan, aunque su colocación debería poseer la mayor claridad posible.

Cada objeto tiene asociados una línea de vida y focos de control. La línea de vida indica el intervalo de tiempo durante el que existe ese objeto. Un foco de control o activación muestra el periodo de tiempo en el cual el objeto se encuentra ejecutando alguna operación, ya sea directamente o mediante un procedimiento concurrente.

2.4.2 Notación

Objeto y línea de vida

Un objeto se representa como una línea vertical discontinua, llamada línea de vida, con un rectángulo de encabezado con el nombre del objeto en su interior. También se puede incluir a continuación el nombre de la clase, separando ambos por dos puntos.

Si el objeto es creado en el intervalo de tiempo representado en el diagrama, la línea comienza en el punto que representa ese instante y encima se coloca el objeto. Si el objeto es destruido durante la interacción que muestra el diagrama, la línea de vida termina en ese punto y se señala con un aspa de ancho equivalente al del foco de control.

En el caso de que un objeto existiese al principio de la interacción representada en el diagrama, dicho objeto se situará en la parte superior del diagrama, por encima del primer mensaje. Si un objeto no es eliminado en el tiempo que dura la interacción, su línea de vida se prolonga hasta la parte inferior del diagrama.

La línea de vida de un objeto puede desplegarse en dos o más líneas para mostrar los diferentes flujos de mensajes que puede intercambiar un objeto, dependiendo de alguna condición.

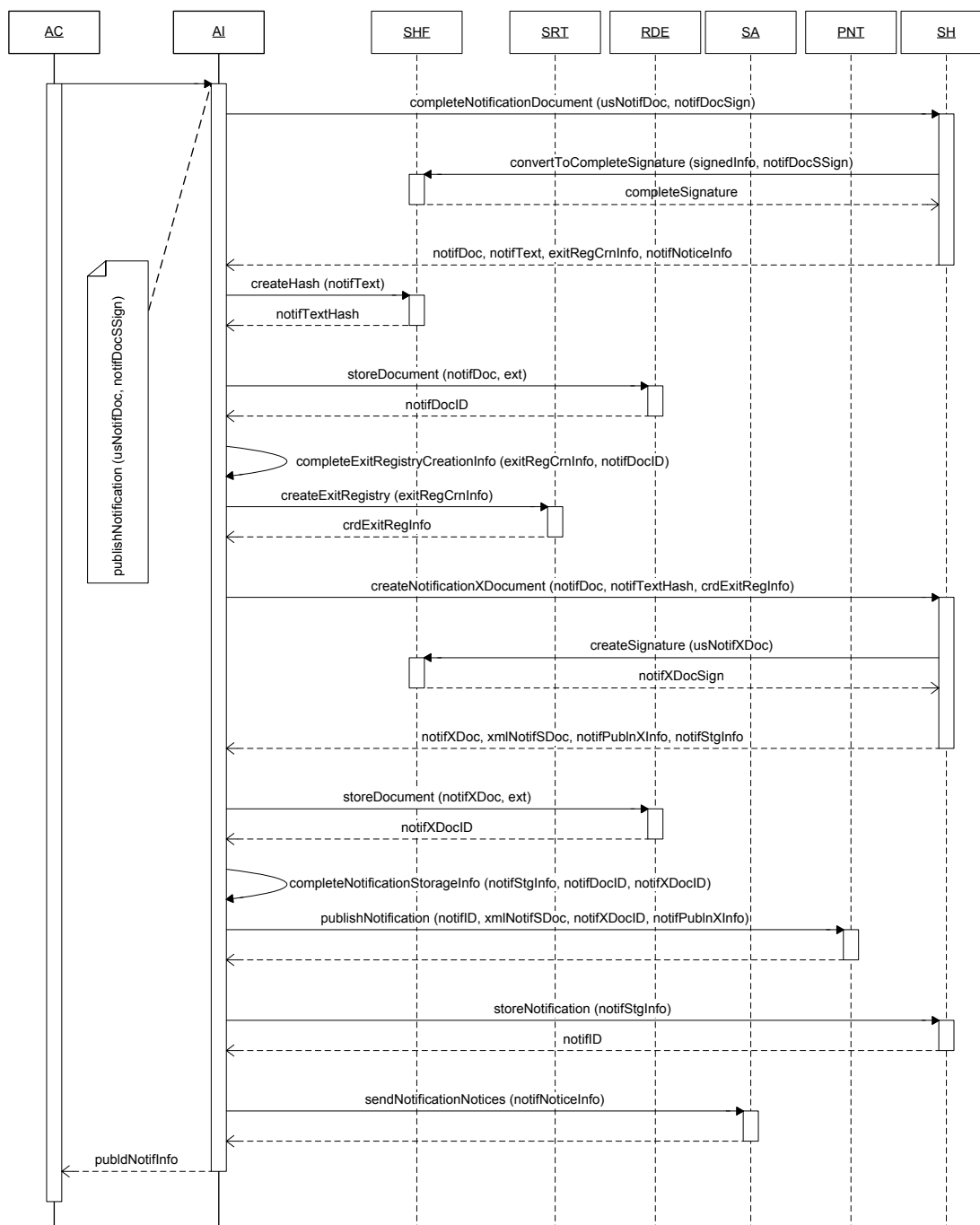
Foco de control o activación

Se representa como un rectángulo delgado superpuesto a la línea de vida del objeto. Su largo dependerá de la duración de la acción. La parte superior del rectángulo indica el inicio de una acción ejecutada por el objeto y la parte inferior su finalización.

Un mensaje se representa como una flecha horizontal entre las líneas de vida de los objetos que intercambian el mensaje. La flecha va desde el objeto que envía el mensaje al que lo recibe. Además, un objeto puede mandarse un mensaje a sí mismo, en este caso la flecha comienza y termina en su línea de vida.

La flecha tiene asociada una etiqueta con el nombre del mensaje y los argumentos. También pueden ser etiquetados los mensajes con un número de secuencia, sin embargo, este número no es necesario porque la localización física de las flechas que representan a los mensajes ya indica el orden de los mismos.

Los mensajes pueden presentar también condiciones e iteraciones. Una condición se representa mediante una expresión booleana encerrada entre corchetes junto a un mensaje, e indica que ese mensaje sólo es enviado en caso de ser cierta la condición. Una iteración se representa con un asterisco y una expresión entre corchetes, que indica el número de veces que se produce.



2.5 Diagrama de clases

El objetivo principal de este modelo es la representación de los aspectos estáticos del sistema, utilizando diversos mecanismos de abstracción (clasificación, generalización, agregación).

2.5.1 Descripción

El diagrama de clases recoge las clases de objetos y sus asociaciones. En este diagrama se representa la estructura y el comportamiento de cada uno de los objetos del sistema y sus relaciones con los demás objetos, pero no muestra información temporal.

Con el fin de facilitar la comprensión del diagrama, se pueden incluir paquetes como elementos del mismo, donde cada uno de ellos agrupa un conjunto de clases.

Este diagrama no refleja los comportamientos temporales de las clases, aunque para mostrarlos se puede utilizar un diagrama de transición de estados, otra de las técnicas propuestas en MÉTRICA Versión 3.

Los elementos básicos del diagrama son:

Clases

Una clase describe un conjunto de objetos con propiedades (atributos) similares y un comportamiento común. Los objetos son instancias de las clases.

No existe un procedimiento inmediato que permita localizar las clases del diagrama de clases. Éstas suelen corresponderse con sustantivos que hacen referencia al ámbito del sistema de información y que se encuentran en los documentos de las especificaciones de requisitos y los casos de uso.

Dentro de la estructura de una clase se definen los atributos y las operaciones o métodos:

- Los **atributos** de una clase representan los datos asociados a los objetos instanciados por esa clase.
- Las **operaciones** o **métodos** representan las funciones o procesos propios de los objetos de una clase, caracterizando a dichos objetos.

El diagrama de clases permite representar clases abstractas. Una Clase abstracta es una clase que no puede existir en la realidad, pero que es útil conceptualmente para el diseño del modelo orientado a objetos. Las clases abstractas no son instanciables directamente sino en sus descendientes. Una clase abstracta suele ser situada en la jerarquía de clases en una posición que le permita ser un depósito de métodos y atributos para ser compartidos o heredados por las subclases de nivel inferior.

Las clases y en general todos los elementos de los diagramas, pueden estar clasificados de acuerdo a varios criterios, como por ejemplo su objetivo dentro de un programa. Esta clasificación adicional se expresa mediante un Estereotipo. Algunos de los autores de métodos OO, establecen una clasificación de todos los objetos que pueden aparecer en un modelo. Los tipos son:

- Objetos Entidad.
- Objetos Límite o interfaz.
- Objetos de control.

Éstos son estereotipos de clases. Un estereotipo representa una la meta-clasificación de un elemento.

Dependiendo de la herramienta utilizada, también se puede añadir información adicional a las clases para mostrar otras propiedades de las mismas, como son las reglas de negocio, responsabilidades, manejo de eventos, excepciones, etc.

Relaciones

Los tipos más importantes de relaciones estáticas entre clases son los siguientes:

- **Asociación.** Las relaciones de asociación representan un conjunto de enlaces entre objetos o instancias de clases. Es el tipo de relación más general, y denota básicamente una dependencia semántica. Por ejemplo, una Persona trabaja para una Empresa.

Cada asociación puede presentar elementos adicionales que doten de mayor detalle al tipo de relación:

- **Rol**, o nombre de la asociación, que describe la semántica de la relación en el sentido indicado. Por ejemplo, la asociación entre Persona y Empresa recibe el nombre de trabaja para, como rol en ese sentido.
- **Multiplicidad**, que describe la cardinalidad de la relación, es decir, especifica cuántas instancias de una clase están asociadas a una instancia de la otra clase. Los tipos de multiplicidad son: Uno a uno, uno a muchos y muchos a muchos.

- **Herencia.** Las jerarquías de generalización/especialización se conocen como herencia.

Herencia es el mecanismo que permite a una clase de objetos incorporar atributos y métodos de otra clase, añadiéndolos a los que ya posee. Con la herencia se refleja una relación “es_un” entre clases. La clase de la cual se hereda se denomina superclase, y la que hereda subclase.

La generalización define una superclase a partir de otras. Por ejemplo, de las clases profesor y estudiante se obtiene la superclase persona. La especialización o especificación es la operación inversa, y en ella una clase se descompone en una o varias subclases. Por ejemplo, de la clase empleado se pueden obtener las subclases secretaria, técnico e ingeniero.

- **Agregación.** La agregación es un tipo de relación jerárquica entre un objeto que representa la totalidad de ese objeto y las partes que lo componen. Permite el agrupamiento físico de estructuras relacionadas

lógicamente. Los objetos “son-parte-de” otro objeto completo. Por ejemplo, motor, ruedas, carrocería son parte de automóvil.

- **Composición.** La composición es una forma de agregación donde la relación de propiedad es más fuerte, e incluso coinciden los tiempos de vida del objeto completo y las partes que lo componen. Por ejemplo, en un sistema de Máquina de café, las relaciones entre la clase máquina y producto, o entre máquina y depósito de monedas, son de composición.
- **Dependencia.** Una relación de dependencia se utiliza entre dos clases o entre una clase y una interfaz, e indica que una clase requiere de otra para proporcionar alguno de sus servicios.

Interfaces

Una interfaz es una especificación de la semántica de un conjunto de operaciones de una clase o paquete que son visibles desde otras clases o paquetes. Normalmente, se corresponde con una parte del comportamiento del elemento que la proporciona.

Paquetes

Los paquetes se usan para dividir el modelo de clases del sistema de información, agrupando clases u otros paquetes según los criterios que sean oportunos. Las dependencias entre ellos se definen a partir de las relaciones establecidas entre los distintos elementos que se agrupan en estos paquetes (ver Diagrama de paquetes).

2.5.2 Notación

Clases

Una clase se representa como una caja, separada en tres zonas por líneas horizontales.

En la zona superior se muestra el nombre de la clase y propiedades generales como el estereotipo. El nombre de la clase aparece centrado y si la clase es abstracta se representa en cursiva. El estereotipo, si se muestra, se sitúa sobre el nombre y entre el símbolo: << >>.

La zona central contiene una lista de atributos, uno en cada línea. La notación utilizada para representarlos incluye, dependiendo del detalle, el nombre del atributo, su tipo y su valor por defecto, con el formato:

visibilidad nombre : tipo = valor-inicial { propiedades }

<< GUI >> Formulario de Reservas	
+	titulo : Titulo
+	prestatario: Informacion_prestatario
+	botonBuscarTitulo_Pulsado ()
+	botonBuscarPrestatario_Pulsado()
+	botonOk_Pulsado ()
+	botonCancelar_Pulsado ()
+	tituloResultado ()
+	prestatarioResultado ()
-	comprobarEstado ()
+	FormularioDeReservas ()
#	botonEliminarTitulo ()

La visibilidad será en general publica (+), privada (-) o protegida (#), aunque puede haber otros tipos de visibilidad dependiendo del lenguaje de programación empleado.

En la zona inferior se incluye una lista con las operaciones que proporciona la clase. Cada operación aparece en una línea con formato:

visibilidad nombre (lista-de-parámetros): tipo-devuelto { propiedad }

La visibilidad será en general publica (+), privada (-) o protegida (#), aunque como con los atributos, puede haber otros tipos de visibilidad dependiendo del lenguaje de programación. La lista de parámetros es una lista con los parámetros recibidos en la operación separados por comas. El formato de un parámetro es:

nombre : tipo = valor-por-defecto

La notación especificada se puede simplificar según el nivel de detalle con el que se quiera trabajar en un momento dado.

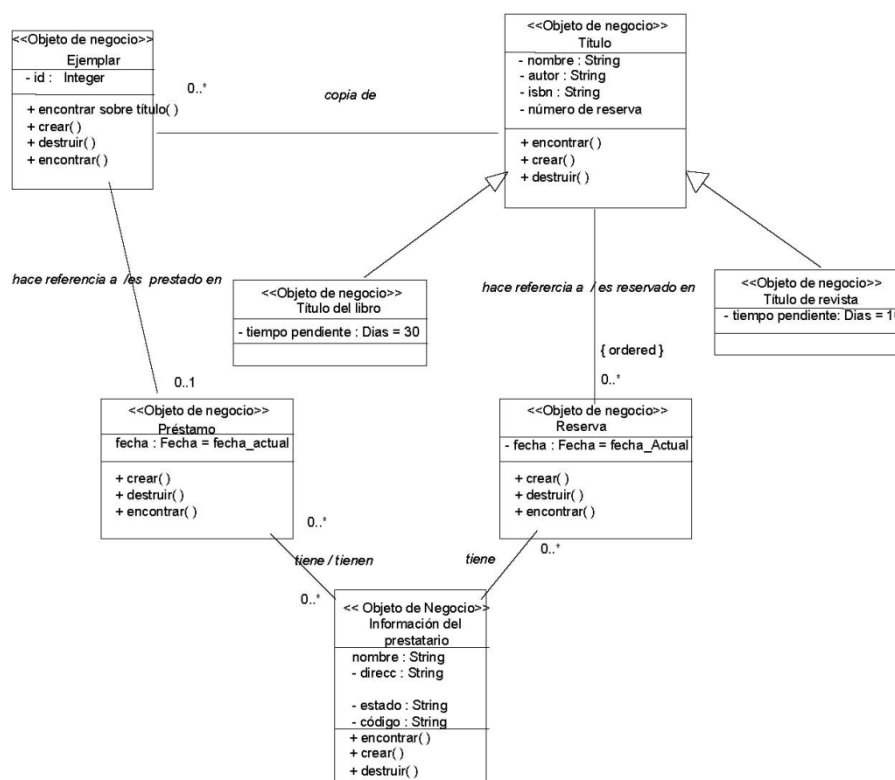
Relaciones

Una relación de asociación se representa como una línea continua entre las clases asociadas. En una relación de asociación, ambos extremos de la línea pueden conectar con la misma clase, indicando que una instancia de una clase, está asociada a otras instancias de la misma clase, lo que se conoce como asociación reflexiva.

La relación puede tener un nombre y un estereotipo, que se colocan junto a la línea. El nombre suele corresponderse con expresiones verbales presentes en las especificaciones, y define la semántica de la asociación. Los estereotipos permiten clasificar las relaciones en familias y se escribirán entre el símbolo: << ... >>.

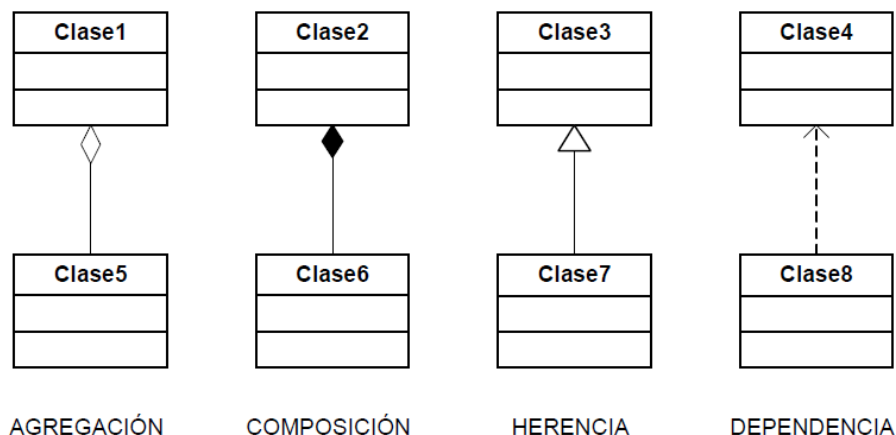
Las diferentes propiedades de la relación se pueden representar con la siguiente notación:

- **Multiplicidad:** La multiplicidad puede ser un número concreto, un rango o una colección de números. La letra 'n' y el símbolo '*' representan cualquier número.
- **Orden:** Se puede especificar si las instancias guardan un orden con la palabra clave '{ordered}'. Si el modelo es suficientemente detallado, se puede incluir una restricción que indique el criterio de ordenación.
- **Navegabilidad:** La navegación desde una clase a la otra se representa poniendo una flecha sin relleno en el extremo de la línea, indicando el sentido de la navegación.
- **Rol o nombre de la asociación:** Este nombre se coloca junto al extremo de la línea que esta unida a una clase, para expresar cómo esa clase hace uso de la otra clase con la que mantiene la asociación.



Además, existen notaciones específicas para los otros tipos de relación, como son:

- **Agregación:** Se representa con un rombo hueco en la clase cuya instancia es una agregación de las instancias de la otra.
- **Composición:** Se representa con un rombo lleno en la clase cuya instancia contiene las instancias de la otra clase.
- **Dependencia:** Una línea discontinua con una flecha apuntando a la clase cliente. La relación puede tener un estereotipo que se coloca junto a la línea, y entre el símbolo: << ... >>.
- **Herencia:** Esta relación se representa como una línea continua con una flecha hueca en el extremo que apunta a la superclase.

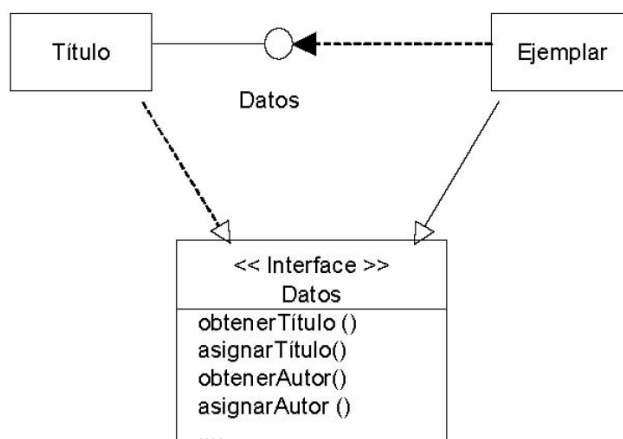


Interfaces

Una interfaz se representa como una caja con compartimentos, igual que las clases. En la zona superior se incluye el nombre y el estereotipo <<Interface>>. La lista de operaciones se coloca en la zona inferior, igual que en las representaciones de clases. La zona en la que se listan los atributos estará vacía o puede omitirse.

Existe una representación más simple para la interfaz: un círculo pequeño asociado a una clase con el nombre de la interfaz debajo. Las operaciones de la interfaz no aparecen en esta representación; si se quiere que aparezcan, debe usarse la primera notación.

Entre una clase que implementa las operaciones que una interfaz ofrece y esa interfaz se establece una relación de realización que, dependiendo de la notación elegida, se representará con una línea continua entre ellas cuando la interfaz se representa como un círculo y con una flecha hueca discontinua apuntando a la interfaz cuando se represente como una clase.



Paquetes

Los paquetes se representan mediante un icono con forma de carpeta y las dependencias con flechas discontinuas entre los paquetes dependientes (ver Diagrama de paquetes).

2.6 Prototipado

El prototipado tiene como objetivo elaborar un modelo o maqueta de las interfaces entre el sistema y el usuario (formatos de pantallas, informes, formularios, etc.), que ayude al usuario a comprender cómo se producirá la interacción con el sistema.

Es importante que el usuario colabore en su desarrollo sugiriendo los cambios que considere oportunos y evalúe hasta que punto las funciones se implementan de forma apropiada y cubren los requisitos identificados.

2.6.1 Descripción

El prototipado constituye un medio a través del cual se simula el aspecto visual del sistema mediante la representación de los conceptos, componentes, objetos gráficos, entradas y salidas requeridas para la ejecución de cada función en respuesta a las necesidades planteadas.

Con objeto de conseguir una interfaz eficiente y compatible con las necesidades de los usuarios evitando así futuras frustraciones, es importante contemplar, previamente, una serie de aspectos que son claves en el diseño de prototipos.

El principal punto a considerar y que constituirá la base sobre la que se centrará el diseño de prototipos es la identificación de los usuarios a los que va dirigido, teniendo en cuenta que debe responder a diferentes individualidades, con distintos conocimientos y habilidades. Cuando los usuarios se sienten a gusto con la imagen del sistema, lo utilizan más eficazmente y cometen menos errores, mejorando su productividad.

Después de estudiar los perfiles de usuarios se deben analizar las funciones que va a soportar el sistema, con el fin de establecer las dependencias existentes entre ellas y su secuencia de ejecución. Además, se debe determinar el tipo de información que requerirá el usuario para llevar a cabo cada función, así como el estilo de interacción más eficaz.

Una vez considerados los dos elementos claves para el diseño de la interfaz, es decir, quiénes son los usuarios y qué funciones tienen asignadas, la forma en que se establezca la comunicación entre el usuario y el sistema será decisiva para conseguir su aceptación. Por tanto, al iniciar el diseño de prototipos de pantallas, es imprescindible tener en cuenta una serie de consideraciones, que potencien la facilidad de uso del sistema, que son reseñados a continuación:

- Utilizar conceptos, términos y símbolos familiares al usuario, de modo que sea fácil de aprender y comprender.
- Mantener la coherencia dentro del propio sistema y entre sistemas: proporcionando abreviaturas estándar, aplicando las mismas reglas de interacción a través de toda la interfaz, y utilizando el mismo formato para los mensajes de error, de aviso o advertencia, mandatos, títulos y comandos con significado similar. De esta manera la experiencia y los conocimientos adquiridos por los usuarios en sistemas previos puede generalizarse a otros sistemas, reduciendo los costes y recursos necesarios para la formación.
- Facilitar la exploración del sistema sin riesgo, permitiendo interrumpir y deshacer las acciones realizadas. De esta forma, el usuario puede utilizar todas las funcionalidades del sistema y trabajar de forma más rápida y eficiente, con la seguridad de que cualquier error puede rectificarse.
- Dificultar la selección de acciones destructivas y no reversibles, pidiéndole verificación de cualquier acción que conlleve un riesgo importante.
- Proporcionar información sobre el estado de ejecución de las funciones, es decir, si la función invocada está en proceso, si se ha completado satisfactoriamente o se ha producido algún error. El usuario nunca debería preguntarse si el sistema está ocupado procesando una transacción o está esperando una nueva entrada y si el sistema no responde a tiempo, iniciar nuevas tareas con el convencimiento de que la anterior finalizó de forma satisfactoria.
- Agrupar las funciones de forma lógica y presentar primero las más utilizadas.
- Buscar la eficiencia en el diálogo evitando cambios frecuentes entre los dispositivos de entrada, tales como el ratón y el teclado.

Estos principios se aplicarán en mayor o menor medida en función de las características del entorno tecnológico y de las necesidades planteadas, no obstante, la adopción de una guía de estilos facilita la coordinación en el equipo de desarrollo y potencia la reutilización.

Una vez considerados estos aspectos, para definir los formatos individuales de las pantallas se realiza un análisis de la información a presentar en cada una de ellas. Este análisis se debe centrar en los siguientes puntos:

- Identificar los diferentes tipos de información como, por ejemplo, campos de datos, títulos, comandos y mensajes de error, con el fin de organizar la pantalla en áreas específicas y conseguir un equilibrio, regularidad, secuencialidad, así como, una simetría en la composición de las mismas.
- Estudiar el espacio disponible en las pantallas para determinar qué datos y en qué situación deben aparecer en las mismas, utilizando un formato de visualización que permita al usuario una rápida asimilación de la información.
- Intentar agrupar los datos relacionados y mostrar sólo aquéllos que son esenciales para la ejecución de la función o para la toma de una decisión, eliminando todas las entradas que sean innecesarias. Nunca se

debe pedir al usuario que introduzca información que pueda adquirirse automáticamente o calcularse internamente.

- Mantener la coherencia entre la entrada y la visualización de datos.
- Proteger al usuario de intentar alguna acción que podría provocar errores, desactivando los comandos que no son operativos en ese contexto.

Finalmente, con objeto de atraer la atención del usuario y mejorar su interacción con el sistema, se deben prestar especial atención a aquellos atributos relacionados con el aspecto estético como son el color y el sonido. Los beneficios de su aplicación son altos si se utilizan de una forma adecuada y consistente.

2.7 Análisis de consistencia

Para realizar el análisis de consistencia será necesario elaborar las siguientes matrices:

- Matriz de requisitos / Casos de uso
- Matriz de mensajes del diagrama de secuencia / operaciones del modelo de clases.
- Matriz de mensajes del diagrama de secuencia / operaciones y atributos del modelo de clases.
- Matriz de objetos del diagrama de secuencia / clases, atributos del modelo de clases.
- Matriz (evento, acción, actividad de clase) / operaciones de clase.
- Correspondencia elementos de negocio de interfaz de usuario / modelo de clases.
- Correspondencia entre elementos de navegación de interfaz de usuario / mensajes del diagrama de interacción de objetos.

Se describen a continuación los análisis de consistencia propuestos:

2.7.1 Matriz de requisitos / Casos de uso

Se comprueba que:

- Cada requisito aceptado del catálogo está soportado en al menos uno de los casos de uso identificados.
- Cada caso de uso implementa al menos un requisito aceptado.

2.7.2 Modelo de clases / Diagramas de secuencia

Se comprueba que:

- Cada mensaje entre objetos se corresponde con una operación de una clase y que todos los mensajes se envían a las clases correctas.
- La clase que recibe un mensaje con petición de datos tiene capacidad para proporcionar esos datos.
- Cada objeto del diagrama de interacción de objetos tiene una correspondencia en el modelo de clases.

2.7.3 Modelo de clases / Interfaz de usuario

Se comprueba que:

- Cada clase que requiera una clase de interfaz de usuario, debe tener asociación con ella en el modelo de clases.
- Todas las clases, atributos y operaciones identificados en la interfaz de usuario, deben tener su correspondencia con algún atributo, operación o clase en el modelo de clases.
- Análisis de la Realización de los Casos de Uso / Interfaz de Usuario
- Cada elemento que active la navegación entre pantallas, debe estar asociado con un mensaje del diagrama de interacción de objetos.

2.7.4 Arquitectura del sistema / Casos de uso

Se revisa que los subsistemas satisfagan la realización de todos los casos de uso, e incluyan las clases identificadas hasta el momento.

2.8 Diagrama de representación

El diagrama de representación tiene como objetivo documentar mediante una imagen una situación específica.

2.8.1 Descripción

Se trata de un diagrama libre, en el que se utiliza cualquier objeto gráfico, con el fin de reflejar algo de interés para el caso y para el que no existe una técnica o práctica.

Se describe, de la forma que se estime oportuno, cada uno de los elementos que componen el diagrama y, en el caso de ser necesario, la notación que se haya empleado, con el fin de facilitar la comprensión del mismo.

2.8.2 Notación

Libre.

2.9 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).

2.9.1 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.

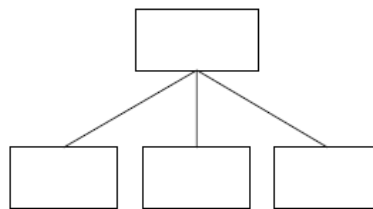
Otros componentes que se pueden representar en el diagrama de estructura son:

- **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.

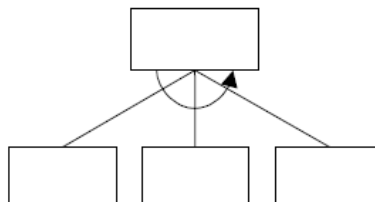
2.9.2 Estructuras del diagrama

Existen ciertas representaciones gráficas que permiten mostrar la secuencia de las llamadas entre módulos. Las posibles estructuras son:

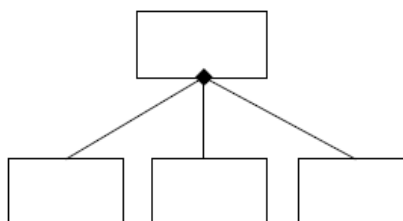
- **Secuencial:** un módulo llama a otros módulos una sola vez y, se ejecutan de izquierda a derecha y de arriba abajo.



- **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 de los de nivel inferior.



2.9.3 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.

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 módulo 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.

2.9.4 Notación

Módulo

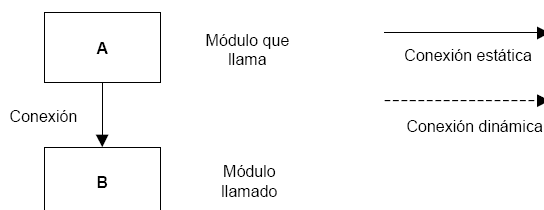
Se representa mediante un rectángulo con su nombre en el interior.



Un módulo predefinido se representa añadiendo dos líneas verticales y paralelas en el interior del rectángulo.

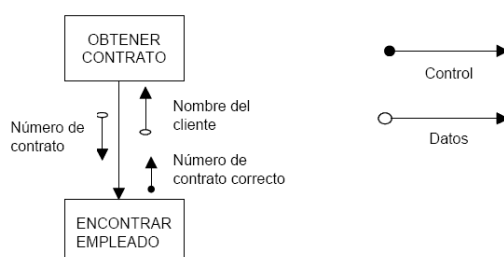
Conexión

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.



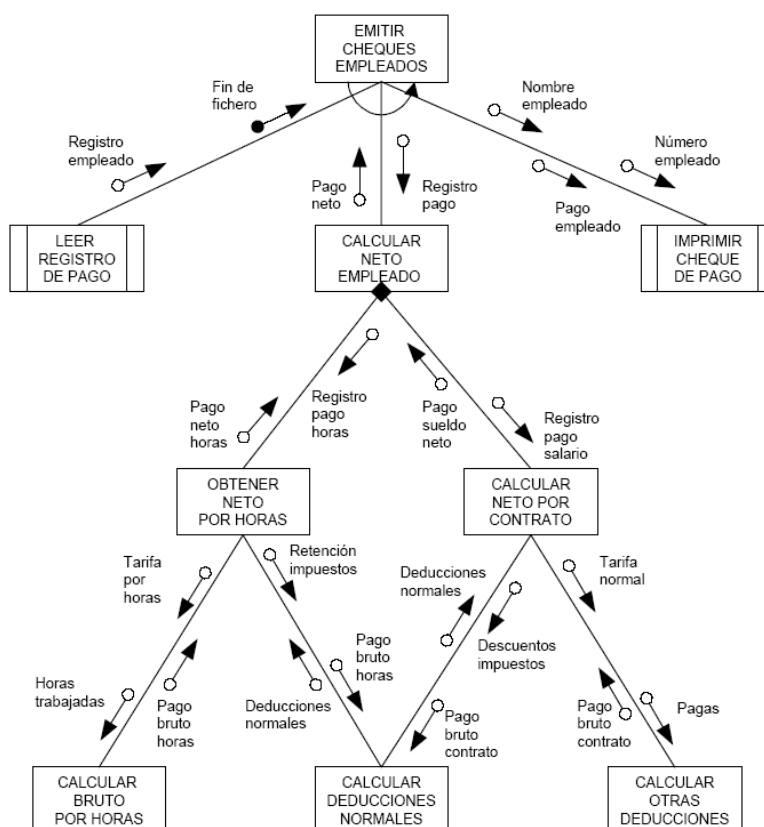
Parámetros

La representación varía según su tipo: control (flags) o datos.



Ejemplo.

El siguiente ejemplo muestra un proceso de emisión de cheques para el pago de nóminas de los empleados de una empresa. 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.



2.10 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.

2.10.1 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

ARINbide

Conceptos Básicos y Técnicas

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 tiene 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, y 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, o si 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 propagaciones, 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).

2.11 Caminos de acceso

El objetivo de esta práctica es analizar la secuencia de acceso a los datos que realizan los módulos a través del modelo de datos. También puede utilizarse para entornos de ficheros.

Permite verificar en el proceso Análisis del Sistema de Información (ASI) que el modelo lógico de datos normalizado satisface las principales consultas de información recogidas en el catálogo de requisitos, y en el proceso Diseño del Sistema de Información (DSI) que el modelo físico de datos soporta adecuadamente los principales accesos de actualización, cuando proceda, y de consulta.

2.11.1 Descripción

Los caminos de acceso representan la secuencia y tipo de acceso a los datos persistentes del sistema, que deben realizar los procesos primitivos a partir del modelo lógico de datos normalizado, o los módulos/clases a partir del modelo físico de datos.

En función del modelo de datos sobre el que se realiza el acceso, se identifican las entidades o tablas/ficheros que deben ser accedidas por cada proceso primitivo o módulo/clase, y se crean vistas del modelo de datos en el que aparecen únicamente dichas entidades (subconjunto del modelo de datos). Es conveniente examinar todas las entidades relacionadas con las identificadas inicialmente, debido a que puede que aparezcan más entidades que no se habían contemplado en un primer momento.

Para cada entidad identificada se indica el tipo de acceso realizado, es decir, si se trata de una lectura, inserción, modificación o eliminación.

Una vez identificadas las entidades o tablas/ficheros y el tipo de acceso, el siguiente paso es determinar el orden que se sigue para la obtención de los datos a través del modelo de datos, con el fin de identificar accesos redundantes o excesivamente complejos que puedan comprometer el rendimiento final del sistema.

Se recomienda aplicar esta práctica para aquellos módulos que presenten, entre otras, alguna de las siguientes características:

- Tratamiento crítico.
- Accesos complejos a datos.
- Alta concurrencia.

2.12 Diagrama de componentes

El diagrama de componentes proporciona una visión física de la construcción del sistema de información. Muestra la organización de los componentes software, sus interfaces y las dependencias entre ellos.

2.12.1 Descripción

Como ya se ha indicado, los elementos de estos diagramas son los componentes software y las dependencias entre ellos. Un componente es un módulo de software que puede ser código fuente, código binario, un ejecutable, o una librería con una interfaz definida. Una interfaz establece las operaciones externas de un componente, las cuales determinan una parte del comportamiento del mismo.

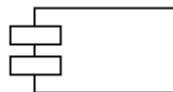
Además se representan las dependencias entre componentes o entre un componente y la interfaz de otro, es decir uno de ellos usa los servicios o facilidades del otro.

Estos diagramas pueden incluir paquetes que permiten organizar la construcción del sistema de información en subsistemas y que recogen aspectos prácticos relacionados con la secuencia de compilación entre componentes, la agrupación de elementos en librerías, etc.

2.12.2 Notación

Componente

Un componente se representa como un rectángulo, con dos pequeños rectángulos superpuestos perpendicularmente en el lado izquierdo.



Para distinguir distintos tipos de componentes se les puede asignar un estereotipo, cuyo nombre estará dentro del símbolo: << ... >>

Interfaz

Se representa como un pequeño círculo situado junto al componente que lo implementa y unido a él por una línea continua. La interfaz puede tener un nombre que se escribe junto al círculo. Un componente puede proporcionar más de una interfaz.



Paquete

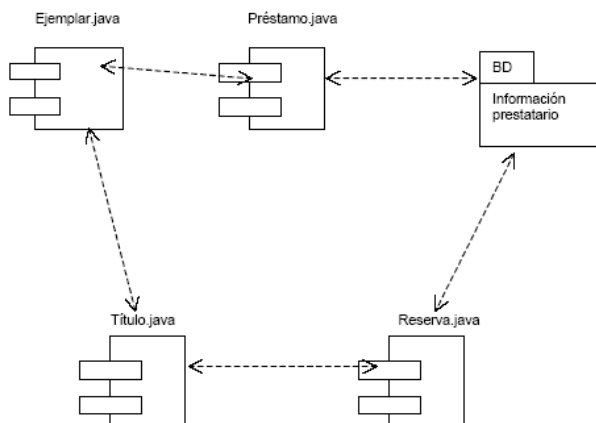
Un paquete se representa con un icono de carpeta (ver Diagrama de Paquetes).

Relación de dependencia

Una relación de dependencia se representa mediante una línea discontinua con una flecha que apunta al componente o interfaz que provee del servicio o facilidad al otro. La relación puede tener un estereotipo que se coloca junto a la línea, entre el símbolo: <<...>>.

Ejemplo.

Sistema encargado de la gestión de los préstamos y reservas de libros y revistas en una biblioteca. El lenguaje de desarrollo será Java, y los accesos a la información del prestatario serán mediante un paquete de Base de Datos.



2.13 Diagrama de despliegue

El objetivo de estos diagramas es mostrar la disposición de las particiones físicas del sistema de información y la asignación de los componentes software a estas particiones. Es decir, las relaciones físicas entre los componentes software y hardware en el sistema a entregar.

2.13.1 Descripción

En estos diagramas se representan dos tipos de elementos, nodos y conexiones, así como la distribución de componentes del sistema de información con respecto a la partición física del sistema.

En MÉTRICA Versión 3 se propone una definición concreta de nodo, prescindiendo de determinados detalles, pero permitiendo una continuidad tanto en el diseño como en la construcción del sistema de información. Con este fin, se utiliza el nodo como partición física ofuncional real, pero sin descender a detalles de infraestructura o

dimensionamiento; por ejemplo, interesa si el nodo procesador es arquitectura Intel, pero no tanto si tiene dos o cuatro procesadores.

Las conexiones representan las formas de comunicación entre nodos. Además, a cada nodo se le asocia un subsistema de construcción que agrupa componentes software, permitiendo de este modo, determinar la distribución de estos componentes. Por lo tanto, un diagrama de despliegue puede incluir, dependiendo del nivel de detalle, todos los elementos descritos en la técnica de diagrama de componentes, además los nodos y las conexiones propios de esta técnica.

2.13.2 Notación

Nodo

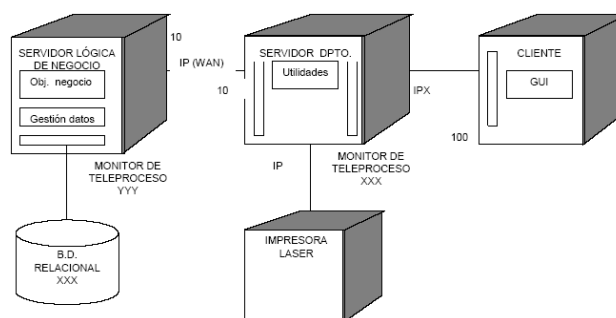
Se representa con la figura de un cubo. El nodo se etiqueta con un nombre representativo de la partición física que simboliza. Se pueden asociar a los nodos subsistemas de construcción.

Conexión

Las conexiones se representan con una línea continua que une ambos nodos y pueden tener una etiqueta que indique el tipo de conexión. (ejemplo: canal, red, protocolo, etc.)

Ejemplo:

El diagrama representa una arquitectura compuesta por un servidor central de lógica de negocio y acceso a datos, en un monitor de teleproceso de tipo XXX, al cual hay conectados 10 servidores departamentales, con clientes (100) e impresora conectados a cada uno de ellos. No interesa tanto recoger en el diagrama la infraestructura real (la exactitud de la configuración, número de procesadores que pueden cambiar con el tiempo y en principio no afecta ni al diseño ni a la construcción), como el tipo “genérico” de los servidores, los volúmenes en el caso de que sean significativos (por ejemplo: 100 puestos por departamento).



2.14 Identificación de la configuración

Los objetivos de la identificación de la configuración son:

- Definir la configuración (conjunto de elementos que conforman el sistema o producto final del proyecto, que serán objeto de control).
- Codificar los elementos de configuración, de forma que permita su identificación unívoca.
- Identificar las líneas base y sus componentes.
- Identificar las relaciones y dependencias entre los elementos de configuración.
- Definir, asignar y actualizar los atributos de los elementos de configuración.
- Establecer las medidas de seguridad para el acceso y la modificación de los productos o elementos de configuración.

La identificación de la configuración se deberá realizar sobre todos los elementos del sistema, estando su configuración constituida básicamente por documentos y unidades de código, según se indica posteriormente. Dado que estos tipos de elementos son de diferente naturaleza se deberá disponer de procedimientos independientes para cada uno de ellos.

En particular, para el control de configuración de unidades de código deberán tenerse en cuenta los siguientes requisitos de carácter general:

- **Estabilidad:** El entorno de trabajo de programación y pruebas requiere un mínimo de estabilidad. Este requisito se satisface mediante una coordinación centralizada de las modificaciones a implantar en el código compartido con una visión global del estado de los trabajos en curso, así como de los objetivos del proyecto.
- **Identificación:** Cada unidad de código, ya se trate de fuentes, objetos o ejecutables, debe ser identificada de forma diferenciada. Para ello, debe marcarse cada una con un código de identificación, que la distinga de las demás y que permita reconocer su estado de evolución.
- **Trazabilidad:** Ha de asegurarse la trazabilidad de los requisitos aplicables al producto software. Ello supone que cada unidad de código ha de ser relacionada con la documentación técnica que contenga dichos requisitos.
- **Reproducibilidad:** En cualquier momento ha de ser posible reproducir cualquier unidad de código a partir de sus fuentes o constituyentes. En consecuencia ha de mantenerse la información relativa a las distintas transformaciones que se efectúan para su obtención.
- **Recuperabilidad:** A pesar del necesario control de accesos al código compartido, cada unidad vigente debe ser fácilmente recuperable. Se ha de prever la posibilidad de recuperar incluso las unidades de código que hayan quedado obsoletas.

2.14.1 Configuración

La configuración es el conjunto de elementos, sus características y requisitos, que definen un sistema de información.

Los elementos, que configuran el sistema o una parte de él, (elementos de configuración) pueden ser tanto productos intermedios como finales.

- Documentos generados en el desarrollo.
- Documentos de referencia del proyecto.
- Documentos de referencia de productos comerciales.
- Diccionarios de herramientas de ayuda al desarrollo.
- Componentes software generados en el desarrollo o comerciales (unidades de código).
- Componentes software de desarrollo y soporte (compiladores, herramientas de desarrollo, sistemas operativos, bancos de pruebas).
- Componentes hardware comerciales.
- Productos subcontratados.

Estos elementos se agrupan por fases del ciclo de vida del proyecto, obteniéndose así los productos fin de fase. Estos productos una vez que han sido verificados, validados y aprobados son congelados con el fin de poder controlar sus modificaciones, sirviendo de referencia para el desarrollo en las fases posteriores.

Se indican a continuación unos criterios básicos que pueden facilitar las tareas de control de configuración del código, si se tienen en cuenta en el momento de definir la arquitectura de bajo nivel del producto software. La importancia relativa de cada criterio expuesto depende de la naturaleza del producto final, de la metodología y de los entornos de desarrollo y explotación:

- **Tamaño:** Una descomposición en elementos de reducido tamaño puede conducir a efectos de excesiva granularidad, con aumentos de esfuerzos y costes, tanto en pruebas como en control. Lo contrario, grandes unidades de código, conduce a la disminución del control de la configuración del mismo. (En la bibliografía se recomiendan tamaños del orden de 100 líneas de código fuente en lenguaje de alto nivel, con un máximo de 200 líneas de código no expandible).
- **Modularidad:** Definir unidades de código con interfaces simples y con una única entrada y salida del flujo de control, obteniéndose acoplamientos débiles entre los distintos elementos.
- **Cohesión:** Definir unidades de código con unicidad y coherencia de sus funcionalidades, procesos y datos manejados por el elemento.
- **Ocultación de la información:** Posibilidad de "ocultar" dentro del elemento la forma en que se realizan los procesos o tareas y las estructuras de datos empleadas, dejando visible únicamente sus resultados.

- Estandarización de interfaces internas: Dar tratamiento a las interfaces internas como unidades de código individualizadas. Puede ser el caso de definición de estructuras de datos que se mantienen en diccionarios.
- Reutilización (exportación de unidades de código): Separación de aquellos elementos que por su generalidad pueden ser aprovechados en otros sistemas.
- Importación de elementos: Tratamiento como unidades de código individualizadas de aquellos elementos que proceden de otros sistemas.
- Expectativa de cambios: Selección de las unidades de código tomando en consideración la probabilidad de la ocurrencia de cambios en fases posteriores del ciclo de vida.
- Portabilidad: Separación en diferentes unidades de código de aquellas características que dependan del entorno de explotación.
- Criticidad: Separación en diferentes unidades de código de las funcionalidades cuyo fallo pueda resultar crítico, por implicar daños materiales o pérdidas económicas.
- Complejidad de diseño y pruebas: Separación en diferentes unidades de código de las características que presentan una complejidad elevada en cuanto al diseño de algoritmos o su facilidad de prueba.
- Organización para el desarrollo y el mantenimiento: Descomposición del producto, teniendo en cuenta la organización de los equipos humanos tanto de desarrollo como de mantenimiento.
- Integrabilidad: Realizar la descomposición teniendo en cuenta la facilidad de integración, evitando bucles entre las diferentes unidades de código a través de sus interfaces.
- Localización geográfica de la explotación: En el caso de los sistemas distribuidos, tomar en consideración la localización de la explotación en la descomposición de los elementos de configuración.

El responsable de gestión de configuración deberá comprobar que la descomposición del producto es la identificada en la configuración y se encuentra reflejada en la documentación de diseño aplicable. En caso contrario, se efectuará dicha descomposición junto con los responsables correspondientes de las distintas áreas del proyecto.

Antes de que comiencen las tareas de codificación, conviene que la descomposición efectuada sea confirmada por el responsable del proyecto y, si procede, incluso por el cliente.

2.14.2 Documentos

Codificación de documentos

Todos los documentos generados o utilizados en el proyecto deberán ser identificados mediante un código, que deberá tener las siguientes características:

- Unicidad. El código deberá ser único para cada documento, no pudiendo darse la posibilidad de que existan dos documentos distintos con la misma identificación. Se deberá incluir en el código del documento su versión, considerando los conceptos de revisión (documento que anula a la anterior versión) y variante (nueva versión de un documento permaneciendo vigente los anteriores).
- Simplicidad. El código debe constar sólo de los caracteres necesarios y dispuestos en una secuencia que facilite su encuadre metodológico y técnico, así como su ubicación dentro del ciclo de vida.
- Autoexplicativo. El código debe proporcionar la mayor información posible sobre el documento.
- Relacionable. El código debe proporcionar información sobre la relación entre los documentos.

En cualquier caso, los criterios de codificación deberán ser explicitados con el detalle suficiente en el plan de gestión de configuración.

Atributos de documentos

El responsable de gestión de configuración deberá definir y actualizar los atributos de cada documento bajo control de configuración.

Los atributos pueden incluir la identificación y características físicas, funcionales y de calidad de los documentos, como son:

- Proyecto que genera la existencia del documento.

- Fase/subfase en que se produce, para documentos componentes de una línea base.
- Código de identificación, incluyendo su versión, según se ha indicado anteriormente.
- Título del documento.
- Estado. .
- Fecha de edición.
- Autor. Nombre del responsable del documento.
- Autorizado. Nombre del que autoriza el documento.
- Tarea o paquete de trabajo que contempla su elaboración.
- Lista de distribución. Lista de las personas, unidades operativas u organizaciones externas que deberán recibir una o varias copias, una vez que ha sido aprobado.
- Documentos de referencia. Lista de los documentos que se han utilizado para su elaboración.
- Otras referencias. Este atributo contempla la posibilidad de que el documento posea una o varias identificaciones diferentes a la asignada por control de configuración.
- Documento sujeto a evaluación y verificación por el grupo de Calidad. Indica si el documento ha de ser revisado, autorizado y, en su caso, aprobado por Calidad.
- Confidencialidad. Indica si el documento debe tener un acceso restringido y el nivel de autorización necesario.
- Soporte. Tipo de soporte donde se ha editado el documento.
- Lenguaje. Idioma en que está escrito el documento.
- Localización.

2.14.3 Unidades de código

Codificación de unidades de código

Cada elemento de configuración del producto software deberá tener asignado un código de identificación único.

Esta identificación deberá tener en cuenta el sistema de gestión de ficheros del entorno de desarrollo al que pertenecen las bibliotecas, de tal forma que sea el mismo para gestión de configuración que para gestión de la biblioteca de desarrollo.

Para mantener la trazabilidad de los requisitos aplicables a las unidades de código, contenidos en la correspondiente documentación técnica, es conveniente que la información proporcionada por la denominación de cada unidad programable, permita determinar el componente de alto nivel del que se deriva. Por tanto, parte de las reglas para la definición de estos códigos de identificación han debido ser definidas antes del comienzo de la fase de diseño, siempre que se conozca y tenga en cuenta el sistema de gestión de ficheros específico del entorno de desarrollo.

La identificación de cada unidad de código debe incluir el concepto de versión.

A continuación, se indican algunos criterios en relación con el sistema de codificación y su utilización:

- Es beneficioso, tal como se ha indicado anteriormente, que el código empleado sea significativo, pudiéndose deducir del mismo la posición que ocupa la unidad en la estructura jerarquizada del sistema. Para ello, se pueden reservar los primeros caracteres del formato. Cuando no exista posibilidad de error se puede omitir el campo correspondiente al proyecto o sistema.
- Reflejar, dentro del formato de identificación, el estado de evolución de la unidad. Para ello se debe reservar un campo del formato del código de identificación, donde se pueda indicar la correspondiente versión.
- Recoger en el código otros conceptos como puede ser el tipo de soporte de almacenamiento (cinta, disco, papel).
- Los elementos de unión de programas, tales como procedimientos catalogados o sentencias de control, deben también ser identificados como el resto de las unidades.
- Las identificaciones asignadas a las unidades de código objeto y ejecutables deben diferenciarse de las correspondientes al código fuente del que proceden.
- Los códigos de identificación de las unidades han de formar parte de las mismas, quedando transcritos internamente, siempre que sea posible.

Atributos de las unidades de código

El responsable de gestión de configuración deberá definir los atributos correspondientes a cada unidad de código, poniéndola a disposición del personal del proyecto.

Para ello debe recopilar previamente la siguiente información:

- Diagrama jerárquico, resultante de la descomposición del sistema.
- Código de identificación asignado a cada unidad de código.
- Subsistema o componente de nivel superior al que pertenece la unidad.
- Relación de módulos que puedan componer cada unidad y un breve resumen de las funcionalidades cubiertas.
- Referencia a la documentación técnica que contenga los requisitos aplicables a la unidad, incluyendo la denominación de la misma en dicho documento (si la citada denominación fuese distinta del código de identificación asignado).
- Descripción de las interfaces previstas para la unidad, ya sea con otras unidades o con otros sistemas.
- Otros datos de interés, tales como el responsable de su codificación, fecha prevista de entrega a la biblioteca o requisitos especiales de prueba.

Los atributos que caracterizan las unidades de código incluyen la identificación y características físicas, funcionales y de calidad de las unidades de código. Estos atributos pueden ser los siguientes:

- Proyecto.
- Fase/módulo en que se produce, para elementos componentes de una línea base.
- Código de identificación, incluyendo su versión, según se ha indicado anteriormente.
- Título de la unidad.
- Estado.
- Fecha de creación, fecha de última modificación y, en su caso, fecha de baja.
- Autor. Nombre del responsable de la unidad.
- Autorizado. Nombre del que autoriza la unidad.
- Tarea o paquete de trabajo que contempla su elaboración.
- Subsistema o componente de nivel superior al que pertenece la unidad.
- Relación de módulos que puedan componer cada unidad y un breve resumen de las funcionalidades cubiertas.
- Referencia a la documentación técnica y requisitos aplicables a la unidad.
- Interfaces previstas para la unidad, ya sea con otras unidades o con otros sistemas.
- Localización. Espacio de almacenamiento y nombre externo del fichero que contiene cada unidad.
- Procedimiento de generación, si procede. La información necesaria para reproducir las unidades generadas como consecuencia de transformaciones a partir de otras (parámetros de transformación).
- Unidades de código que deben ser aprobadas conjuntamente.
- Otras referencias. Este atributo contempla la posibilidad de que la unidad posea una o varias identificaciones diferentes a la asignada por control de configuración.
- Unidad sujeta a evaluación y verificación. Indica si la unidad ha de ser inspeccionada, autorizada y, en su caso, aprobada.
- Confidencialidad. Indica si la unidad de código debe tener un acceso restringido, su tipo y el nivel de autorización necesario.
- Soporte. Tipo de soporte donde se ha editado.
- Entorno software de base. Generalmente se indicará exclusivamente el lenguaje de programación y su versión, aunque en algún caso podría ser necesario ampliar esta especificación.

Esta información deberá ser actualizada periódicamente por el responsable de gestión de configuración o persona en quien delegue (administrador de la biblioteca) a lo largo del ciclo de vida del sistema, permitiendo mantener la identificabilidad y trazabilidad del código y facilitando el estudio y propagación de los cambios que se vayan requiriendo.

2.14.4 Líneas base

Se denomina línea base a los documentos y/o unidades de código que se obtienen como resultado de cada una de las fases del ciclo de vida y que sólo alcanzan dicha consideración, después de haberse superado un proceso de evaluación y verificación, habiendo sido aprobados y aceptados para su uso posterior. Se incluirán también como parte de la línea base, el software de desarrollo y apoyo utilizado en el proyecto, y el software subcontratado a terceros.

Una de las características esenciales de una línea base es que, una vez que han sido establecidas, pasan automáticamente a estar bajo control de configuración y cualquier cambio futuro, que pudiera surgir, deberá implantarse siguiendo un procedimiento formal de cambios.

- En definitiva, lo que se pretende con las fases del ciclo de vida y las líneas base correspondientes, es establecer una serie de puntos intermedios o hitos en el transcurso del proyecto, para:
- Consolidar los resultados, estableciendo una base coherente que sirva de punto de partida para abordar el desarrollo ulterior en un entorno estable.
- Evaluar el progreso obtenido en el desarrollo.
- Adelantar la "visibilidad" sobre el producto software a fases más tempranas del desarrollo y asegurar el cumplimiento de las necesidades de usuario.

Otras denominaciones utilizadas en la bibliografía para significar el concepto de línea base son: documento base, línea de referencia, referencial, líneas básicas y el término anglosajón "baseline".

2.15 Control de cambios

Es necesario prever la modificación de algún aspecto de las líneas base ya consolidadas, no pudiéndose mantener su congelación con carácter definitivo.

Los cambios, que a priori pueden plantearse como localizados, se propagan a través de las interfaces internas incluso con efectos multiplicadores. Se requiere, por tanto, analizar su impacto real en la globalidad de los trabajos ya realizados o en curso.

Por otra parte, puesto que la línea base externa representa el acuerdo alcanzado sobre la evolución del producto software en puntos discretos de su ciclo de vida, los cambios no deben incorporarse de forma unilateral. Igualmente ha de tenerse en cuenta la opinión de otros grupos que puedan quedar afectados.

Adicionalmente, es necesario coordinar y controlar las tareas de ejecución de los cambios autorizados.

La solución a toda esta problemática es establecer un procedimiento formal para la autorización e implantación de los cambios a las líneas base.

2.16 Gestión de bibliotecas

En el desarrollo de un proyecto se generan diferentes productos susceptibles de ser incorporados a una biblioteca. Estos productos pueden ser documentos de gestión y técnicos o productos software, tanto en soporte papel, como en soporte magnético.

Cada uno de estos tipos de soporte precisa unas especificaciones diferentes sobre condiciones ambientales, métodos de almacenamiento, etc., por lo que se deberán crear las siguientes bibliotecas de proyecto:

2.16.1 Biblioteca de documentos

Esta biblioteca contiene todos los documentos escritos y productos software en soporte papel.

La biblioteca de documentos debe cumplir las especificaciones técnicas normales de un archivo de documentación escrita, pudiéndose seguir la normativa y los procedimientos establecidos en la empresa para la biblioteca de producción.

2.16.2 Biblioteca de desarrollo

Esta biblioteca contiene, en términos generales, todos los documentos y productos software en soporte magnético.

Respecto a la biblioteca de desarrollo se realizan inicialmente las siguientes consideraciones de carácter general:

- Se considerarán como elementos de la biblioteca de desarrollo los productos software (unidades de código) y, como ya se ha indicado anteriormente, la documentación, ya sea técnica o de gestión, cuando ésta se haya generado mediante medios informatizados y se encuentre en soporte magnético.
- Se almacenarán como producto software los procedimientos catalogados, sentencias de control y tanto las unidades de código fuente como los objetos y ejecutables. En el caso de que se incluya como requisito del proyecto la realización de pruebas y su documentación, también se deberán almacenar los ficheros de datos correspondientes a los casos de prueba.
- Deberá disponer de un almacenamiento centralizado del código independiente de los espacios de trabajo reservados al equipo de programación. Este almacenamiento centralizado proporciona un entorno que permite la aplicación de las medidas necesarias para el control de configuración del código durante el transcurso de un proyecto de desarrollo.

Para el desempeño de las tareas administrativas de la gestión de la biblioteca de desarrollo en su área de almacenamiento centralizado, se crea la figura de un administrador de la biblioteca, que podrá ser el responsable de gestión de configuración o persona delegada; en cualquier caso la responsabilidad de esta área corresponde al responsable de gestión de configuración.

Los beneficios que se obtienen de la implantación de las bibliotecas es mayor en proyectos grandes y complejos, pero es conveniente seguir las directrices de este documento aún en el caso de los pequeños.

2.16.3 Control de accesos

El administrador debe mantener las unidades almacenadas en medios controlados, de forma que:

- Se eviten los accesos no autorizados.
- Se minimice el riesgo de pérdida de datos.
- Sea posible su localización y recuperación.

2.16.4 Copias de seguridad

El administrador de la biblioteca debe efectuar periódicamente copias de seguridad, para posibilitar la recuperación de la información en caso de pérdida accidental de datos.