Capitulo 1 Marco Teórico

1.1 Reingeniería del software

La reingeniería del software nace como una necesidad de replantear o actualizar sistemas que por varias razones han perdido funcionalidad, o que simplemente debido a las tareas propias de la vida diaria de un sistema, han sido modificadas y cambiadas con el ánimo de satisfacer las nuevas necesidades de los procesos de una empresa, sin considerar las buenas prácticas de ingeniería del software, a tal punto que se han vuelto frágiles e inestables.

El software imposible de mantener no es un problema nuevo, de hecho, el interés por la reingeniería del software ha sido desarrollado por un "iceberg" de mantenimiento del software que lleva creciendo desde hace más de treinta años.

1.1.1 Mantenimiento del software

La analogía del mantenimiento del software con un "iceberg" tiene su lógica debido a esperamos que lo que se ve sobre la superficie sea lo único, pero sabemos que existe una enorme masa de potenciales problemas y costos debajo.

El mantenimiento del software puede considerarse alrededor del 60% de las inversiones efectuadas por una empresa de desarrollo, el mismo que crece a medida que se desarrolla software.

Una explicación de por qué se invierte tanto tiempo en mantenimiento nos la da Osborne y Chikofsky [1] así:

Gran parte del software del que dependemos en la actualidad tiene por término medio entre 10 y 15 años de antigüedad. Aún cuando estos programas se crearon empleando las mejores técnicas de diseño y codificación conocidas en su época, se crearon cuando los tamaños de programas y almacenamiento eran las preocupaciones principales. A continuación se trasladaron a las nuevas plataformas, se ajustaron para adecuarlos a las nuevas máquinas y sistemas operativos, se mejoraron para satisfacer nuevas necesidades de usuarios. Todo esto sin tener en cuenta la arquitectura global.

Los cambios son inevitables en un sistema computarizado, por lo tanto debemos desarrollar mecanismos para evaluar sus modificaciones.

Tipo de mantenimiento: correctivo, adaptativo, mejoras o perfeccionamiento y **preventivo o reingeniería.**

[1] Ingeniería del Software, Pressman

1.1.2 Modelo de procesos de reingeniería del software

La reingeniería es una tarea de reconstrucción, se la puede entender mejor haciendo analogía a la reconstrucción de una casa. Considerar lo siguiente:
- Se adquiere una casa en la que sabemos de antemano que puede ser preciso reconstruirla.
- Sería bueno hacer una inspección profesional, para determinar si efectivamente necesita la reconstrucción, se crearía una lista de criterios para que la inspección fuera sistemática.
- Revisar la estructura, pues si esta en buen estado es posible conservarla, y simplemente hacer una remodelación.
- Entender la forma en que fue construida originalmente.
- Si se decide a reconstruir o remodelar, utilice materiales modernos y de buena calidad.
- Use herramientas que garanticen calidad.

Para implementar estos principios en un proceso de reingeniería de software, se han definido seis actividades, las mismas que en algunas ocasiones pueden presentarse en un modelo secuencial lineal, pero no siempre es así.

El paradigma mostrado en el gráfico 1 es un modelo cíclico, lo que significa que cualquiera de las etapas puede visitarse nuevamente en otras ocasiones, el proceso puede terminar en cualquiera de estas actividades.

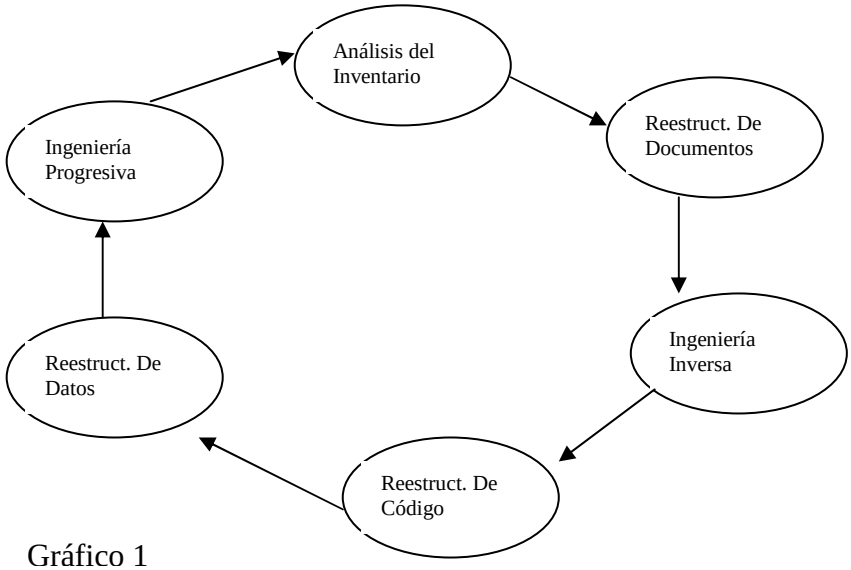


Gráfico 1

1.1.2.1 Análisis de inventario

Hace referencia a los datos que se deben disponer en las aplicaciones de software existentes en una empresa:
- nombre de la aplicación
- año de creación
- número de cambios importantes realizados en ella

- esfuerzo para aplicar estos cambios (horas/hombre)
- fecha último cambio
- sistema en que reside
- aplicaciones con las cuales tiene relación
- base de datos a las que accede
- errores presentados en los últimos 18 meses
- número de usuarios
- números de equipos en los que está instalado
- complejidad: arquitectura, código, documentación
- calidad de la documentación
- mantenibilidad general
- longevidad del proyecto
- costo anual de mantenimiento
- costo anual de funcionamiento
- valor anual de negocios
- importancia para el negocio

De la información recolectada para cada aplicación, aparecen los posible candidatos para reingeniería.

1.1.2.2 Reestructuración de documentos

Se plantean alternativas ante una situación de documentación escasa:
- Representa el caso de programas de los cuales no tenemos documentación, pero que no nos acompañarán por mucho tiempo, en estos casos se recomienda dejar las cosas así.
- Si no se dispone de los recursos suficientes, se recomienda documentar únicamente lo que se ha modificando, con el tiempo se dispondrá de una basta documentación.
- Redocumentar únicamente aquellos procesos críticos para el negocio.

1.1.2.3 Ingeniería Inversa

El término **Ingeniería Inversa** tiene sus orígenes en el mundo del hardware, en donde una empresa desensambla un equipo de la competencia para comprender los secretos del diseño y construcción del mismo.
Estos secretos se podrían comprender fácilmente si se obtuvieran las especificaciones de diseño y construcción elaboradas por el fabricante.
En el mundo del software, el principio es el mismo, solo que no necesariamente aplica a productos de un competidor, sino de la misma compañía elaborados anteriormente.
Consiguientemente, la ingeniería inversa del software es el proceso que consistente en analizar un programa en un esfuerzo por crear una representación del mismo con un nivel de abstracción más elevado que el del código fuente.

A diferencia del mantenimiento, la reingeniería implica afectar la estructuras del sistema.

La ingeniería inversa es un proceso de recuperación del diseño.
Las herramientas de ingeniería inversa extraen información acerca de los datos, arquitectura y diseño de procedimientos de un programa ya existente

La ingeniería inversa invoca una imagen de *ranura mágica*, en donde por un lado se inserta un listado de código no estructurado no documentado, y por el otro lado sale la documentación completa del programa, desafortunadamente esta no existe, la ingeniería inversa puede extraer la información de diseño acerca del código fuente, pero el nivel de abstracción, la completitud de la documentación, el grado en el cual trabajan al mismo tiempo las herramientas y el analista humano y la direccionalidad del proceso son sumamente variables.

- ***Nivel de abstracción:*** aluden a la sofisticación de la información de diseño que se puede extraer del código fuente, lo ideal sería un nivel alto de abstracción.

- ***La completitud:*** alude al nivel de detalle que se proporciona de un determinado nivel de abstracción, en la mayoría de los casos, la completitud decrece a medida que aumenta el nivel de abstracción.

- ***La interactividad:*** alude al grado con el cual el ser humado se integra con herramientas automatizadas para crear un proceso de ingeniería inversa

- ***La direccionalidad:*** es monodireccional**,** es decir, el único sentido posible es del código fuente al diseño.

El ingeniero tiene que evaluar el viejo programa y a partir del código fuente tiene que extraer una especificación significativa del proceso que se realiza, la interfaz de usuario que se aplica y las estructuras de datos de programa o de la base que utiliza.

- Ingeniería inversa para comprender el proceso: La primera actividad de ingeniería inversa intenta comprender y luego hacer abstracciones de procedimientos representadas en código fuente.
Para comprender las abstracciones de procedimientos se analiza el código en distintos niveles de abstracción, sistema, programa, módulo, trama y sentencia.
La funcionalidad de todo el sistema debe ser algo perfectamente comprendido antes que se produzca un trabajo de ingeniería inversa más detallado.
Cada uno de los programas de que consta el sistema de aplicaciones representará una abstracción funcional con un elevado nivel de detalle.

- Ingeniería inversa para comprender los datos: La ingeniería inversa de datos suele producirse en diferentes niveles de abstracción.
En el nivel de programa, es frecuente realizar una ingeniería inversa de las estructuras de datos de programa internas, como parte de un esfuerzo global de reingeniería.
En nivel de sistema es frecuente que se efectúe una reingeniería de las estructuras globales de datos, por ejemplo archivos y bases de datos.

Estos conceptos se ilustran en el gráfico 2.

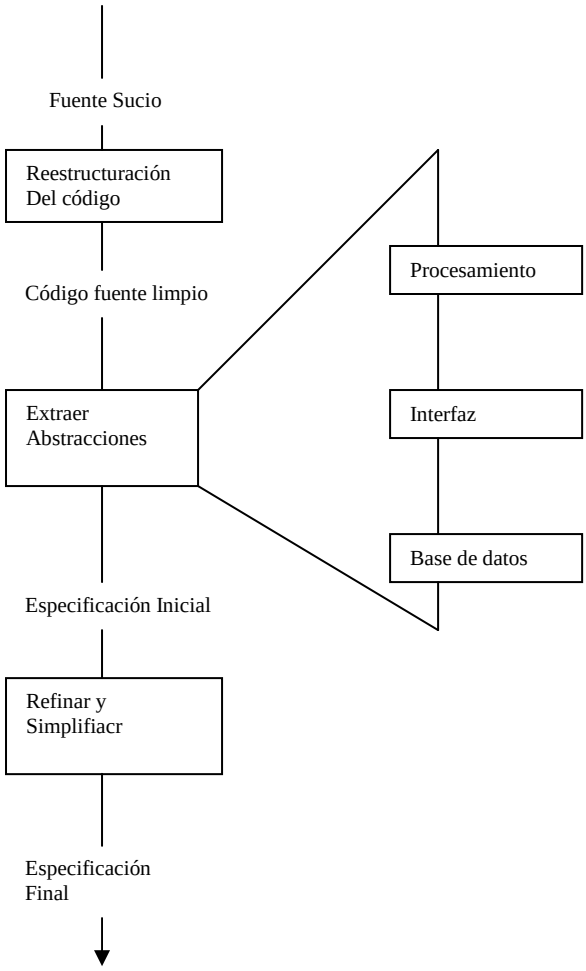


Gráfico 2

Estructuras Internas: Las técnicas de ingeniería inversa para datos de programa internos se centran en la definición de *clases de objetos*, esto se logra analizando el código del programa con la idea de agrupar variables relacionadas.
Se identifican indicadores y estructuras de datos locales dentro del programa que registren información importante acerca de estructuras de datos globales.
Se define la relación entre estructuras y datos locales con estructuras de datos globales.
Estos pasos capacitan al ingeniero del software para identificar las clases dentro del programa que interactúan con las estructuras de datos globales.

Estructuras de bases de datos: Independientemente de su organización lógica y estructura física, las bases de datos permiten definir objetos de datos, y admiten algún método para establecer relaciones entre objetos.
La reingeniería de un esquema de bases de datos para formar otro, exige una comprensión de los objetos ya existentes y de sus relaciones.
. Se construye un modelo de objetos inicial.
. Se determinan los candidatos a claves
. Se refinan las clases tentativas
. Se definen las generalizaciones
. Se descubren las asociaciones.

- Ingeniería inversa de interfaces de usuario: A medida que las GUI (graphic user interface) se han vuelto sofisticadas y los entornos gráficos se hacen más comunes, el desarrollo de interfaces de usuario ha pasado a ser uno de los tipos más comunes de actividades de reingeniería.
Para comprender una interfaz de usuario ya existente, es preciso especificar la estructura y comportamiento de la interfaz.
Merlo y sus colaboradores[2] sugieren tres preguntas a responder cuando se inicia una actividad de ingeniería inversa de la interfaz de usuario:
. ¿Cuáles son las acciones básicas que debe procesar la interfaz , como teclado y clic del ratón. ?
. ¿Cuál es la descripción de la respuesta del sistema a estas acciones ?
. ¿ Qué concepto de equivalencia de interfaces es relevante en este caso?
La descripción de la interfaz hace uso de *agentes y acciones.*
Un agente es algo que da vida a algún aspecto del sistema, las acciones permiten que los agentes se comuniquen entre sí, en escencia el álgebra de procesos es una notación taquigráfica para representar la forma en que los agentes y las acciones dan vida a la función de una interfaz de usuario.

1.1.2.4 La reestructuración.

La reestructuración del software modifica el código fuente y/o los datos en un intento de hacerlo adecuado para cambios futuros. En general, la reestructuración no modifica la arquitectura global del programa.
Tiende a centrarse en los detalles de diseño de módulos individuales y en estructuras de datos locales definidas dentro de los módulos.

Si el esfuerzo se extiende más allá de los límites de los módulos y abarca a la arquitectura del software, la estructuración pasa a ser *ingeniería progresiva*.

Algunos de las ventajas de la reestructuración son:
. Programas de mayor calidad, con mayor documentación y menos complejidad.
. Mejora la comprensión de los ingenieros que deban trabajar con el programa, mejorando por tanto la productividad, y haciendo más sencillo el aprendizaje.
. Reduce el esfuerzo en actividades de mantenimiento.
. Hace al oftware más sencillo de comprobar y depurar.

Reestructuración de código: se lleva a cabo para conseguir un diseño que produzca la misma función pero con mayor calidad que el programa original. En general, las reglas de reestructuración de código modelan la lógica del programa empleando álgebra Booleana.
El objetivo es tomar el código en forma de *plato de spaguetis* y derivar a partir de él en un diseño de procedimientos que se ajuste a la filosofía de la programación estructurada.

Reestructuración de datos: antes que pueda comenzar la reestructuración de los datos, es preciso llevar a cabo una ingeniería inversa , un análisis del código fuente.

[2] Ingeniería del Software, Pressman

Se evaluarán primeramente todas las sentencias del lenguaje que contengan definiciones de datos, descripciones de archivos, entrada / salida y descripciones de interfaz.
El objetivo es extraer elementos y objetos de datos para obtener información acerca del flujo de datos, así como entender las estructuras de datos existentes. Esta actividad se denomina *análisis de datos.*

Una vez finalizado el análisis de datos empieza el *rediseño de datos*. En su forma más sencilla se emplea un paso de estandarización de rediseño de datos que clarifica las definiciones de datos para lograr una consistencia entre nombres de objetos de datos o entre formatos de registros físicos.

Otra forma de rediseño denominada *racionalización de nombres de datos a*segura que todas las convenciones de normalización de datos se ajusten a los estándares locales y asegura también que se eliminen los alias a medida que fluyen los datos a través del sistema.
Cuando la reestructuración va más allá de la estandarización y de la racionalización, se efectúan modificaciones físicas en las estructuras de datos ya existentes con objeto de hacer que el diseño de datos sea más efectivo.
Esto puede significar una traducción de un formato de archivo a otro o en algunos casos, una traducción de un tipo de base de datos a otra.

1.1.2.5 Ingeniería progresiva

Un programa con un flujo de control equivalente a un plato de *espaguetis* y que no posea documentación, debe ser modificado para adecuarse a los requisitos del usuario. Existen las siguientes opciones:
. Realizar el esfuerzo necesario para lograr implementar los cambios.
. Tratar de entender el funcionamiento interno para que las modificaciones sean más efectivas
. Rediseñar, recodificar y probar solo aquellas partes que necesitan ser cambiadas, aplicando enfoques de ingeniería de software.
. Rediseñar, recodificar y probar el sistema en su totalidad aplicando herramientas CASE

No existe una opción única, pues pueden utilizarse según las circunstancias.
Miller define la ingeniería progresiva como[3]*:* ***la aplicación de las metodologías de hoy a los sistemas de ayer para prestar apoyo a los requisitos del mañana.***

A primera vista, puede parecer extravagante la reprogramación de sistemas que se encuentran funcionando, sin embargo debemos considerar lo siguiente:
. El costo de una línea de mantenimiento puede estar entre 30 y 40 veces por encima del costo de desarrollo.
. El rediseño de la estructura del software empleando conceptos modernos puede facilitar mucho el futuro mantenimiento.
. Los usuarios tienen conocimiento del software, por lo tanto los nuevos requisitos de cambio pueden manejarse más fácilmente.

[3] Ingeniería del Software, Pressman

. Las herramientas CASE para reingeniería pueden hacer algunas tareas automáticamente.
. Cuando termine el mantenimiento preventivo, se dispondrá de una configuración completa de software, (documentos programas y datos).

El proceso de ingeniería progresiva aplica principios, conceptos y métodos de ingeniería de software para volver a crear las aplicaciones existentes.

En la mayoría de los casos, la ingeniería progresiva no se limita a crear un equivalente moderno de un programa anterior, más bien se integran los nuevos requisitos y las nuevas tecnologías en ese esfuerzo de reingeniería.

El programa desarrollado de nuevo extiende las capacidades de la aplicación anterior.

Ingeniería Progresiva para arquitecturas O.O. La ingeniería de software orientada a objetos se está transformando rápidamente en el paradigma de desarrollo de elección para muchas organizaciones de software.

Sin embargo debemos ver la posibilidad de migrar todas aquellas aplicaciones que se han desarrollado considerando metodologías tradicionales.

En primer lugar se hace ingeniería inversa del software existente para que sea posible crear los modelos adecuados de datos, funcional y de comportamiento. Los modelos de datos creados durante la ingeniería inversa, se utilizan para establecer la base para la definición de clases.

Las jerarquías de clases, los modelos de relaciones entre objetos, los modelos de comportamiento de objetos y los subsistemas se definen a continuación y comienza el diseño orientado a objetos.

Si la aplicación existente posee un dominio que ya contiene aplicaciones orientadas a objetos, es probable que exista una robusta biblioteca reutilizable que podemos usar durante la ingeniería progresiva.

Para aquellas clases que sea preciso construir partiendo de cero, quizá sea posible reutilizar algoritmos y estructuras de datos procedentes de la aplicación convencional original.

Ingeniería Progresiva para (arquitecturas) interfaces de usuario: Se aplica especialmente cuando se migran aplicaciones de ambientes centralizados a ambientes distribuidos, pues los usuarios no están dispuestos a aceptar interfaces basadas en caracteres, precisan de ambientes gráficos modernos.

De hecho, una gran parte del esfuerzo invertido en la transición de ambientes centralizados a cliente servidor, se pueden reinvertir en la reingeniería de interfaces de usuario de la aplicación cliente.

Merlo y sus colaboradores[4] suguieren el siguiente modelo:

[4] Ingeniería del Software, Pressman

. Comprender la interfaz original y los datos que se trasladas entre ella y el resto de la aplicación. Si se ha de desarrollar una nueva IGU, entonces el flujo de datos entre la IGU y el resto del programa debe ser consistente con los datos que en la actualidad fluyen entre la interfaz basada en caracteres y el programa.
. Remodelar el comportamiento implícito de la interfaz existente para formar una serie de abstracciones que tengan sentido en el contexto de una IGU. Aún cuando el modo de la interacción sea radicalmente distinto, el comportamiento de negocios interno debe seguir siendo el mismo.
. Introducir mejoras que hagan que el modo de interacción sea más eficiente.

1.1.3 Economía de la reingeniería

En un ambiente ideal, todo programa que no se pueda mantener, se retiraría inmediatamente para ser sustituido por aplicaciones de alta calidad, fabricadas mediante reingeniería y desarrolladas empleando prácticas de ingeniería de software modernas.

Sin embargo, por las limitantes de recursos deben considerarse costos y prioridades a los procesos a ser sometidos a reingeniería.

Sneed[5] propone un modelo de costo y beneficio basado en nueve parámetros:
P1 = Ccosto de mantenimiento anual de una aplicación.
P2 = Costo de operación anual
P3 = Valor de negocios real actual
P4 = Costo de mantenimiento anual previsto (después de la reingeniería)
P5 = Costo de operaciones anual previsto (después de la reingeniería)
P6 = Valor de negocios previsto después de la reingeniería
P7 = Vostos estimados de la reingeniería
P8 = Fecha estimada de reingeniería
P9 = Factor de riesgo de la reingeniería. (1,0 valor nominal)
L = Vida esperada del sistema (en años)

El costo asociado al mantenimiento continuo se define como:

$$Cmant = [P3 – (P1 + P2)] \times L$$

El costo asociado a la reingeniería se define como:

$$Creing = [P6 – (P4 + P5) \times (L – P8) – (P7 \times P9)]$$

Empleando las formulas anteriores, tenemos el beneficio como:

$$Beneficio = Creing – Cmant$$

Aquellas aplicaciones que muestren mayor beneficio, son candidatas potenciales a aplicar reingeniería.

[5] Ingeniería del Software, Pressman

1.2 OMT

Modelamiento y diseño orientado a objetos constituye una nueva forma de pensar acerca de problemas empleando modelos que se han organizado tomando como base conceptos del mundo real.
La construcción fundamental es el objeto que combina las estructuras de datos con los comportamientos en una entidad única.
Los modelos orientado a objetos son útiles para comprender los problemas, comunicarse con expertos en esa aplicación, modelar empresas, preparar documentación y diseñar programas y bases de datos.

Una de las metodologías orientado a objetos es la llamada **Técnica de Modelado de Objetos (OMT)** , que se extiende desde el análisis hasta la implementación pasando por el diseño.

En primer lugar se construye un modelo de análisis para abstraer los aspectos esenciales del dominio de la aplicación sin tener en cuenta la implementación eventual. Estos objetos constituyen el marco de trabajo del modelo de diseño, por último el modelo de diseño se implementa en algún lenguaje de programación, base de datos, y hardware.

Se utiliza una notación gráfica para expresar los modelos orientados a objetos.
Los objetos de dominio de la aplicación y del dominio de la computadora se pueden modelar, diseñar e implementar utilizando los mismos conceptos y la misma notación orientados a objetos.

Orientado a objetos: En términos simples significa que el software se organiza como una colección de objetos discretos que contienen tanto estructuras de datos como un comportamiento. Esto se opone a la programación convencional, en la cual las estructuras de datos y el comportamiento solamente están relacionadas de forma débil. Existe un cierto desacuerdo en cuanto a las características precisas que requiere un enfoque orientado a objetos aunque suelen incluirse cuatro aspectos: identidad, clasificación, polimorfismo y herencia.

. Identidad: quiere decir que los datos están cuantificados en entidades discretas y distinguibles denominadas objetos, Ej. Una ventana de mi estación de trabajo, un párrafo de un documento, la reina blanca de un juego de ajedrez.

. Clasificación: Significa que los objetos con la misma estructura de datos (atributos) y comportamiento (operaciones) se agrupan para formar una clase. Se dice que cada objeto es una *instancia* de su clase. Se grafica un ejemplo en el gráfico 3.

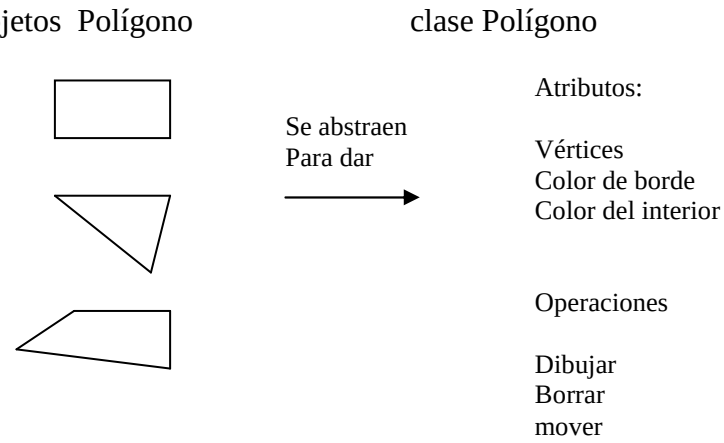


Gráfico 3

. Polimorfismo: significa que una misma operación puede comportarse de modos distintos en distintas clases. La operación *mover* por ej. Puede comportarse de diferente manera en las clases *ventana y pieza de ajedrez.*
Una operación es una acción que se le aplica a un objeto.

. Herencia: es compartir atributos y operaciones entre clases, tomando como base una relación jerárquica.
En términos generales se puede definir una clase que después se irá refinando sucesivamente para producir *subclases*. Todas las subclases poseen o heredan todas y cada una de su superclase, y añaden además sus propiedades exclusivas. No es necesario repetir las propiedades de la clase en sus subclases.

1.2.1 Metodología Orientada a Objetos

La metodología consiste en construir un modelo de un dominio de aplicación añadiéndose detalles de implementación durante el diseño de un sistema.
Esta aproximación se denomina la Técnica de Modelado de Objetos (OMT), la cual consta de las siguientes fases:

1.2.1.1 Análisis

Comenzando desde la descripción del problema, el analista construye un modelo de la situación del mundo real que muestra sus propiedades importantes.
Dicho analista debe trabajar con quien hace la solicitud para comprender el problema más porque las definiciones del mismo no suelen ser completas ni correctas.
El modelo del análisis es una abstracción resumida y precisa de lo que debe hacer el sistema deseado y no de la forma en que se hará.
Los objetos del modelo deberán ser conceptos del dominio de la aplicación y no conceptos de implementación de la computadora tales como estructuras de datos.
Un buen modelo podrá ser comprendido y criticado por expertos de la aplicación que no sean programadores.
El modelo de análisis no debe contener ninguna decisión de implementación.

1.2.1.2 Diseño del sistema

El diseñador de sistemas toma decisiones de alto nivel acerca de la arquitectura global. Durante el diseño, el sistema de destino se organiza en subsistemas basados tanto en la estructura del análisis como en la arquitectura propuesta.
El diseñador del sistema deberá decidir qué características de rendimiento hay que optimizar, seleccionando una estrategia para atacar el problema y efectuando unas reservas de recursos tentativas.

Por ejemplo el diseñador del sistema podría decidir que los cambios habidos en la pantalla de la estación de trabajo sean rápidos y suaves aunque se muevan o borren ventanas y seleccionará un protocolo de comunicaciones adecuado así como una estrategia de reserva de memoria

1.2.1.3 Diseño de objetos

El diseñador de objetos construye un modelo de diseño basándose en el modelo de análisis que lleva incorporados detalles de implementación.
El diseñador añade detalles al modelo de acuerdo con la estrategia establecida durante el diseño del sistema.

El foco de atención del diseño de objetos son las estructuras de datos y los algoritmos necesarios para implementar cada una de las clases.
Las clases de objetos procedentes del análisis siguen siendo significativas pero se aumentan con estructuras de datos y algoritmos del dominio de la computadora seleccionados para optimizar medidas importantes de rendimiento.

Tanto los objetos del dominio de la aplicación como los objetos del dominio de la computadora se describen utilizando unos mismos conceptos y una misma notación orientado a objetos aún cuando existan en planos conceptuales diferentes.

1.2.1.4 Implementación

Las clases de objetos y las relaciones desarrolladas durante su diseño se traducen finalmente en un lenguaje de programación concreto, a una base de datos o a una implementación de hardware.
La programación debería ser una parte relativamente pequeña del ciclo de desarrollo y fundamentalmente mecánica porque todas las decisiones importantes deberán hacerse durante el diseño.

El lenguaje influye en cierta medida sobre las decisiones de diseño pero este no debería depender de la estructura final de un lenguaje de programación.
Durante la implementación es importante respetar las buenas ideas de la ingeniería del software , de tal manera que el seguimiento hasta el diseño sea sencillo y de forma que el sistema implementado siga siendo flexible y extensible.

Es posible aplicar conceptos orientado a objetos a lo largo de todo el ciclo de vida de desarrollo del sistema.

1.2.2 Temas orientados a objetos

Hay varios temas que subyacen a la tecnología orientada a objetos, aunque estos temas no son exclusivos de los sistemas orientada a objetos, están bien apoyados especialmente por es esta tecnología.

1.2.2.1 Abstracción

La abstracción consiste en centrarse en los aspectos esenciales inherentes a una entidad, e ignorar sus propiedades accidentales. En el desarrollo de sistemas esto significa centrarse en lo que es y lo que hace un objeto antes de decidir cómo debería ser implementado.
El uso de la abstracción mantiene nuestra libertad de tomar decisiones durante el mayor tiempo posible evitando comprometernos de forma prematura con ciertos detalles.

La mayoría de los lenguajes modernos proporcionan abstracción de datos pero la capacidad de utilizar herencia y polimorfismo proporciona una potencia adicional.
El uso de la abstracción durante el análisis significa tratar solamente conceptos del dominio de la aplicación y no tomar decisiones de diseño o de implementación antes de haber comprendido el problema.

Un uso adecuado de la abstracción permite utilizar el mismo modelo para el análisis, diseño de alto nivel , estructura del programa, estructura de una base de datos y documentación.

Un estilo de diseño independiente del lenguaje pospone los detalles de programación hasta la fase final, relativamente mecánica del desarrollo.

1.2.2.2 Encapsulamiento

O encapsulación, se denomina también *ocultamiento de información,* y consiste en separar los aspectos externos del objeto, a los cuales pueden acceder otros objetos, de los detalles internos de implementación del mismo, que quedan ocultos para los demás.

La encapsulación evita que el programa llegue a ser tan interdependiente que un pequeño cambio tenga efectos secundarios masivos.
La implementación de un objeto se puede modificar sin afectar a las aplicaciones que lo utilizan. Quizá sea necesario modificar la implementación de un objeto para mejorar el rendimiento, corregir un error, consolidar el código o para hacer un transporte a otra plataforma.

La encapsulación no es exclusiva de los lenguajes orientados a objetos, pero la capacidad de combinar la estructura de datos y el comportamiento en una única entidad hace que la encapsulación sea aquí más limpia y potente que en los lenguajes convencionales que separan las estructuras de datos y el comportamiento.

1.2.2.3 Combinación de datos y comportamiento

Aquel que invoca una operación no necesita considerar cuantas implementaciones existen de una operación dada. El *polimorfismo* de operadores traslada la carga de decidir qué implementación hay que utilizar llevándola del código que hace la llamada a la jerarquía de clases.

Por ejemplo, en código no orientado a objetos, para visualizar el contenido de una ventana debe distinguir el tipo de cada figura (un polígono, círculo, texto) y debe invocar al procedimiento adecuado para visualizarlo.

Un programa orientado a objetos invocaría simplemente la operación dibujar para cada figura; la decisión del procedimiento que hay que utilizar es realizada implícitamente por cada objeto basándose en su clase. No es necesario repetir la selección de procedimiento cada vez que se invoque a la operación en el programa de aplicación.

El mantenimiento es más sencillo porque el código que hace la llamada no necesita ser modificado cuando se añade una clase nueva.
En un sistema orientado a objetos la jerarquía de estructuras de datos es idéntica a la jerarquía de operaciones, como se muestra en el gráfico 4.

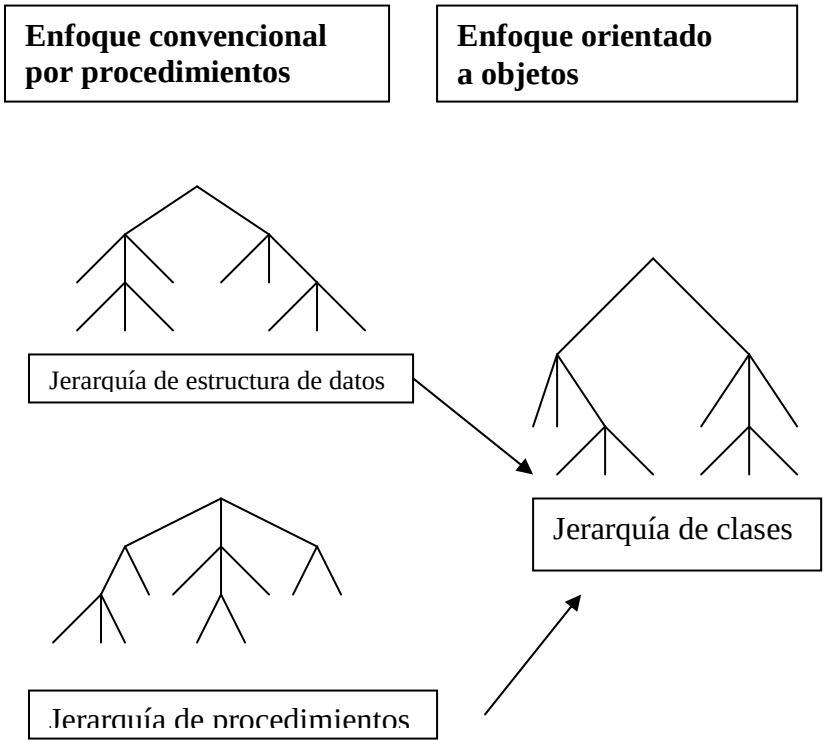


Gráfico 4

El enfoque orientado a objetos tiene una sola jerarquía unificada.

1.2.2.4 Compartir

Las técnicas orientadas a objetos impulsan a compartir en distintos niveles. La *herencia* tanto de estructuras de datos como de comportamientos, permite compartir una estructura común entre varias subclases similares sin redundancia.

Una de las principales ventajas de los lenguajes orientados a objetos es compartir código empleando la herencia. Todavía más importante que el ahorro de código es la claridad conceptual que surge al reconocer que distintas operaciones son todas ellas, realmente, una misma cosa.

Esto reduce el número de clases distintas que es preciso comprender y analizar.
El desarrollo orientado a objetos no sólo permite compartir información dentro de una aplicación sino que, además ofrece la perspectiva de volver a utilizar diseños y códigos en futuros proyectos.

Aún cuando esta posibilidad se ha hecho resaltar excesivamente como justificación de la tecnología orientada a objetos, el desarrollo orientado a objetos no es una fórmula mágica para asegurar la reutilización del código, sin embargo. La reutilización no sucede; debe ser planteada pensando más allá de la aplicación inmediata e invirtiendo un esfuerzo adicional en un diseño más general.

1.2.2.5 Énfasis en la estructura de objetos, no de los procedimientos

La tecnología orientada a objetos hace hincapié en especificar lo que es un objeto más que en la forma en que éste se utiliza.

Las aplicaciones de un objeto dependen mucho de los detalles de la aplicación y suelen cambiar durante el desarrollo. A medida que evolucionan los requisitos, las posibilidades proporcionadas por un objeto son mucho más estables que las formas en que se utilizan; por tanto los sistemas de software construidos sobre una estructura de objetos son más estables en el tiempo.

El desarrollo orientado a objetos pone un mayor énfasis en la estructura de objetos y hace menos hincapié en la estructura de procedimientos que las metodologías tradicionales de descomposición funcional.

En este aspecto, el desarrollo orientado a objetos es parecido a las técnicas de modelado de información que se utilizan en el diseño de base de datos, si bien el desarrollo orientado a objetos añade el concepto de comportamiento dependiente de la clase.

1.2.2.6 Sinergia

La identidad, clasificación, polimorfismo y herencia caracterizan los principales lenguajes orientados a objetos. Cada uno de estos conceptos puede ser utilizado aisladamente, pero en su conjunto, se complementan unos a otros de forma *sinérgica.*

Los beneficios de un enfoque orientado a objetos son mayores de lo que podría pensarse en un principio. El hincapié hecho en las propiedades esenciales de los

objetos obliga al desarrollador de software a pensar en forma más clara y profunda acerca de lo que es y de lo que hace el objeto, alcanzando un resultado consistente en que el sistema puede ser más limpio, más general y más robusto de lo que sería si el hincapié se hiciera únicamente sobre el uso de datos y de operaciones.
Según Thomas[6], estas características se unen para crear un estilo distinto de programación.

Cox[7] afirma que la encapsulación es el fundamento de la programación orientado a objetos desplazando el énfasis de la técnica de codificación al empaquetamiento mientras que la herencia se basa en la encapsulación para hacer práctica la reutilización del código.

1.2.3 Modelos de OMT

Resulta útil modelar un sistema desde tres puntos de vista distintos, aunque relacionados, cada uno de los cuales captura aspectos importantes del sistema, pero siendo todos ellos necesarios para una descripción completa.
La técnica de modelado de Objetos (OMT) es el nombre que se da a una metodología que combina estos tres puntos de vista para el modelado de sistemas.
El *modelo de objetos* representa los aspectos estáticos, estructurales "de datos" del sistema.
El *modelo dinámico* representa los aspectos temporales, de comportamiento "de control" del sistema.
El *modelo funcional* representa los aspectos transformacionales "de función" del sistema.

Un procedimiento típico de software contiene estos tres aspectos, utiliza las estructuras de datos (modelo de objetos) secuencia las operaciones en el tiempo (modelo dinámico) y transforma valores (modelo funcional).
Cada modelo contiene referencias a entidades de otros modelos, por ejemplo, las operaciones se asocian a los objetos en el modelo de objetos pero se expanden de forma más completa en el modelo funcional.
Las tres clases de modelos desglosan el sistema en puntos de vista ortogonales que se pueden representar y manipular empleando una notación uniforme.
Los distintos modelos no son completamente independientes (un sistema es algo más que una colección de partes independientes) pero cada modelo puede ser examinado y comprendido por sí mismo en gran parte.

Las interconexiones entre los distintos modelos son limitadas y explícitas. Por su puesto siempre es posible crear diseños malos en los cuales los tres modelos estén tan entremezclados que no sea posible separarlos, los buenos diseños en cambio aíslan los distintos aspectos del sistema y limitan el acoplamiento entre ellos.

[6] El Lenguaje Unificado de Modelado, Booch, Rumbaugh, Jacobson
[7] El Lenguaje Unificado de Modelado, Booch, Rumbaugh, Jacobson

1.2.3.1 Modelo de Objetos

Este modelo captura la estructura estática del sistema, mostrando sus objetos, sus relaciones, y los atributos que caracteriza a cada clase.
Este es el más importante de los tres modelos, y proporcionan una representación gráfica intuitiva del sistema, mediante la cual podemos comunicarnos con los clientes y documentar las estructuras del sistema.

***Objetos**:* El propósito esencial del modelado de objetos es describir objetos.
Por ejemplo son objetos: Juan Pérez, la empresa XYZ, el proceso de compras, un objeto es sencillamente algo que tiene sentido en el contexto de una aplicación.

Un objeto se define como un concepto, abstracción o cosa con límites bien definidos y con significado a efectos del problema que se tenga que resolver.
Los objetos tienen dos propósitos: promover la comprensión del mundo real y proporcionar una base para la implementación en computadora.
La descomposición del problema en objetos depende del juicio y de la naturaleza del problema, pueden existir varias representaciones correctas.
Todos los objetos poseen identidad propia y se pueden distinguir entre sí.

El término *identidad* significa que los objetos se distinguen por su existencia inherente y no por sus propiedades descriptivas.
Cuando se desea ser preciso y aludir a una cosa exactamente se utiliza la frase *instancia de objeto,* y para aludir a un grupo de cosas similares, *clase de objeto.*

***Clases**:* Una clase de objetos describe un grupo de objetos con atributos y propiedades similares, con relaciones comunes y con una semántica común.
Ejemplo de clases de objetos son: persona, compañía, proceso; una persona tiene edad, coeficiente intelectual, y puede realizar cierto trabajo.
Los objetos y sus clases pueden aparecer como sustantivos en las descripciones de problemas.

Frecuentemente se utiliza la abreviatura *clase* en lugar de *clase de objeto*. Los objetos de una clase tienen los mismos atributos y los mismos patrones de comportamiento
Los objetos de una clase comparten un propósito semántico común, más allá de los requisitos de comunidad de atributos y de comportamiento, por tanto, aún cuando un granero y un caballo tienen un coste y una edad, pueden pertenecer a diferentes clases, pero si se considera al granero y al caballo como activos financieros, pueden pertenecer a la misma clase.

La interpretación de la semántica depende del proceso de cada aplicación.
Cada objeto conoce a su clase, la mayoría de los lenguajes de programación orientada a objetos pueden determinar la clase del objeto al momento de la ejecución.
Al agrupar objetos en clases se abstrae el problema, la abstracción da al modelado su potencia y capacidad de generalizar, partiendo de unos pocos casos específicos hasta llegar a una multitud de casos similares. Las definiciones comunes se almacenan una sola vez en cada clase, en lugar de almacenarse una vez por instancia. Es posible escribir operaciones una sola vez para cada clase, de forma que todos los objetos de la clase se beneficien utilizándola.

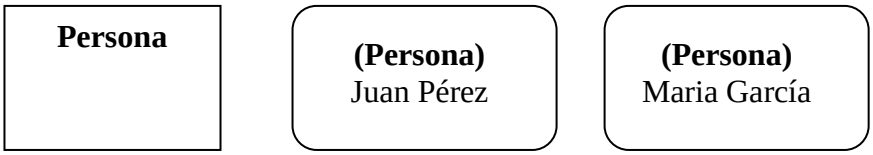
. ***Diagramas de objetos****:* Proporcionan una notación gráfica formal para el modelado de objetos, clases y sus relaciones entre sí, son útiles tanto para el modelado abstracto como para diseñar programas reales. Los diagramas de objetos son concisos, fáciles de entender y funcionan bien en la práctica.
Existe dos tipo de diagramas de objetos, diagramas de clases y diagramas de instancias.

. **Diagramas de Clases**: es un esquema, patrón o plantilla para describir muchas instancias de datos posibles, describen clases de objetos.
. **Diagrama de Instancias**: describe la forma en que un cierto conjunto de objetos se relacionan entre sí, describe instancias de objetos, son útiles para documentar casos prácticos, escenarios, y para describir ejemplos.

Un diagrama de clases dado, se corresponde con un conjunto infinito de diagramas de instancia.

Ilustración en el gráfico 5:

Gráfico 5



Clase Obietos

Diagrama de clases a la izquierda y un posible diagrama descrito por el primero, los objetos Juan Pérez y María García son instancias de la clase persona.
El símbolo OMT de un objeto es un cuadro con esquinas redondeado, con el nombre de la clase entre paréntesis y en negrillas.
El símbolo OBM de una clase es un cuadrado con el nombre de la clase en negrillas.

Los diagramas de clases describen el caso general al modelar un sistema, los de instancias se utilizan para mostrar ejemplos que nos ayuden a clarificar los diagramas de clases complejos.

Los diagramas de clases y de instancias pueden aparecer en un mismo diagrama de objetos, pero en general no resulta útil mezclarlos.

Atributos*:* Se lo define como un valor de un dato que está almacenado en los objetos de una clase, por ejemplo nombre, edad, peso son atributos del objeto de tipo Persona.
Cada atributo tiene un valor para cada instancia del objeto, por ejemplo el atributo edad tiene un valor de 40 en el objeto Juan Pérez.
El nombre de un atributo es único dentro de una clase, pero diferentes clases pueden tener un mismo atributo, por ejemplo la clase *persona* y la clase *compañía* pueden tener el atributo *dirección*.
Los atributos se enumeran en la segunda parte del cuadro de clase, pudiendo ir seguido del tipo y valor por omisión, donde el tipo irá precedido por : , y el valor por omisión precedido por un = .

Los cuadros de clase tiene una línea para dividir el nombre de los atributos, los cuadros de objetos no. Se Ilustra en el gráfico 6

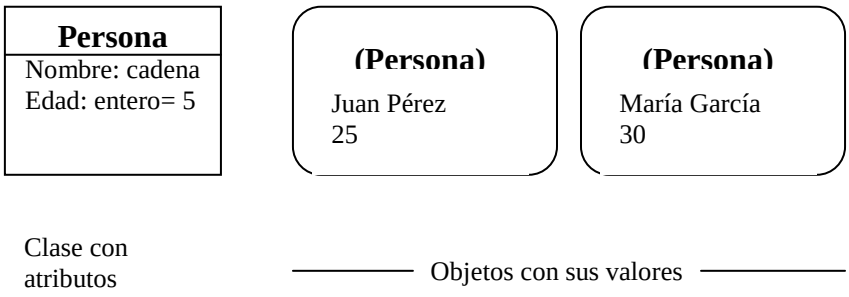


Gráfico 6

Algunos medios de implementación como bases de datos, exigen un identificador (ID) único que reconozca a cada objeto, pero estos identificadores de objeto explícitos no son necesarios en el modelado de objetos, cada uno posee su propia identidad única.

No hay que confundir los identificadores internos con atributos del mundo real, los identificadores internos son solamente una comodidad de implementación, y no tienen significado en el dominio del problema.

Operaciones y métodos*:* Se define como operación a una función o transformación que se puede aplicar a todos los objetos de una clase.
Ejemplos de operaciones una clase *Compañía* son contratar, despedir, pagar impuestos. Todos los objetos de una clase comparten las mismas operaciones
Cada operación tiene un objeto blanco como argumento implícito, el comportamiento de la operación depende de la clase de su blanco.
Una misma operación puede aplicarse a muchas clases distintas, tal operación será polimórfica, esto significa que una misma operación adopta distintas formas en distintas clases.

Un método es la implementación de una operación para una clase, por ejemplo la clase *archivo* puede tener una operación llamada *imprimir* , se podrían implementar distintos métodos para imprimir archivos de diferentes tipos, cada uno de estos efectúa una misma tarea lógica (imprimir) sin embargo, cada método puede estar implementado mediante un segmento de código diferente adecuado para cada tipo de archivo.

Una operación puede poseer argumentos, tales argumentos parametrizan la operación.

Cuando una operación posee métodos aplicables a distintas clases es importante que todos los métodos tengan la misma signatura, por ejemplo *imprimirl*o debería tener como argumento el nombre-de-archivo para un método y puntero-del-archivo para otro. El comportamiento de todos los métodos de una operación debería tener una internacionalidad común, lo mejor es no utilizar el mismo nombre para dos operaciones que sean semánticamente distintas, aún cuando sean aplicables a conjuntos distintos.

Las operaciones se enumeran en el tercio inferior del cuadro de clase, el nombre de cada operación puede ir seguido de detalles opcionales como la lista de argumentos y el tipo de resultado.

Una lista de argumentos se escribirá entre paréntesis a continuación del nombre, los argumentos irán separados por comas (,) . El nombre y tipo de argumento pueden especificarse también. El tipo y el resultado vienen precedidos por dos puntos (:) y no deberían omitirse. Una lista de argumentos vacía entre paréntesis muestra que no existe argumentos. Ejemplos en los gráficos 7 8 y 9

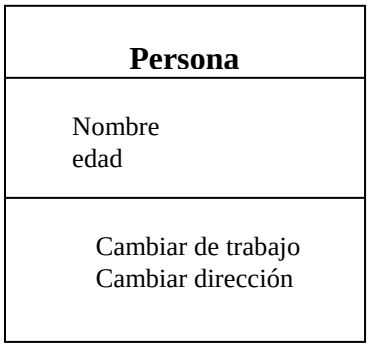


Gráfico 7

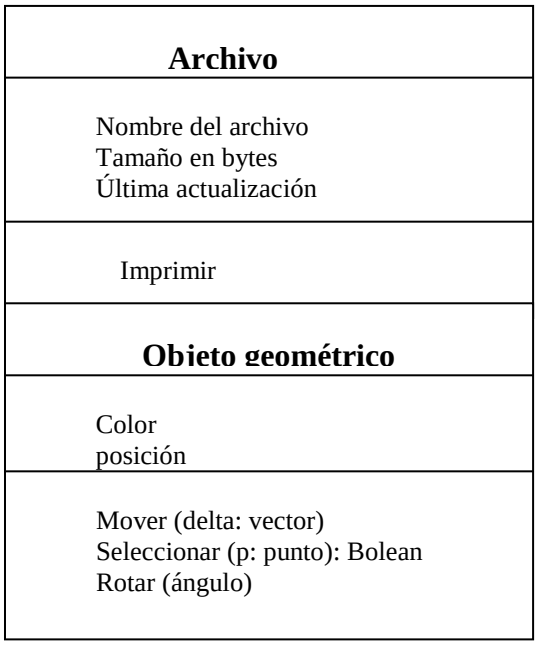


Gráfico 8

Resumen de la notación:

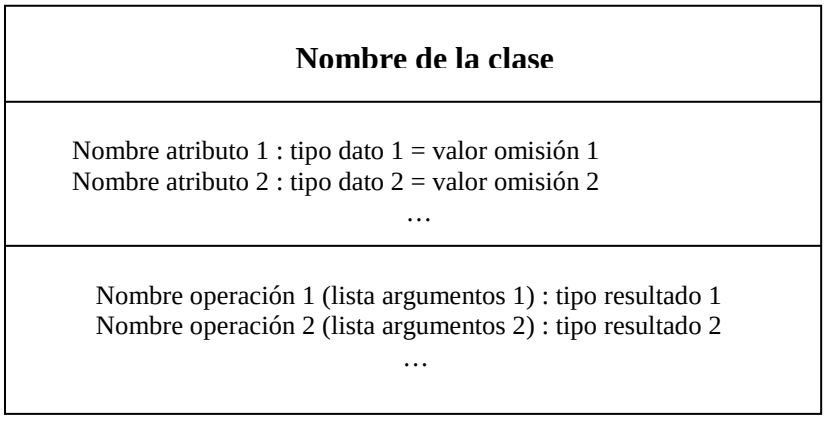


Gráfico 9

***Enlaces y asociaciones**:* Los enlaces y asociaciones son los medios que nos permiten establecer relaciones entre objetos.

. *Enlace:* es una conexión física o conceptual entre instancias de objetos, por ejemplo Juan Pérez trabaja para la compañía XYZ. Matemáticamente se define un enlace como una dupla, esto es, una lista ordenada de instancias de objetos. Un enlace es una instancia de una asociación.

. *Asociación:* describe un grupo de enlaces con estructura y semántica comunes, por ejemplo una persona trabaja para una compañía , todos los enlaces de cada asociación conectan objetos procedentes de las mismas clases.
Las asociaciones y los enlaces suelen aparecer como verbos en las definiciones de los problemas. Las asociaciones describen un conjunto de enlaces lo mismo que las clases describen un conjunto de objetos potenciales.

Las asociaciones son inherentemente bidireccionales, el nombre de una asociación binaria suele leerse en un cierto sentido, pero la asociación binaria se puede recorrer en ambas direcciones. La dirección implicada por el nombre es la dirección hacia delante, la dirección opuesta es la dirección hacia atrás, por ejemplo: *trabaja para* conecta a una persona a la compañía, lo inverso sería *da trabajo a* entonces conecta la compañía a la persona.
Las asociaciones suelen implementarse en los lenguajes de programación como punteros que van desde un objeto hasta otro. Un *puntero* es un atributo de un objeto que contiene una referencia explícita de otro objeto. La implementación de asociaciones en forma de punteros es perfectamente valido, pero las asociaciones, pero las asociaciones no deberían modelarse de esta forma.

Los enlaces muestran una relación entre uno o más objetos. El modelado de un enlace como un puntero oculta el hecho consistente en que el enlace no forma parte de ninguno de los objetos por si mismo, sino que depende de ambos a la vez.

Todas las conexiones entre clases deberían modelarse como asociaciones, incluso al modelar el programa, se insiste en que las asociaciones no son solamente construcciones de bases de datos, aun cuando las bases de datos relacionales se construyen basándose en el concepto de asociaciones.
Aún cuando las asociaciones se modelan como bidireccionales, no es necesario implementarlo en ambos sentidos. Ejemplos en el gráfico 10

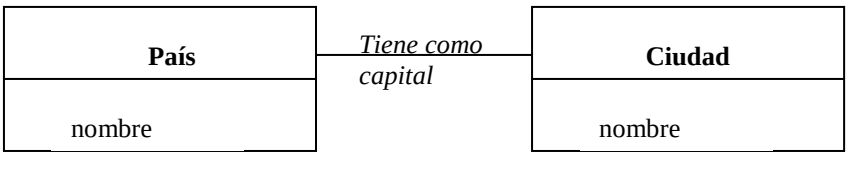


Gráfico 10

La notación OMT para las asociaciones es una línea entre clases, se trazan los enlaces como líneas entre objetos, los nombres de las asociaciones se ponen en cursiva, se puede omitir el nombre de la asociación si su significado es obvia.
Es bueno organizar las clases para que en lo posible se lean de izquierda a derecha.
Las asociaciones pueden ser binarias, terciarias o de orden superior, en la práctica, la inmensa mayoría son binarias, las terciarias y de orden superior son complejas de dibujar y entender.

***Multiplicidad**:* Especifica el número de instancias de una clase que pueden estar relacionadas con una única instancia de una clase asociada. La multiplicidad limita el número de objetos relacionados. Se suele describir la multiplicidad como "una" o "muchas", pero en general se trata de un subconjunto de los número enteros no negativos.
Por ejemplo, el número de puertas de un coche es 2 ó 4.
La multiplicidad se representa mediante símbolos especiales al final de las líneas de asociación, se pueden expresar mediante un número o mediante un conjunto de intervalos, tal como:
. 1 uno exactamente
. 1 + uno o más
. 3-5 entre tres y cinco, ambos inclusive
. 2,4,18 dos, cuatro, o bien dieciocho.
Existen ciertos terminadores de líneas especiales que indican valores frecuentes de multiplicidad:
. • círculo negro es el símbolo OMT que denota muchos
. O círculo en blanco denota opcional, es decir cero o uno.
. una línea sin símbolos de multiplicidad denota una asociación uno a uno.
La multiplicidad se escribe junto al final de la línea. Véase gráfico 11

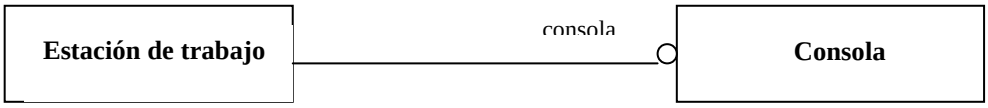


Gráfico 11

No hay que preocuparse excesivamente por la multiplicidad en las fases iniciales del desarrollo del software. Primero se determinan los objetos, clases y asociaciones, después se decidirá su multiplicidad.

La distinción de multiplicidad más importante es la existente entre uno y muchos, subestimar la multiplicidad puede limitar la flexibilidad de una aplicación.

***Importancia de las asociaciones**:* Las asociaciones se han utilizado ampliamente en el modelado de bases de datos desde hace varios años, sin embargo, aquí se hace hincapié en que las asociaciones son estructuras de modelado útiles para los programas, así como para los sistemas de bases de datos y del mundo real, independientemente de la forma en que estén implementados.

***Atributos de los enlaces**:* Un atributo es una propiedad de los objetos de una clase. De manera similar, un atributo de enlace es una propiedad de los enlaces de una asociación.

En el gráfico 12, *permiso de acceso* es un atributo de *puede acceder*.

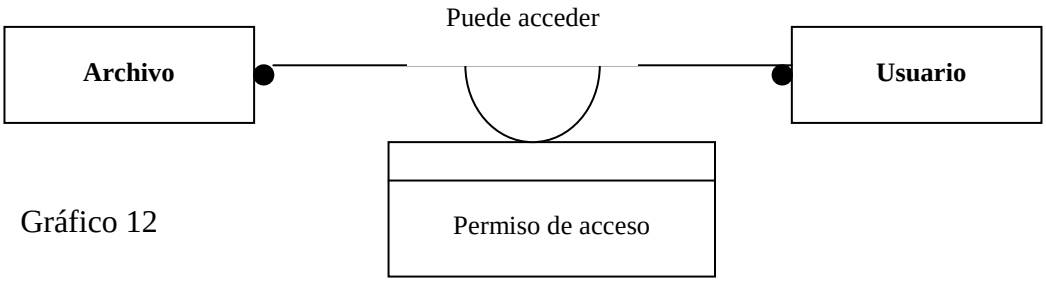


Gráfico 12

Archivo 1	(lectura)	Juan Pérez
Archivo 2	(lectura – escritura)	María García
Archivo 3	(Lectura – escritura)	Pedro León

La notación OMT para un atributo de enlace es un cuadro ligado a la asociación mediante un lazo, esta notación hace hincapié en la similitud existente entre los atributos de los objetos y los atributos de los enlaces.

Modelado de una asociación*:* (gráfico 13) En algunos casos resulta útil modelar la asociación como clases, cada enlace pasa a ser una instancia de la clase.
El cuadro de atributos puede poseer un nombre, unas operaciones y sus atributos.
El ejemplo siguiente muestra la información de autorizaciones para usuarios de estaciones de trabajo, los usuarios pueden estar autorizados a trabajar en muchas estaciones, cada autorización conlleva unos privilegios de prioridad de acceso, que se muestran en forma de atributos de enlace, cada usuario tiene un directorio origen para cada estación, pero un mismo directorio puede ser compartido por varios usuarios.

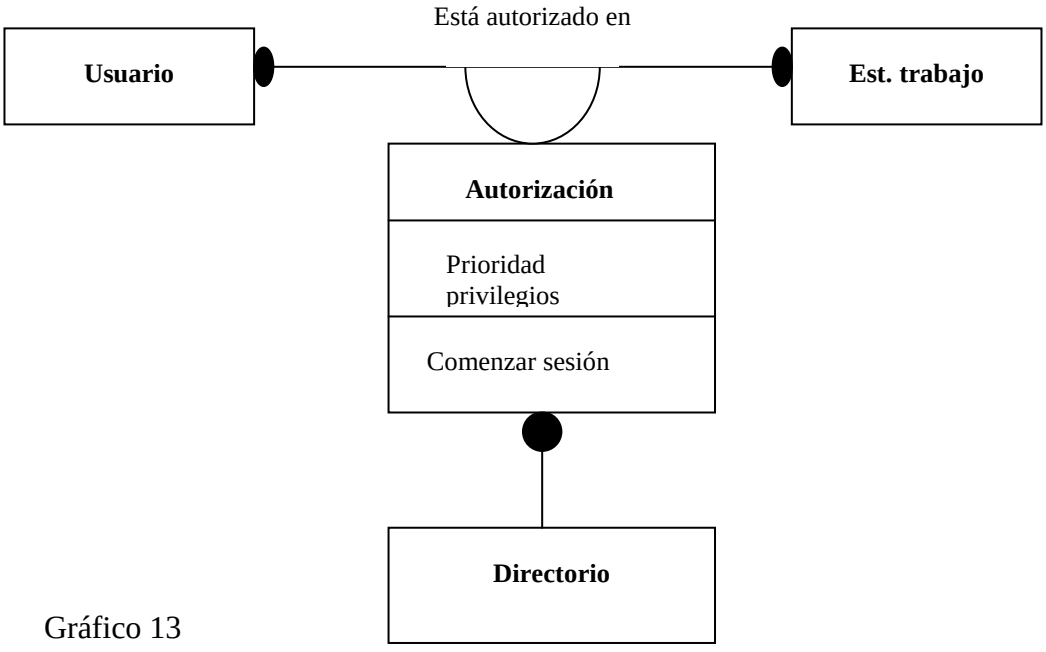
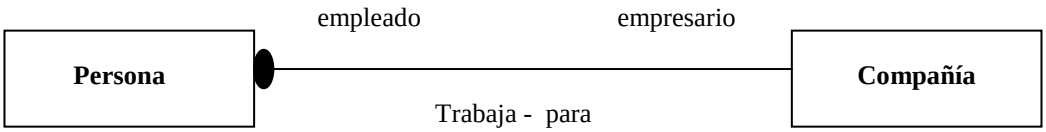


Gráfico 13

Es de utilidad modelar las asociaciones como clases cuando estos pueden participar en asociaciones con otros objetos, o cuando están sometidos a operaciones

***Nombre de Rol**:* Un rol es un extremo de una asociación, una asociación binaria posee dos roles, cada uno de los cuales puede poseer un nombre de rol.
Un nombre de rol es un nombre que identifica de forma única un extremo de una asociación. Los roles proporcionan una forma de visualizar las asociaciones binarias como un recorrido desde un objeto hasta un conjunto de objetos asociados.
El uso de nombres de rol proporciona una forma de recorrer las asociaciones desde un objeto de un extremo sin mencionar, explícitamente la asociación.
Los roles pueden aparecer como sustantivos en las descripciones del problema.
En el gráfico 14 se especifica la forma en que *persona y compañía* participan en la asociación *trabaja – para.* Una persona asume el rol de empleado con respecto a una compañía, una compañía asume el rol de empresario con respecto a la persona. El nombre del rol se escribe junto a la línea de asociación, al lado de la clase que desempeñe.



	Empleado	Empresario
	Juan Pérez	Simple
	María García	Simple

Gráfico 14

Los nombres de rol son necesarios para las asociaciones entre dos objetos de la misma clase.

***Clasificación**:* Generalmente, los objetos del lado “muchos” de una asociación no tienen un orden explícito, y se pueden considerar como un conjunto, sin embargo, en algunas ocasiones los objetos están ordenados explícitamente.
Por ejemplo, en el gráfico 15 muestra la pantalla de una estación de trabajo que contiene un cierto número de ventanas superpuestas, las mismas que están ordenadas explícitamente de la forma que la ventana más próxima al usuario es visible desde cualquier punto de la pantalla.
La clasificación es inherente a la asociación.
Un conjunto ordenado de objetos en el extremo “muchos” de la asociación, se indica escribiendo “{ordenado}”.

Gráfico 15

***Cualificación**:* La asociación cualificada o calificada relaciona dos clases de objetos y un cualificador, el mismo que es un atributo especial que reduce la multiplicidad efectiva de una asociación. Las asociaciones uno a muchos y muchos a muchos pueden ser cualificadas.

El cualificador distingue entre el conjunto de objetos que se encuentran en el externo de muchos de la asociación.
Por ejemplo, en el gráfico 16, un directorio posee muchos archivos, un archivo solo puede pertenecer a un directorio (en ambiente dos y windows), *directorio y archivo* son clases de objetos, y *nombre de archivo* es el cualificador.
La cualificación reduce la multiplicidad efectiva de esta asociación, pasando de uno a muchos, a uno a uno. Los directorios poseen muchos archivos, cada uno de los cuales tiene un nombre único.

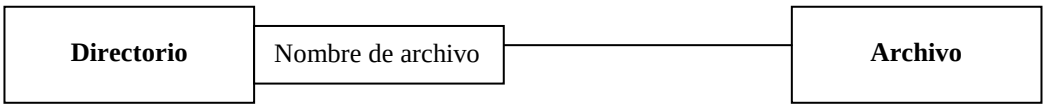


Gráfico 16

Un cualificador se dibuja en forma de un cuadrado pequeño en el extremo de la línea de asociación que se encuentra más próximo a la clase a la cual cualifica.
La cualificación mejora la precisión de la semántica e incrementa la visibilidad de las vías de navegación.

Agregación: Una agregación es la relación “parte todo” o “una parte de” en la cual los objetos que representan los componentes de algo, se asocian en un objeto que representa el ensamblaje completo.
Por ejemplo, un nombre, una lista de argumentos y una sentencia compuesta forman parte de la definición del lenguaje C, que a su vez forma parte de un programa completo.
La agregación es una forma fuertemente acoplada de asociación, una cierta cantidad de semántica adicional.
La propiedad más significativa de la agregación es la *transitividad,* esto es, si A es parte de B, y B es parte de C, entonces A es parte de C.
También es *antisimétrica*, esto es, si A es parte de B, entonces B no es parte de A.
Podemos definir la agregación como aquella que relaciona una clase ya ensamblada con una clase componente.
Un ensamblaje de muchas clases de componentes se corresponde con muchas relaciones de agregación.
La agregación es una forma especial de asociación.
Las agregaciones se representan igual a las asociaciones, salvo por un pequeño rombo que indica el extremo de ensamblaje.
En el gráfico 17 se representa un documento que consta de muchos párrafos, los mismos que constan de muchas frases.

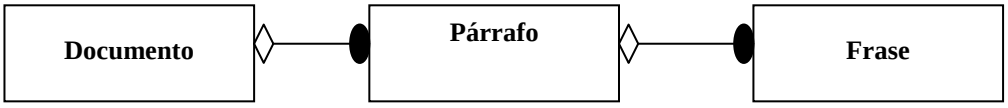


Gráfico 17

Generalización y herencia*:* Son potentes abstracciones para compartir similitudes entre clases al mismo tiempo que se mantienen sus diferencias.

La generalización es la relación entre una clase y una o más versiones refinadas de esa misma clase. La que se está refinando se denomina *superclase*, y cada versión refinada se denomina *subclase*. Los atributos y operaciones comunes a un grupo de subclase se asocian a las superclase y son compartidos por todas las subclases.
Se dice que cada subclase hereda las características de su superclase.
La generalización y la herencia son transitivas a través de un número arbitrario de niveles. Los términos ascendente y descendente hacen alusión a la generalización de las clases a través de múltiples niveles.
Una instancia de una subclase es simultáneamente una instancia de todas sus clases antecesoras.
En una subclase, toda operación aplicable a una clase antecesora, podrá ser aplicada a una instancia suya.
Las subclases heredan, no solamente todas las características de sus antecesoras sino que además añaden sus propios atributos y operaciones específicos.
La notación para la generalización es un triángulo que conecta la superclase con sus subclases, la superclase se conecta con una línea a la parte superior del triángulo, las subclases se conectan mediante una línea a una barra en la parte inferior del triángulo, de forma que la superclase queda en la parte superior, y las subclases en la parte inferior, así, gráfico 18:

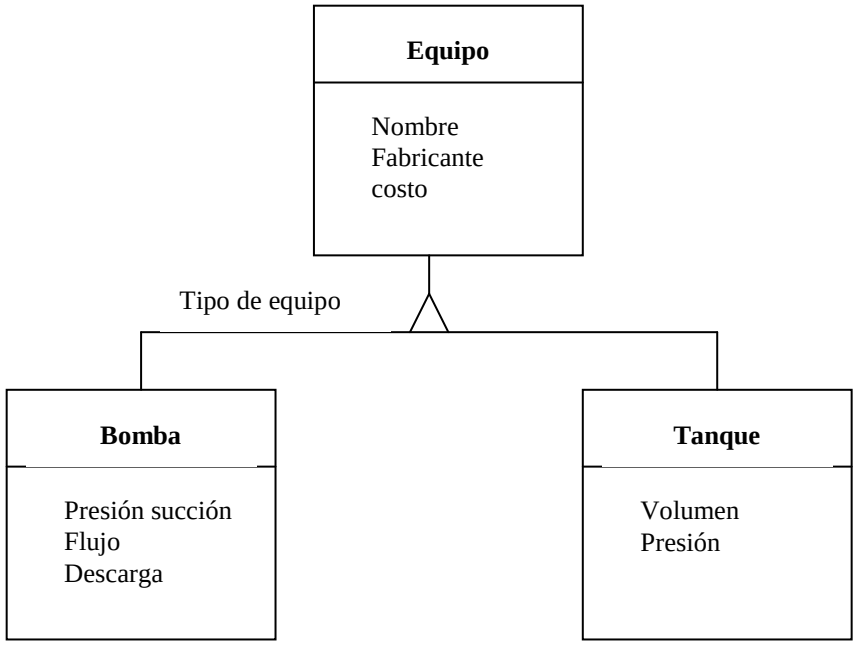


Gráfico 18

El texto que va al lado del triángulo es un discriminador, un atributo de enumeración que indica qué propiedad del objeto está siendo abstraída por una relación de generalización en particular.

La generalización es una construcción útil tanto para el modelado conceptual como para la implementación.
La generalización facilita el modelado estructurando las clases y capturando de forma concisa lo que es similar y lo que es distinto con respecto a las clases.

La herencia de operaciones resulta útil durante la implementación, como vehículo para la reutilización de código.

1.2.3.2 Modelo Dinámico

En principio, las relaciones dinámicas son difíciles de entender, lo más adecuado es examinar la estructura estática, después se examina los cambios de los objetos y sus relaciones con el tiempo, lo que constituye el modelo dinámico.
El *control* es aquella parte del sistema que describe las secuencias de operaciones que se producen en respuesta a estímulos externos, sin tener en cuenta lo que hagan las operaciones, aquello a lo que afectan ni la forma en que sean implementadas.
Describiremos los conceptos relacionados al flujo de control, con las interacciones y el secuenciado de operaciones en un sistema de objetos concurrentemente activos.
Los conceptos más importantes del modelo dinámico son los sucesos que representan estímulos externos, y los estados que representan los valores de los objetos.

Sucesos y estados*:* El modelado de objetos describe las posibles tramas de objetos, atributos y enlaces que pueden existir en un sistema.
Se denomina *estado* a los valores de los atributos y de los enlaces mantenidos por un objeto. A lo largo del tiempo, los objetos se estimulan unos a otros dando lugar a una serie de cambios a sus estados.
Un estímulo de un objeto que llega a otro se denomina un suceso, y la respuesta a este suceso depende del estado en que se encuentre el objeto que lo recibe.
Este comportamiento puede representarse en un diagrama de estados.

Sucesos*:* es algo que transcurre en un período de tiempo, ejemplo *el vuelo 022 sale para Guayaquil*. Los sucesos no tienen duración.
Un suceso puede proceder o seguir lógicamente a otro, a bien los dos sucesos pueden estar relacionados, en el ejemplo anterior, el vuelo 022 debe salir de Quito antes que pueda llegar a Guayaquil, entonces se dice que los sucesos están relacionados casualmente, caso contrario se considera sucesos concurrentes, es decir no se afectan entre sí.
Un suceso es una transmisión de información de dirección única entre dos objetos, un objeto que envía un suceso a otro objeto, puede esperar una respuesta, pero la respuesta es otro suceso distinto, bajo el control del segundo objeto, que puede decidir si lo envía o no.
Todo suceso aporta información de un objeto a otro, los valores de datos aportados por un suceso se llaman *atributos,* los mismos que se muestran entre paréntesis después del nombre de la clase de suceso. Ejemplos

Salida de vuelo (línea aérea, número de vuelo, destino) Pulsado botón del ratón (botón, situación) Teléfono descolgado Dígito marcado

Los atributos pueden o no especificarse.

***Escenarios y seguimiento de sucesos**:* El escenario es una secuencia de sucesos que se produce durante una ejecución concreta de un sistema, la figura 19 muestra la utilización de una línea telefónica, en donde cada objeto se muestra como una línea vertical, y cada suceso como una flecha horizontal que va desde el objeto emisor al receptor, el tiempo se representa secuencialmente de arriba hacia abajo.

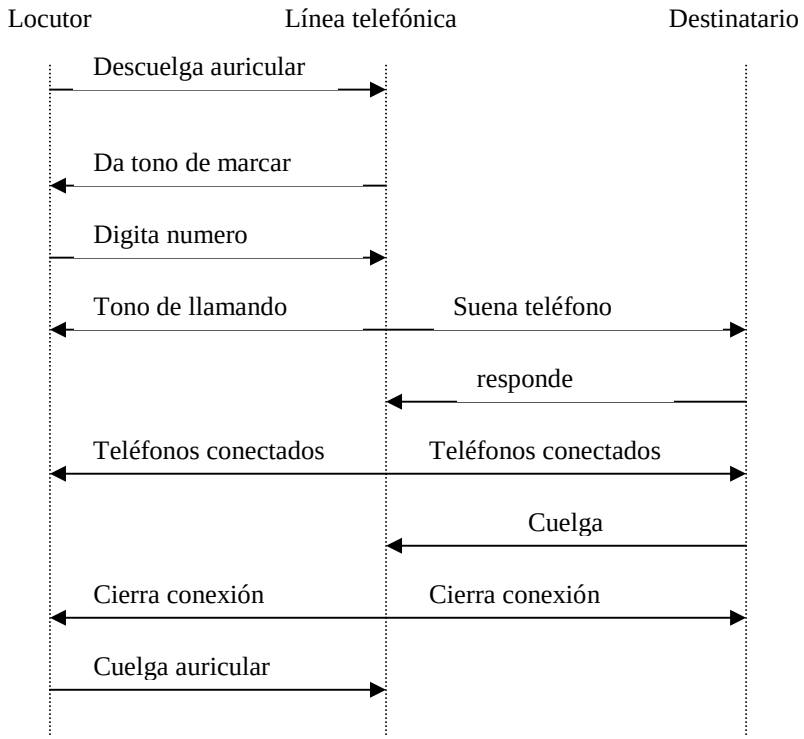


Gráfico 19

***Estados**:* Es una abstracción de los valores de los atributos y de los enlaces de un objeto, por ejemplo el estado de un banco es de solvencia o insolvencia, dependiendo de si su haber supera o no al debe. Un estado especifica la respuesta del objeto a los sucesos entrantes.

Un estado corresponde al intervalo entre dos sucesos recibidos por un objeto, los sucesos representan puntos temporales, los estados representan intervalos de tiempo.
Los estados tienen duración, ocupan un intervalo de tiempo. Los estados suelen asociarse con actividades continuas.
Los estados y los sucesos suelen ser duales entre sí, un suceso separa a dos estados, y un estado separa a dos sucesos.

***Diagrama de estado**:* Un diagrama de estados relaciona sucesos y estados, cuando se recibe un suceso, el estado siguiente depende del actual así como del suceso, un cambio de estado causado por un suceso es lo que se llama una transición.
Un diagrama de estados es un grafico cuyos nodos son estados, y cuyos arcos dirigidos son transiciones rotuladas con nombres de sucesos.
Los estados se representan como cuadros redondeados que contiene un nombre (opcional)

Las transiciones se representan con flechas desde el estado receptor al estado destino, el nombre de la flecha es el nombre del suceso que da lugar a la transición.
Todas las transiciones que salgan de un estado deben corresponder a sucesos distintos.
Ejemplo de una llamada en el gráfico 20:

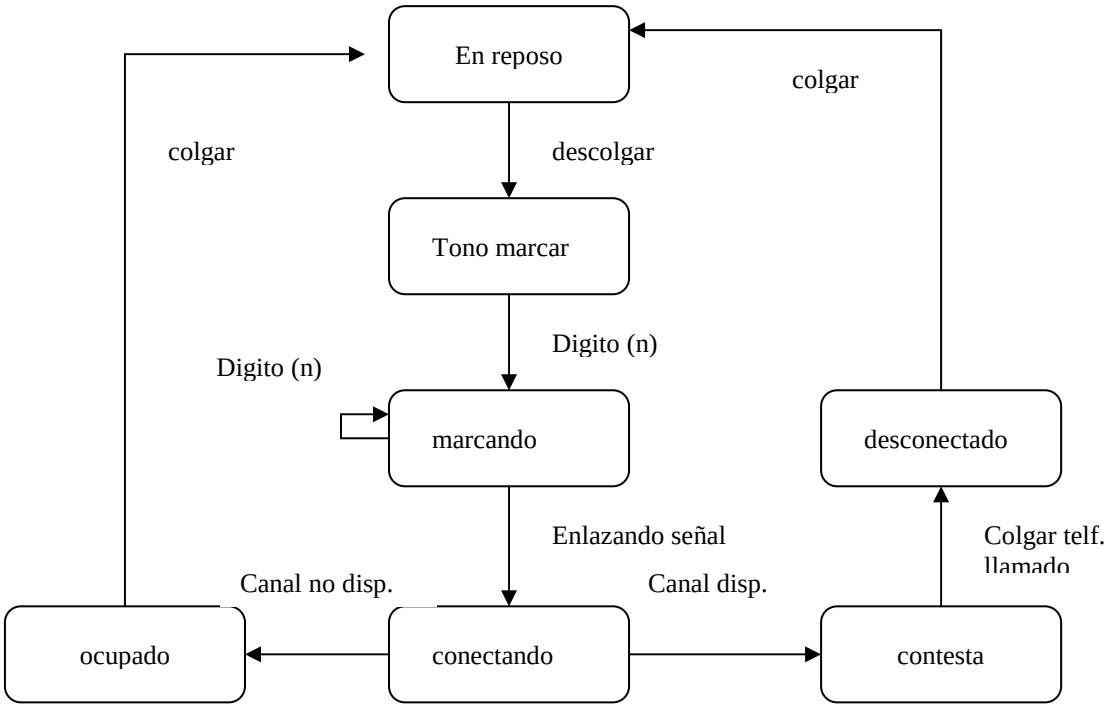


Gráfico 20

Los diagramas de estados pueden representar ciclos vitales únicos o bien bucles continuos.
Los diagramas de un solo uso representan objetos de duración finita y tienen estados iniciales y finales, se entra en el estado inicial al crear el objeto, al entrar al estado final estamos implicando la destrucción del objeto.
El estado inicial se ilustra mediante un círculo negro, y el final con un círculo blanco con el centro negro.

Ejemplo de un juego de ajedrez:

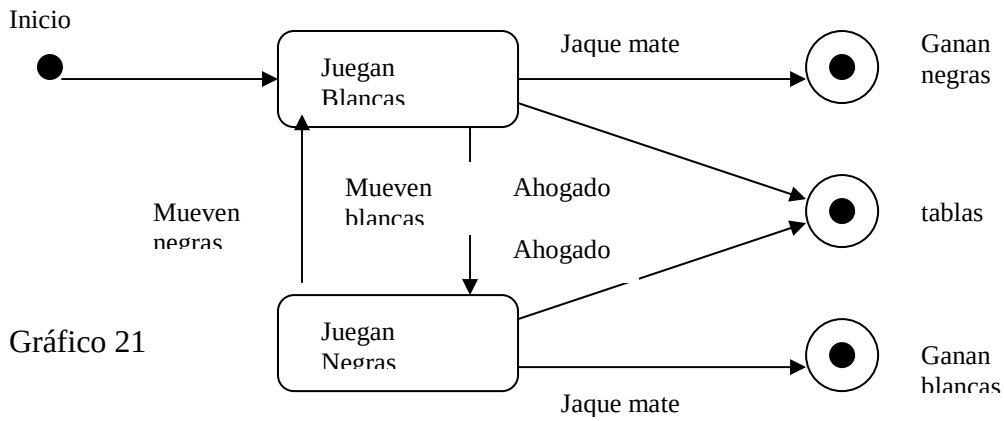


Gráfico 21

***Condiciones**:* Una condición es una función booleana lógica que tiene a objetos como valores, ejemplo: estuvo lloviendo desde las 2 PM hasta las 4 PM.
Es importante distinguir las condiciones de los sucesos que no tienen duración temporal.
Las condiciones se pueden usar como protecciones en las transiciones, una transición con protección se dispara cuando se produce su suceso, pero solo si la condición de protección es verdadera.
Las condiciones de protección se muestran entre corchetes como en el gráfico 22:

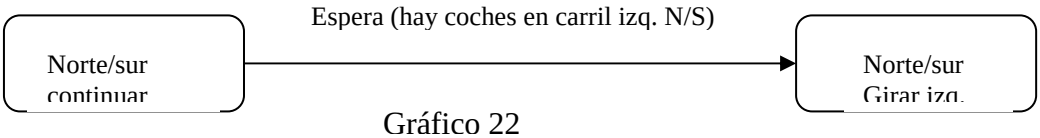


Gráfico 22

***Control de operaciones**:* Nos diagramas de secuencias también deben especificar lo que hace un objeto como respuesta a los sucesos. Las operaciones asociadas a estados o a transacciones se efectúan en respuesta a los correspondientes estados o sucesos.
Una *actividad* es una operación cuya realización requiere un cierto tiempo, toda actividad está asociada a un estado.
Una *acción* es una operación instantánea que va asociada a un suceso. Por ejemplo, desconectar la l línea de teléfono puede ser una acción como respuesta al suceso colgar.
La notación para una acción que afecta a una transición es una barra "/" y el nombre o descripción , después del nombre del suceso que lo produce.
La figura 23 muestra el diagrama de estado para un menú abatible en una estación de trabajo, cuando se pulsa el botón derecho se muestra el menú, cuando se suelta este botón, el menú se borra, mientras sea visible el menú, el ítem de cada opción se actualiza mientras se mueva el cursor.

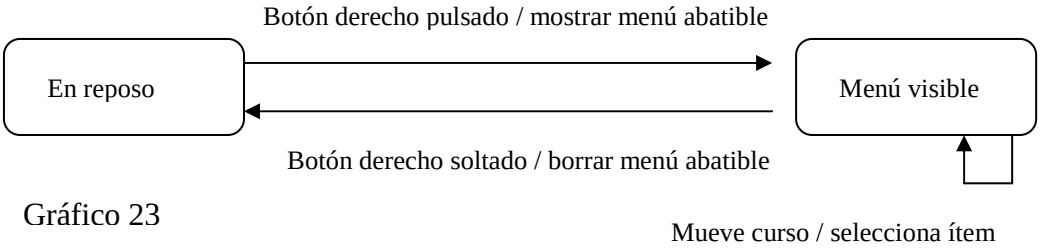


Gráfico 23

***Diagramas anidados**:* Este tipo de diagramas se pueden estructurar para hacer posible descripciones concisas de sistemas complejos.
La generalización es el equivalente a expandir las actividades anidadas, permiten describir una actividad empleando un nivel alto, y expandirla después a un nivel más bajo añadiendo detalles. Además la generalización permite que los estados y sucesos se dispongan en jerarquías de generalización con herencia de estructuras y comportamientos comunes, de forma similar a la herencia de atributos y de operaciones en las clases.

La agregación permite que el estado se descomponga en componentes ortogonales, con una interacción limitada entre ellos, de forma similar a una jerarquía de agregación de objetos.

***Anidamiento de diagramas de estados**:* Una actividad de estado se puede expandir en forma de diagrama de estado de nivel inferior, en el cual cada uno representará un paso de la actividad. Las actividades anidadas son diagramas de estado de un solo uso, con transiciones de entrada y salida parecidas a las subrutinas.

***Generalización de estados**:* Los diagramas de estados anidados equivalen a una forma de generalización de estados, un objeto que se encuentre en un estado del diagrama del alto nivel tiene que estar en uno de los estados del diagrama anidado.
Los estados del diagrama anidado son refinamientos del estado del diagrama de alto nivel.
Los estados del diagrama anidado no son afectados por las transiciones del diagrama de alto nivel.
Los estados pueden ser subestados que hereden las transiciones de sus super restados, a semejanza de las que heredan atributos y operaciones de sus superclases.
Todas las acciones aplicables a un estado, son aplicables a sus sub estados.

***Generalización de sucesos**:* Al igual que los estados, los sucesos también se pueden organizar en una jerarquía de generalización, de forma que hereden sus atributos.
Los atributos que se heredan de un suceso, se muestran en la segunda parte de cada cuadro, ejemplo en el gráfico 24:

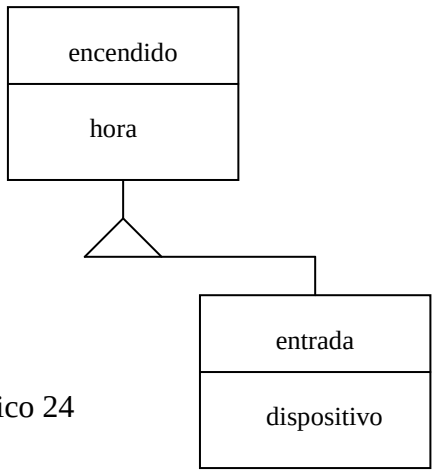


Gráfico 24

El suceso entrada, hereda el atributo hereda el atributo del suceso encendido.

1.2.3.3 Modelo Funcional

En el modelo funcional se describen los cálculos existentes dentro del sistema, muestra la forma en que se derivan los valores producidos en un cálculo a partir de los valores introducidos, sin considerar al orden en que se ejecutan dichas operaciones.

Consta de múltiples diagramas de flujos de datos, en las que se ilustra desde el ingreso de los datos, operaciones, almacenamientos internos y las salidas que se producen.

Los DFD (diagramas de flujos de datos) no muestran el control ni la información acerca de la estructura de los objetos, esto aparece en los modelos dinámico y de objetos.

Los modelos funcionales describen los resultados del cálculo sin detallar el cómo o cuándo los realiza.

Por ejemplo: una declaración de impuestos es una gran descripción funcional, especifica fórmulas para el cálculo basados en ingresos, gastos, donaciones, etc.. Un conjunto de formularios de impuestos y de instrucciones es un algoritmo que implementa el modelo funcional.

1.2.3.3.1 Diagrama de Flujo de Datos:

Un diagrama funcional consta de varios diagramas de flujos de datos (DFD), que determinan el significado de las operaciones y restricciones, muestra las relaciones funcionales entre valores calculados por un sistema, incluyendo valores introducidos, obtenidos y almacenados.

Los DFDs, muestran procesos que transforman datos, flujos de datos que los trasladan, objetos actores que producen y consumen datos y almacenes que los contienen en forma pasiva.

- ***Procesos**:* En principio, un proceso transforma valores en datos, los procesos de más bajo nivel son funciones puras, sin efectos laterales.
  Un gráfico completo de flujo de datos es un proceso de alto nivel, pueden tener efectos laterales si no contienen componentes no funcionales tales como almacenes de datos u objetos externos.
  Los procesos se ilustran en forma de elipses que contienen una descripción de la transformación, normalmente su nombre.
  Cada proceso tiene flechas de entrada y salida de datos, las mismas que llevan un valor de u tipo dado. Un proceso puede tener más de una salida.
  El esquema muestra el patrón de entradas, el cálculo de valores a partir de los datos de entrada, y las salidas.

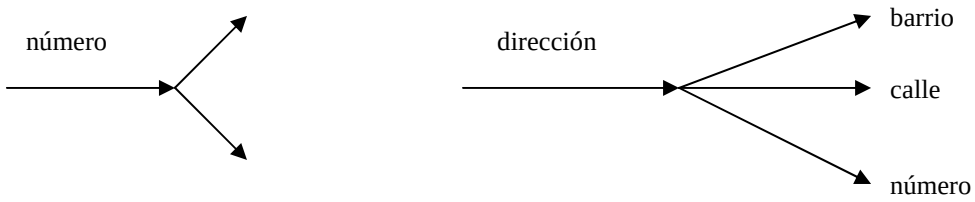
  Un proceso de alto nivel puede expandirse en todo un diagrama de flujo de datos de forma similar a la que se van expandiendo rutinas que llaman a subrutinas.

  Los procesos se implementan como fragmentos o sub métodos de operaciones que se aplican a clases de objetos.

- ***Flujo de datos**:* Conectan la salida de un objeto o proceso con la entrada de otro, representa un valor de datos intermedio dentro del cálculo que no es modificado por el flujo de datos, se dibujan como flechas entre el origen y el destino de ese valor de datos, y está rotulada con una descripción de los datos que lleva, generalmente su nombre o tipo.

  El mismo valor puede enviarse a varios destinos, esto se denota mediante la bifurcación con varias flechas que emergen de la original, en cuyo caso estas últimas no tienen rótulo, pues representan el mismo valor, como en el gráfico 25.

Gráfico 25



Como el segundo gráfico, algunas veces, un valor puede descomponerse en subvalores, cada uno de los cuales va a un proceso diferente, entonces cada flecha debe rotularse con el componente que lleva.

. ***Actores**:* Un actor es un objeto activo que controla el flujo de datos produciendo o consumiendo valores, los actores están asociados a las entradas y salidas del grafo y del flujo de datos, de cierta forma, los actores yacen en la frontera del grafo, pero hacen que concluya al flujo de datos como fuentes y sumideros de datos, por lo que en algunos casos de denominan terminadores.

Ejemplos de actores pueden ser un usuario de un sistema, un equipo, etc.

Las acciones de los actores no forman parte de los diagramas de flujo, éstos deben representarse en el modelo dinámico.

Los actores se representan como rectángulos, las flechas entre el actor y el diagrama son las entradas y salidas del diagrama. La “memoria intermedia” del gráfico del ejemplo que se ilustra más adelante, es un actor que consume operaciones de pixeles.

. ***Almacenes de datos**:* Un almacén de datos es un objeto pasivo dentro de un diagrama de flujo de datos, en el cual se almacenan datos para su posterior utilización.
Estos no generan operaciones, se limitan a responder a requerimientos de almacenamiento y acceso de datos.

Permiten además acceder a información en cierto orden de acuerdo al requerimiento, por ejemplo una base de datos de clientes lo podemos acceder por código, por nombre, por dirección, etc.
La forma de representarlos dentro del DFD es mediante un par de líneas paralelas horizontales entre las cuales se inserta el nombre del almacén. Las flechas de entrada significan operaciones que modifican los valores del objeto, las flechas de salida indican información que se ha extraído del mismo.

Las estructuras de los almacenes deben ser descritas en el modelo de objetos, junto son su descripción y operaciones que estén permitidas.

A continuación se ilustra un diagrama de flujo de datos (sistema gráfico de ventanas, gráfico 26) para comprender mejor el uso de los objetos antes mencionados:

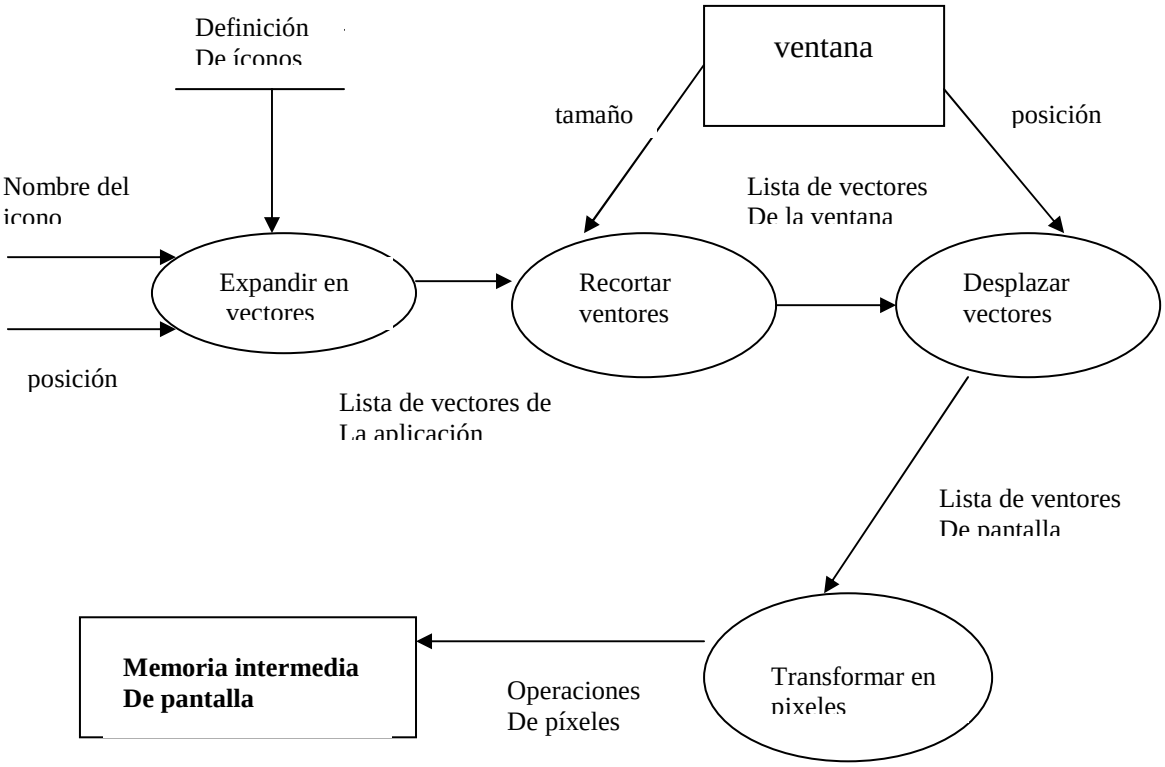


Gráfico 26

. ***Flujo de datos anidados****:* Resultan útiles para mostrar la funcionalidad de alto nivel del un sistema, un proceso se puede expandir en otro diagrama de flujo de datos.

Todas las entradas y salidas de un proceso lo serán también del nuevo diagrama, también pueden contener almacenes de datos que no consten en el nivel superior.

Los diagramas se pueden anidar hasta una profundidad arbitraria. A todo el conjunto de diagramas anidados se denomina árbol.

Un diagrama que hace referencia al mismo de forma repetida, representa un cálculo recursivo.

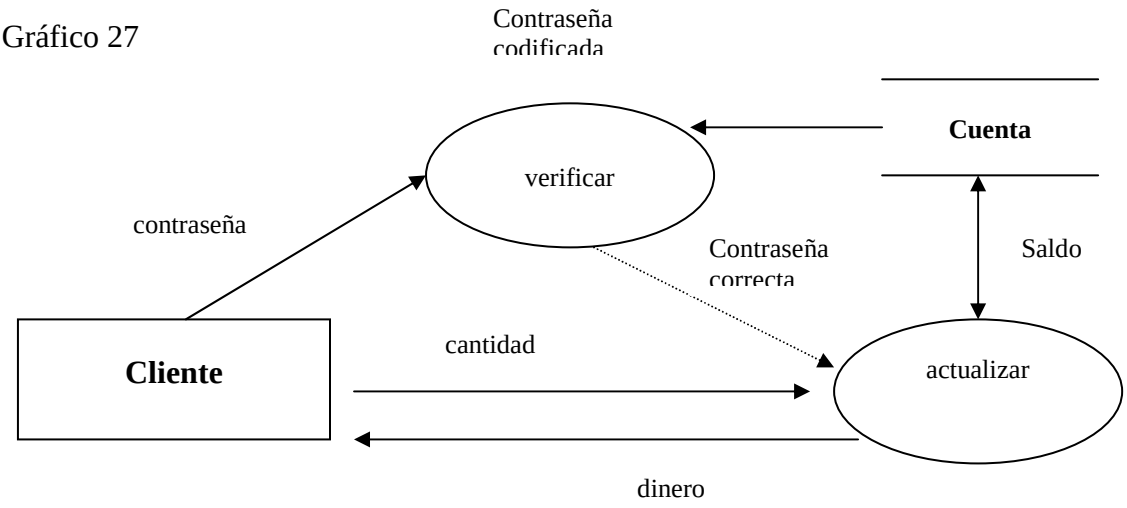
El anidamiento de diagramas también se denomina nivelado, haciendo referencia a que los diagramas se organizan en niveles.

Por lo regular, los niveles más bajos de los diagramas anidados concluyen con funciones sencillas.

. ***Flujos de control:*** Es un valor booleano que afecta si un proceso es o no evaluado. Se los representa mediante una línea punteada que va desde el proceso que produce el valor booleano hasta el que está controlado.

En el gráfico 27 se ilustra el proceso para un retiro de fondos en un cajero, en el cual se ejecuta la actividad “actualizar” siempre que se cumpla cierta condición en el proceso “verificar”.

Gráfico 27



. ***Especificación de operaciones****:* Eventualmente los diagramas de objetos deben contener operaciones que se aplican a los objetos, de hecho, todo proceso de más bajo nivel es una operación.

Una operación puede especificarse de diferentes maneras, así:
- funciones
- matemáticas
- tablas de valores
- tablas de decisión
- seudo código
- lenguaje natural.

Una especificación de operaciones incluye una signatura y una transformación.
La signatura define la interfaz de la operación: los argumentos y los valores que proporciona, especificando el número, orden y tipos.
La transformación define el efecto de una operación. Sus funciones y los efectos colaterales sobre los objetos tenidos como operandos.

El objetivo de las especificaciones de operaciones es indicar lo que debe hacer la operación y cómo deber ser implementada.

Las operaciones pueden catalogarse en tres categorías:
- Consultas: es una operación que carece de efectos laterales, no altera ni modifica objetos.
- Acciones: es una operación que causa efectos laterales sobre el objeto destino o sobre otros objetos del sistema que resulten alcanzables desde él. Se puede definir una acción del sistema antes y después de la acción.
  Las acciones se pueden definir de diferentes maneras, por ejemplo ecuaciones matemáticas, árboles, tablas de decisión.
- Actividad: se define como una operación hecha sobre un objeto que tiene una duración temporal, por lo cual tiene efectos colaterales.
  Las actividades solo tienen sentido para actores y objetos que tengan operaciones propias.

. ***Restricciones**:* Muestra la relación entre dos objetos al mismo tiempo o bien entre distintos valores del mismo objeto en instantes diferentes.
Las restricciones pueden aparecen en todas las clases del modelo, también especifican que algunos objetos dependen entera o parcialmente de otros objetos.
Las restricciones funcionales especifican limitaciones aplicables a operaciones.
Las restricciones aplicables a objetos a lo largo del tiempo es lo que se conoce como *invariantes,* las mismas que son de utilidad para especificar el comportamiento de las operaciones.

1.3 UML:

Representado por sus siglas en inglés, UML significa *Lenguaje Unificado de Modelamiento.*
Los lenguajes de modelado orientados a objetos, nacen hace aproximadamente veinte y veinticinco años, producto de las nuevas tendencias programación y desarrollo de sistemas que apuntaban a conceptos de orientación a objetos.

El número de métodos orientados a objetos llegó a más de cincuenta apara fines de 1994, destacándose entre ellos a los más importantes los siguientes:
- Booch
- OOSE (object oriented software engineering) de Jacobson
- OMT (Object Modeling Technique) de Rumbauh, visto en páginas anteriores.
- Fusion
- Shalaer - Mellor
- Coad - Yourdon

Todos estos poseían sus fortalezas y debilidades, dependiendo la etapa del desarrollo de sistemas a la que estemos enfocando.

Entonces nace UML como idea de unificar criterios entre tres de los principales gestores de los lenguajes de modelamiento, esto es Grady Booch, James Rumbauh & Ivar Jacobson.

Cuando comienza la unificación, se establecieron tres metas, así:

1. Modelar sistemas en todas sus fases, desde el concepto hasta los ejecutables utilizando técnicas orientada a objetos
2. Cubrir cuestiones de tamaño inherentes a sistemas complejos y críticos.
3. Crear un lenguaje de modelado que sea aplicable tanto por personas como por máquinas
4. Aportar con terminología de sintaxis y diagramación tendientes a establecer estandarización.

Fueron varias las firmas que contribuyeron en este esfuerzo de unificar técnicas de modelado, entre ellas podemos mencionar a: Digital Equipment Corporation, Hewelt Packard, I-Logix, IBM, entre otras.

UML consiste de un número de elementos gráficos combinados para formar diagramas con reglas para la combinación de los mismos.

El propósito de estos diagramas es presentar múltiples vistas de un sistema determinado, al que llamaremos *modelo,* el mismo que describe lo que realiza el sistema, *no indica cómo implementar el sistema.*

UML define los siguientes diagramas como parte del lenguaje de modelamiento: Clases, Objetos, Casos de uso, Estados, Secuencias, Actividad, Colaboración, Componentes y Despliegue.

Si bien no es indispensable la utilización de todos los diagramas en el desarrollo de un modelo, algunos de ellos sí son requisito indispensable, como veremos más adelante.

1.3.1 Diagrama de clases:

Como se había indicado ya, una clase es un grupo de cosas con atributos similares (representado por sus propiedades), y comportamientos comunes (representado por sus operaciones).

Un diagrama de clases muestra un conjunto de clases con sus respectivas interfaces, colaboraciones y relaciones.

Son los más utilizados en el modelado de sistemas orientada a objetos, muestra un conjunto de clases, interfaces y relaciones.

El diagrama de clases sirve como base a otros diagramas relacionados, estos son el de componentes y el de despliegue, que veremos más adelante.

Su representación gráfica es un rectángulo dividido en 3 partes, en la parte superior se indica el nombre de la clase, en la parte intermedia se especifican los atributos del objeto, y en la parte inferior se indican las operaciones a las que es susceptible el objeto.

En un trabajo de ingeniería inversa, debemos llegar a los diagramas de clases partiendo del código fuente y de las relaciones de estructuras ya establecidas y desarrolladas.

Desde el punto de vista del desarrollador, este diagrama provee las representaciones con las cuales debe trabajar, desde el punto de vista del analista, provee de diagramas entendibles por el usuario, facilitando de esta forma las tareas de comprensión del tema a estudiar.

Los diagramas de clases contienen los siguientes elementos: clases, interfaces, colaboraciones y relaciones de dependencia.

En el gráfico 28 se nuestra un ejemplo de un diagrama de clases de una empresa:

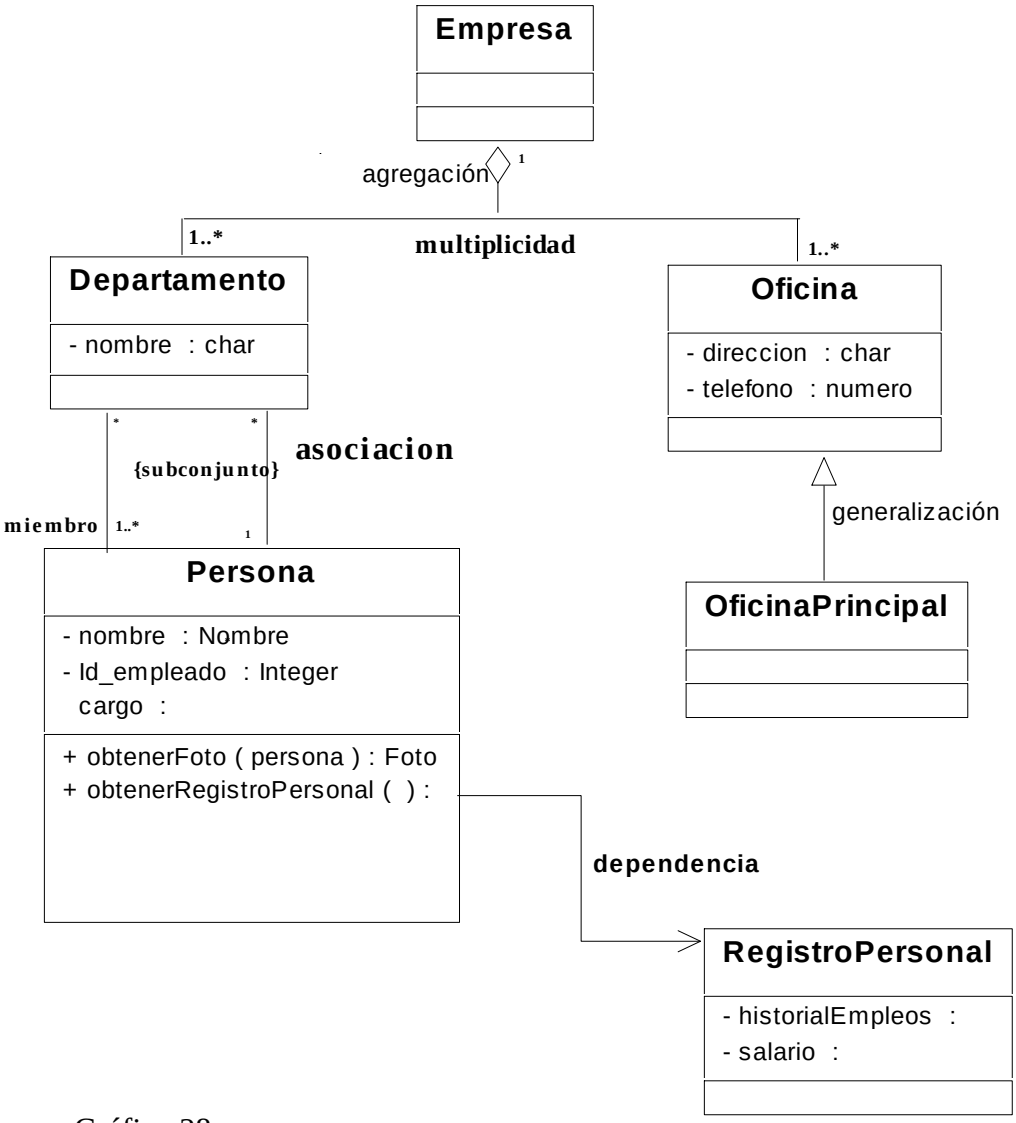


Gráfico 28

1.3.2 Diagrama de Objetos:

En esencia representan una instancia del diagrama de clases, ver gráfico 29

Estos diagramas modelan las instancias de los elementos contenidos en los diagramas de clases, y muestran un conjunto de objetos y sus relaciones en un momento dado.

Se utilizan para mostrar la vista estática de diseño y procesos de un sistema, y su representación gráfica se la hace en un icono rectangular, el nombre está subrayado. El nombre de la instancia se encuentra en la parte izquierda y el nombre de la clase en la parte derecha. Así:

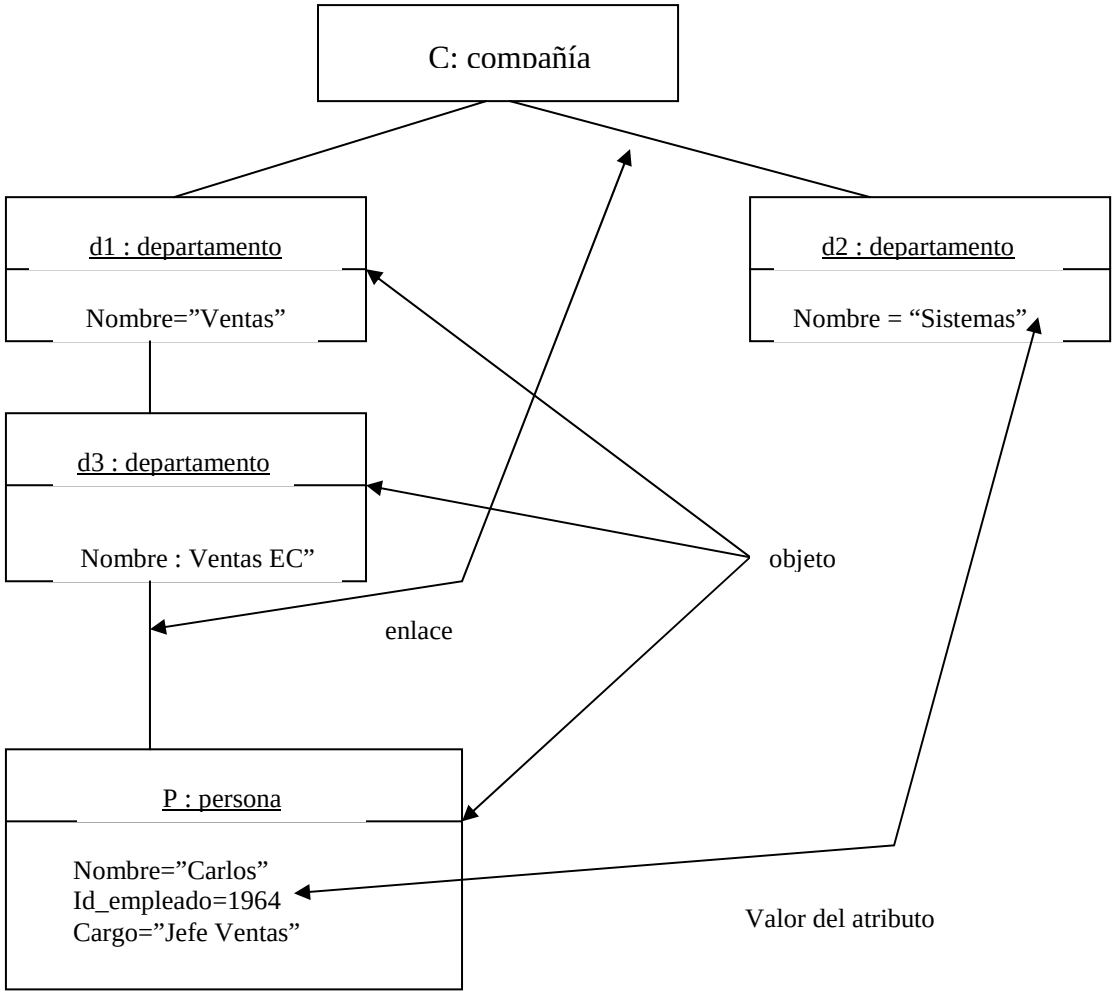


Gráfico 29

1.3.3 Diagrama de Casos de uso

. *Casos de uso:* Especifica el comportamiento de un sistema, o parte de un sistema, y es una descripción de un conjunto de secuencias de acciones (incluyendo variantes o restricciones), que ejecuta un sistema para producir un resultado.

Se emplean para capturar el comportamiento del sistema sin tener que especificar cómo se implemente dicho comportamiento.

Los casos de uso denotan comportamientos esenciales del sistema y de sus módulos, por lo cual no deben ser ni excesivamente genéricos, ni muy detallados tampoco, pueden aplicarse al sistema completo, parte del sistema, incluso hasta clases y/o interfaces individuales.

Un caso de uso involucra la interacción de actores con el sistema, los actores pueden ser personas o sistemas mecánicos, especifica qué hace un sistema, no como lo hace.

UML provee de una representación gráfica mediante un actor y un proceso como se ilustra en el gráfico 30:

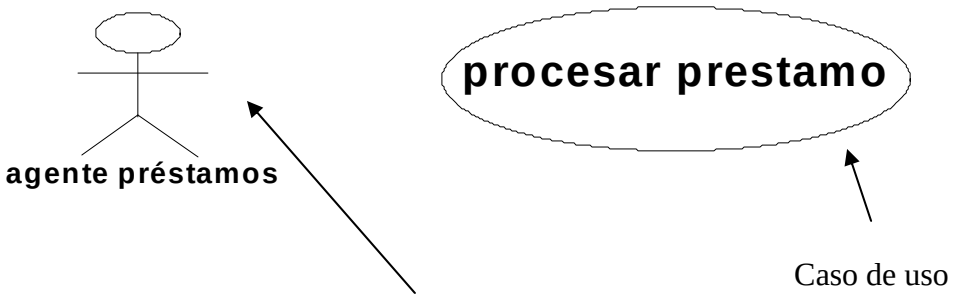


Gráfico 30

Cada caso de uso debe tener un nombre que lo distinga de otros.

. *Diagrama de casos de uso*: Estos diagramas son uno de los cinco que junto con los diagramas de actividad, estados, de secuencia y colaboración que utiliza UML para el modelado de los aspectos dinámicos de un sistema.

Son importantes para la visualización, especificación y documentación de un elemento, son importantes para probar sistemas a través de la ingeniería directa, y para comprender sistemas ejecutables a través de la ingeniería inversa.

Los casos de uso se definen como un diagrama que muestra un conjunto de casos de uso, actores y sus relaciones.

Generalmente un diagrama de casos de uso contiene los siguientes elementos:
- Casos de uso
- Actores
- Relaciones de dependencia, generalización y asociación.

Uno de los usos de estos diagramas es para modelar los requisitos de un sistema, en cuyo caso implica qué deberá hacer el sistema desde un punto de vista externo, independientemente de cómo lo haga, se emplean los diagramas para especificar el comportamiento deseado del sistema.
Para modelar el contexto de un sistema se debe considerar:

- Identificar los actores en torno al sistema, considerando cuáles requieren ayuda del sistema para completar sus tareas, quienes interactúan con el hardware y software, y quiénes realizan funciones secundarias.
- Organizar los actores similares en jerarquías de generalización y especialización.
- Introducir estos actores en diagramas de casos de uso y especificar las vías de comunicación.

Para modelar los requisitos de un sistema debemos considerar lo siguiente:
- Establecer el contexto del sistema especificando sus actores
- Considerar el comportamiento de cada actor del sistema.
- Nombrar los comportamientos comunes como casos de uso.
- Modelar los casos de uso, actores y relaciones en diagramas.
- Añadir comentarios a los diagramas.

Los dos siguientes diagramas 31 y 32 muestran el modelado de contexto de un sistema de validación de tarjetas de crédito, y el modelado de requisitos del mismo sistema, respectivamente.

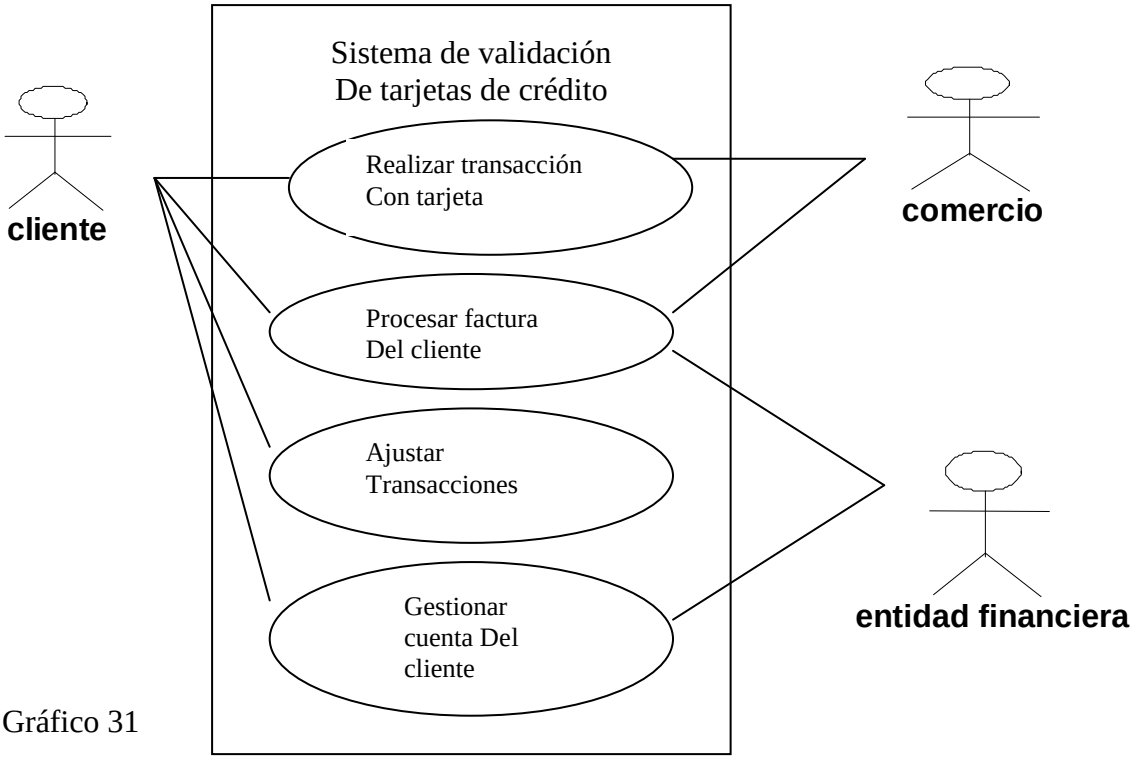


Gráfico 31

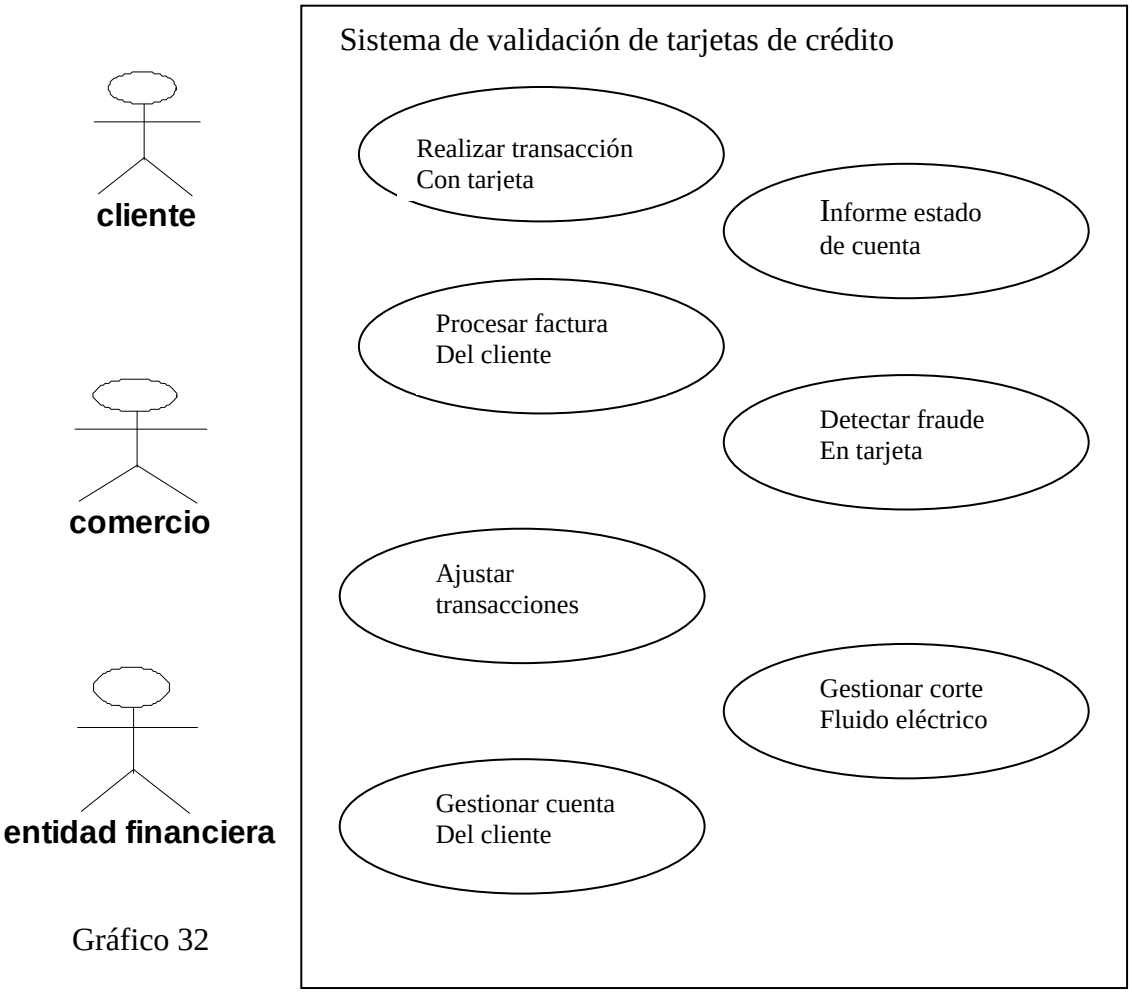


Gráfico 32

Para aplicar ingeniería inversa se puede estudiar un sistema existente y discernir su comportamiento en forma manual, para ponerlo después en forma de diagramas de casos de uso, en similitud a lo que regularmente se hace cuando se toma un sistema sin documentación, UML proporciona un lenguaje estándar para diagramar lo descubierto.

Para llegar a diagramas de casos de uso mediante ingeniería inversa:
- Identificar a cada actor que interactúa con el sistema.
- Determinar la forma en que cada actor interactúa con el sistema
- Trazar un flujo de eventos del sistema a cada actor, empezando por flujos principales, posteriormente los caminos alternativos.
- Agrupar flujos relacionados declarando el correspondiente caso de uso.
- Representar los actores y casos de uso en un diagrama de casos de uso, y establecer relaciones.

1.3.4 Diagrama de Estados

Son utilizados para modelar aspectos dinámicos del sistema, la mayoría de veces, esto supone el modelado del comportamiento de objetos reactivos. Un objeto reactivo es aquel para el que la mejor forma de caracterizar su comportamiento es señalar cuál es su respuesta a los eventos enviados desde fuera de su contexto.

Los diagramas de estado pueden asociarse a las clases, casos de uso o a sistemas completos para visualizar, especificar, construir y documentar la dinámica de un objeto determinado.

La representación UML para un estado es un rectángulo redondeado, el símbolo para una transición es una flecha, el círculo denota el inicio del estado, y un círculo “ojo de buey” indica la finalización del estado, como muestra el gráfico 33:

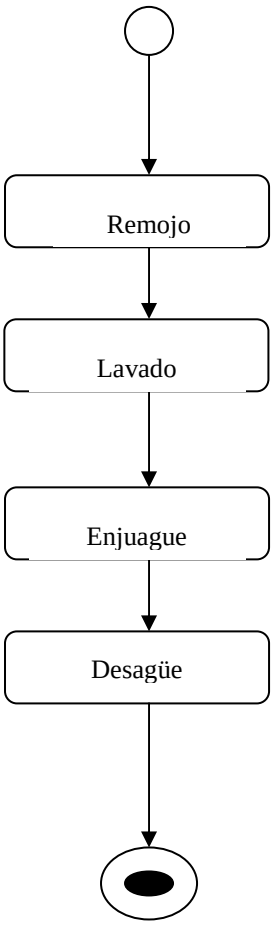


Gráfico 33

El gráfico 33 muestra por posibles estados de una máquina de lavar.

La ingeniería inversa es teóricamente posible, pero no es muy útil en la práctica, la elección de un estado significativo depende del diseñador, las herramientas de ingeniería inversa no tienen capacidad de abstracción, por lo cual no son capaces de generar diagramas de estado significativo.

1.3.5 Diagrama de Interacción:

Llamamos diagramas de interacción a los diagramas de secuencia y a los diagramas de colaboración, y son usados también para modelar los aspectos dinámicos del sistema.

Consisten en una serie de objetos y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellos.

Estos diagramas son importantes también para construir sistemas ejecutables por medio de ingeniería directa e ingeniería inversa.

. ***Diagramas de Secuencia****:* Estos diagramas destacan la ordenación temporal de los mensajes, como se muestra en dibujo siguiente.
Se forma colocando en la parte superior los objetos que participan en la interacción, por lo regular a la izquierda el objeto que inicia la interacción y los objetos subordinados hacia la derecha. Se colocan a continuación los mensajes que se envían y reciben a lo largo del eje Y en orden de sucesión en el tiempo desde arriba hacia abajo.
El gráfico 34 ilustra un diagrama de secuencias de una máquina dispensadora de sodas en su más simple comportamiento:

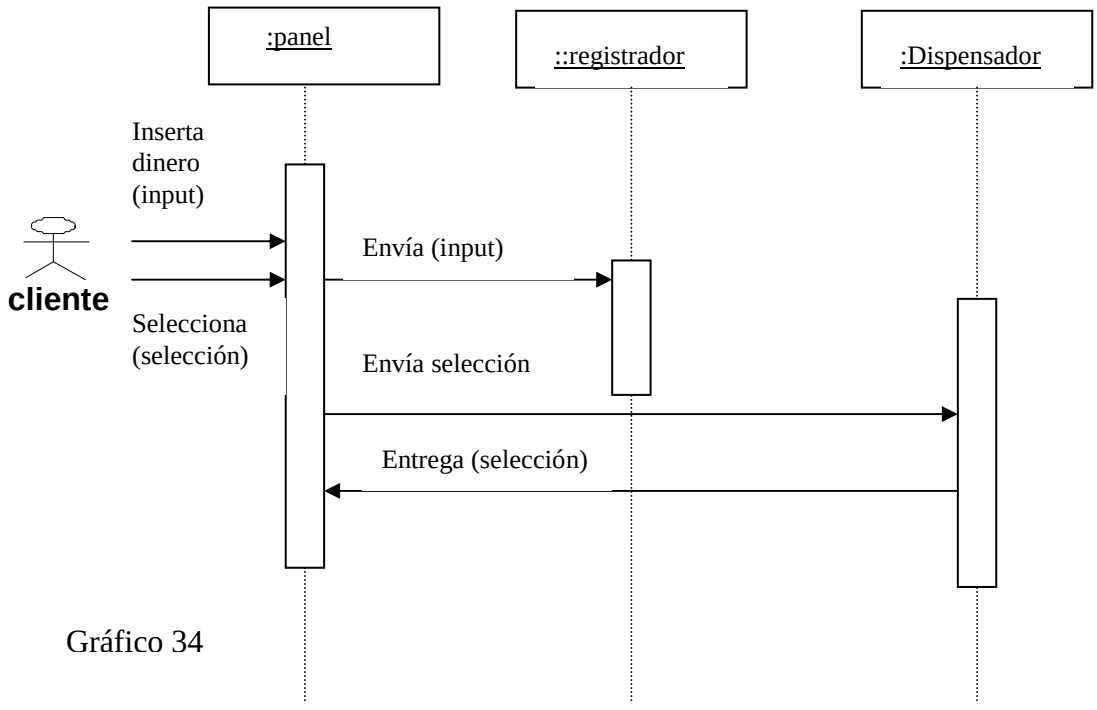


Gráfico 34

. ***Diagramas de Colaboración****:* estos diagramas destacan la organización de los objetos que participan en la interacción, se lo construye colocando los objetos que intervienen como nodos del gráfico, luego se representan los enlaces que conectan los objetos antes mencionados, por último en las relaciones se añaden los mensajes que reciben y envían los objetos.

Los dos tipos de diagramas (secuencia y de colaboración) son similares, de hecho son semánticamente equivalentes, y se puede pasar de uno a otro indistintamente.

La organización del diagrama de secuencias es sobre la base del tiempo, mientras el de colaboración es de acuerdo al espacio.

Un diagrama de objeto muestra los objetos y sus relaciones con otros, visto así, el diagrama de colaboración es una extensión del diagrama de objetos.

A continuación ilustramos el gráfico 35, del diagrama de secuencias de la máquina dispensadora de sodas:

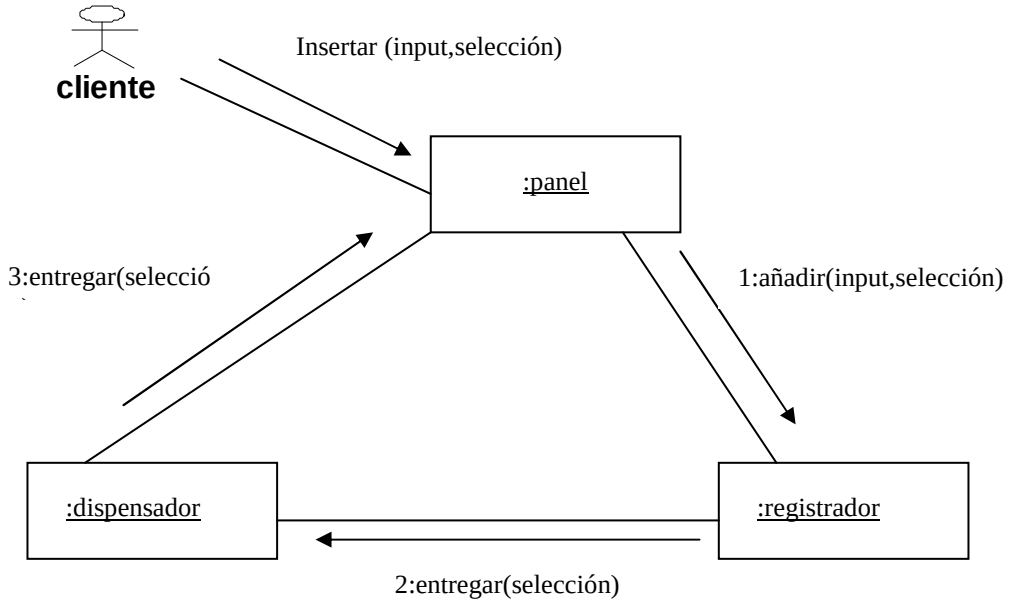


Gráfico 35

1.3.6 Diagrama de Actividad

Es otro de los diagramas utilizados para modelar los aspectos dinámicos de un sistema, donde se muestra fundamentalmente el flujo de control entre actividades.

Implica modelar pasos secuenciales y eventualmente concurrentes de un proceso computacional, pudiendo modelar también el flujo de un objeto conforme pasa de estado a estado en diferentes puntos del flujo de control.

Un diagrama de actividad contiene generalmente:

. *Estados de Actividad:* Es la representación de actividades como avaluación de expresiones, invocación de operaciones, creación y eliminación de objetos temporales, y se representan con un rectángulo con los bordes redondeados.

. *Transiciones:* Especifican el flujo o camino de un estado de actividad o de acción al siguiente, y se lo representa con una línea dirigida.

. *Bifurcación:* Son útiles para modelar ciertos flujos de control, en que pueden seguirse ciertos caminos basados en condiciones específicas. Se representa con un rombo del cual se desprenden flujos distintos según el resultado de una evaluación booleana.

División y unión: Utilizados para modelar flujos de control paralelos. Una barra de sincronización se representa con una línea horizontal ancha. El inicio y fin de procesos paralelos se marcan con una barra horizontal.

Calles: Son especialmente útiles para modelar flujos de trabajo en organizaciones, donde cada sección o calle representa un grupo de la organización. Cada calle tiene un nombre único dentro del diagrama.

A continuación gráfico 36 que ilustra los conceptos antes anotados:

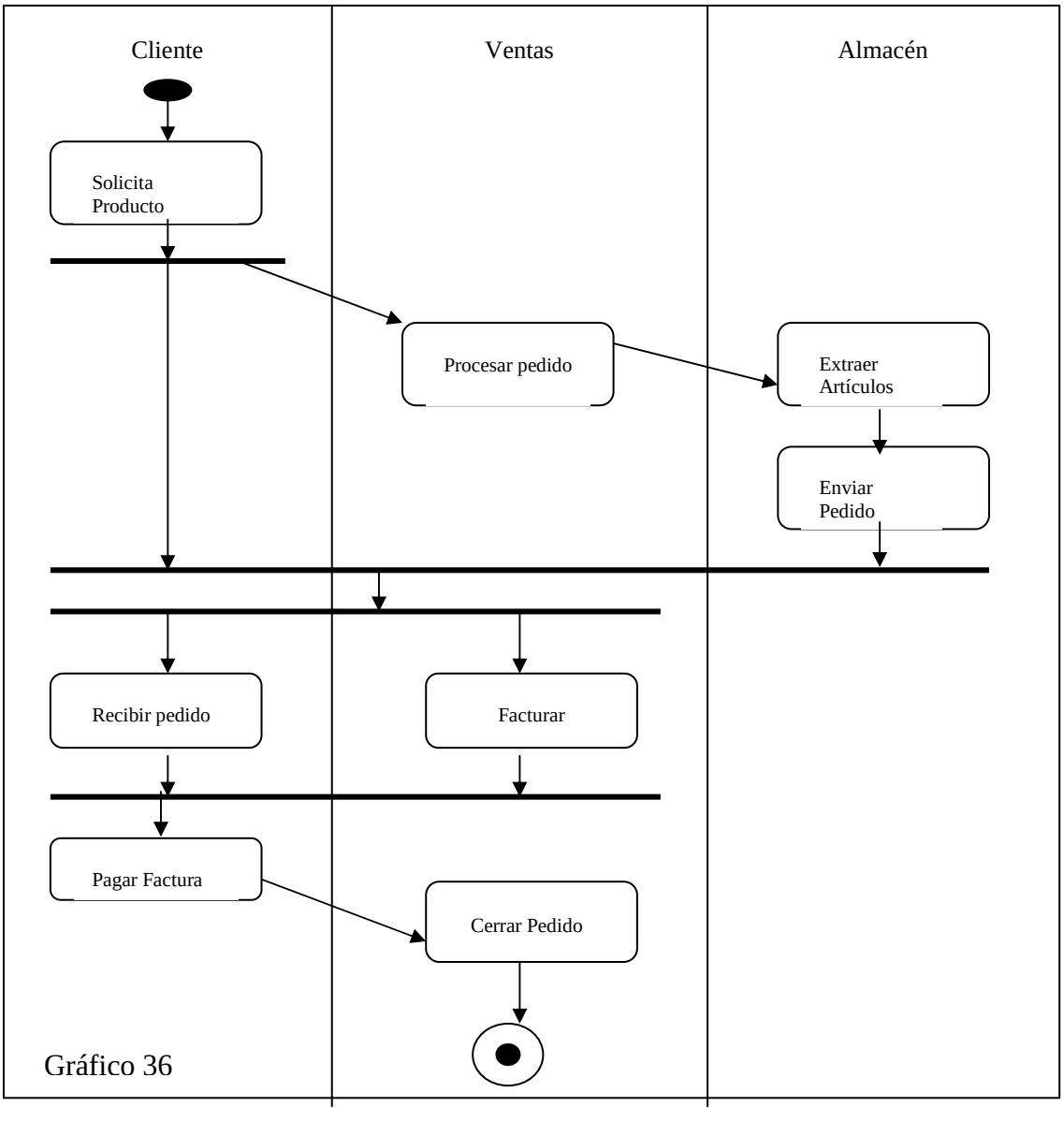


Gráfico 36

1.3.7 Diagrama de Componentes

Este tipo de diagramas tienen aplicación cuando se modelan aspectos físicos de sistemas orientadao a objetos, mostrando la relación y dependencia entre un conjunto de componentes.

Implica el modelado de cosas físicas que residen en un nodo, tales como ejecutables, bibliotecas, tablas, archivos y documentos, gráficamente es una colección de nodos y arcos, pueden contener además notas y restricciones.

. *Para modelar código fuente:* se pueden utilizar para modelar la gestión de configuración de estos archivos, los cuales representan los componentes obtenidos como producto del trabajo.
. *Para modelar versiones ejecutables:* Una versión es una aplicación relativamente consistente y completa que se entrega a un usuario, en términos de componentes, una versión representa las partes necesarias para entregar un sistema en ejecución.
. *Para modelar bases de datos:* Representa al almacenamiento de información en tablas de una base de datos física relacional u orientada a objetos.

Ejemplo de un diagrama de componentes de un procesador de textos, en el gráfico 37:

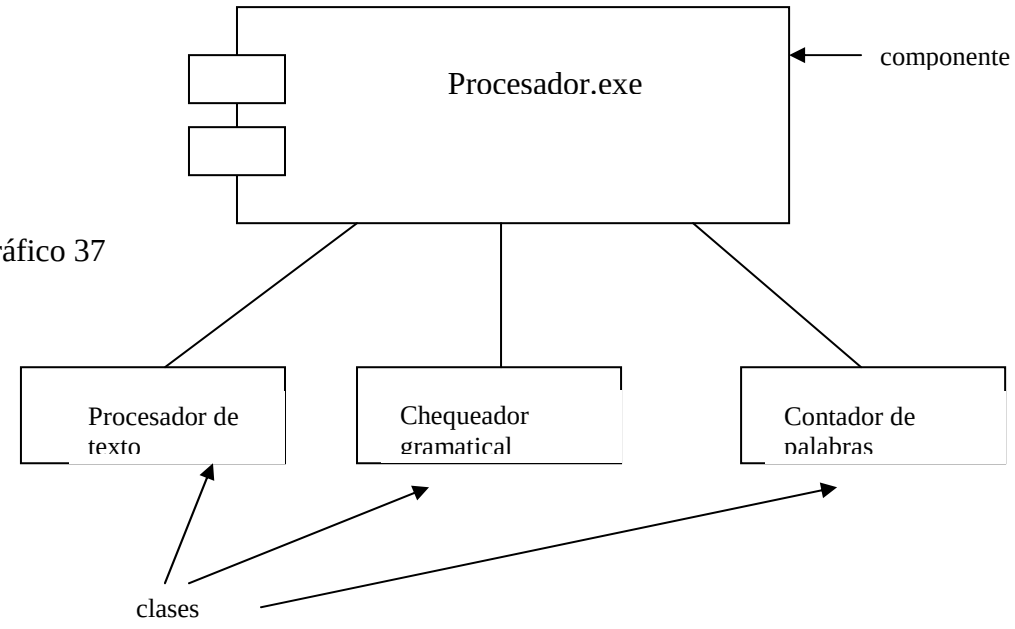


Gráfico 37

1.3.8 Diagrama de Despliegue

Usados cuando se modelan aspectos físicos de sistemas orientada a objetos, muestra la configuración de nodos que participan en la ejecución y de los componentes que residen en dichos nodos.

Son importantes para visualizar, especificar y documentar sistemas empotrados, cliente servidor, y sistemas distribuidos.

En la mayoría de las veces, implica modelar el hardware sobre el cual están implementadas las aplicaciones, por lo que UML contempla también el diseño de elementos de hardware.

Regularmente un diagrama de despliegue contiene:
. Nodos
. Relaciones de dependencia y asociación.

UML representa los aspectos estáticos de los nodos y sus relaciones, como se muestra en el gráfico 38:

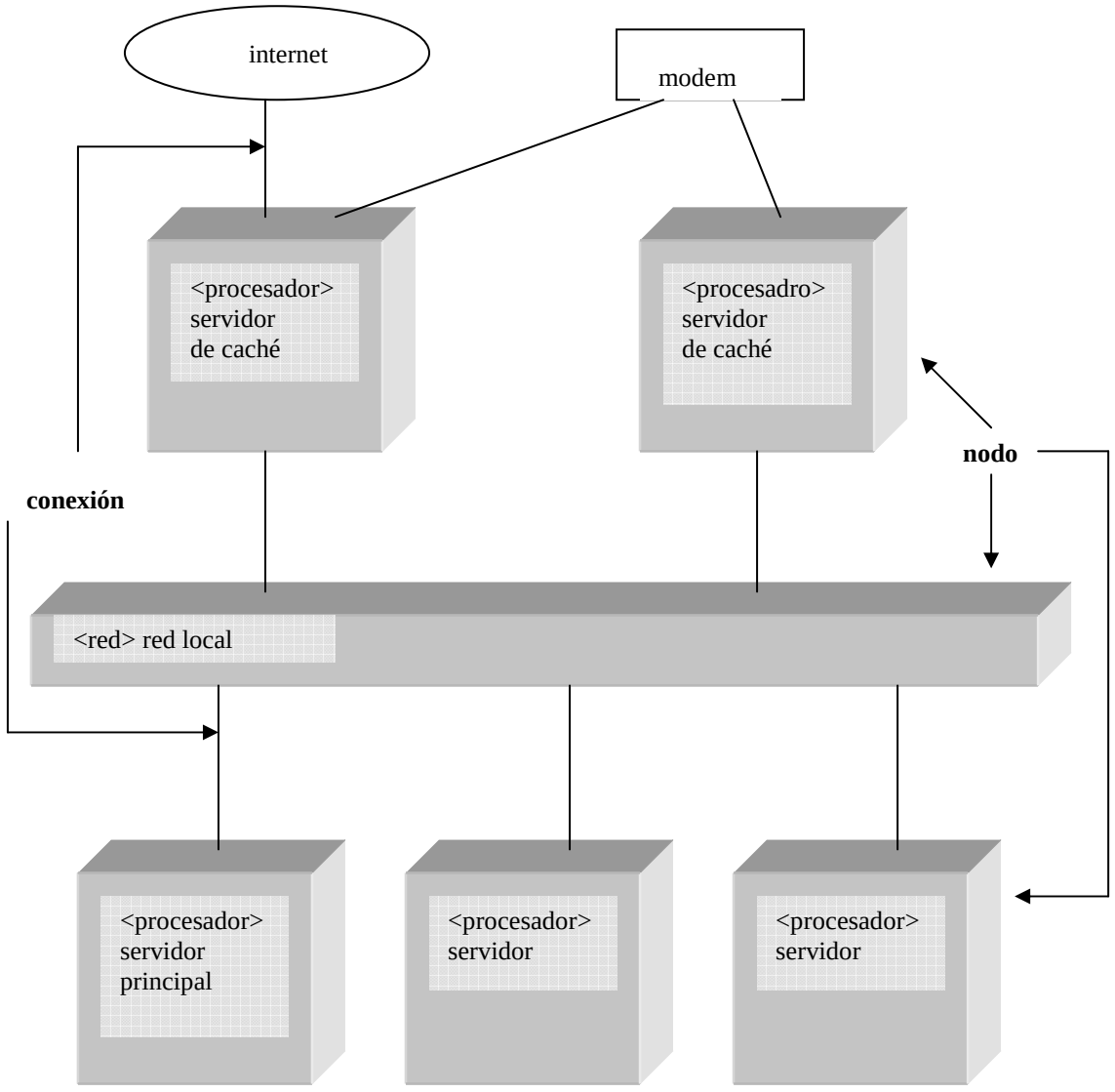


Gráfico 38

Capítulo 2 Etapas de la Reingeniería

***Antecedentes**:*
Este capítulo pretende orientar en las actividades que pueden aplicarse a un proceso de reingeniería de software, bajo la óptica de los diferentes componentes que integran un sistema automático de datos, esto es, programas, bases de datos, documentación, y hardware, los mismos que deberán aplicarse según las circunstancias que rodean a cada caso específico.

Es de entenderse que cada caso es particular, por lo que su resolución puede requerir la aplicación de una o varias de los pasos de reingeniería con un nivel de abstracción diferente.

A continuación se definen las etapas de un proceso de reingeniería:

2.1 Análisis del Inventario:

Nombre de la Aplicación..... ……. EVACOM (evaluación de competitividad)
Año de Creación..... jun-1988
Número de Cambios Importantes..... 4
Esfuerzo empleado en cambios ..... 50 horas
Fecha del último cambio nov-1988
Base de datos a las que accede ..... propias de EVACOM (sql-anywhere)
Número de usuarios ..... Monousuario
Número de maquinas instaladas 1
Complejidad de Arquitectura..... Simple
Código ..... Compleja
Documentación ..... Simple
Calidad Documentación ..... No existe.
Longevidad del proyecto ..... 4 años (estabilidad de la economía)
Número de cambios aprox. 36 meses. No estimable
Importancia para el negocio ..... Alta

2.2 Reestructuración de Documentos:

En vista que el sistema actualmente no dispone de ningún tipo de documentación, es parte de las necesidades del usuario contar con documentación completa de manuales técnicos y de usuario.

Estos manuales se adjuntan a este documento.

2.3 Ingeniería Inversa.

A continuación se presenta un cuadro en el cual se especifica la funcionalidad del sistema, con sus respectivos módulos y opciones actuales, el mismo que servirá de base a los nuevos requerimientos del usuario.

Adicionalmente se adjunta en los anexos 1 y 2, los respectivos diagramas de entidad relación, y los diagramas de flujo de datos (DFDs) que representan la estructura actual de la aplicación.

Módulo	Opción	Sub opción	Descripción
Códigos			Permite parametrizar el sistema
	Períodos		Define períodos de tiempo para producción, cierre…
	Ramas CIIU		Definición internacional de ramas CIIU
	Unidades		Define unidades de medida
	Compañías		Creación de compañías a ser evaluadas
	Temas Propuestos		Temas propuestos por el consultor (textos y gráficos)
	Cuadros		Cuadros interactivos de cálculo de índices
	Cuentas de Balance		Definición de cuentas contables (Sup. Compañías)
	Factores		Factores de producción
	Origen / destino		Tipo de relaciones en la cadena de producción
	Indicadores		Indicadores por rama CIIU
	Genera Indicadores		Ahorra esfuerzo de digitación, genera nuevos indicadores tomando como base la rama 2909
	Encuesta		Ingreso de cuestionario a ser contestado más adelante
	Productos		Productos producidos por una empresa
		Factores	Que intervienen en un producto de una compañía específica
		Cadenas	Relación hacia delante y hacia atrás de un producto específico de una compañía.

Módulo	Opción	Sub opción	Descripción
Procesos			Permite gestionar la información particular de cada cliente del consultor
	Ingresar Producción		Registra datos de producción de cada producto de una compañía seleccionada, según parámetros previamente especificados.
		Factores	Descompone porcentualmente la producción en factores.
	Temas de Usuario		Relacionan temas y gráficos similares a los propuestos por el consultor, ajustados a cada usuario y clientes del consultor.
	Encuesta		Permite registrar respuestas de cada cliente (o compañía) al cuestionario planteado por el consultor.
	Balance		Registra los datos contables del balance de los 4 últimos años.
		Cierre	Pasa el primero de los 4 años a un histórico y permite el ingreso de un nuevo período contable.

Módulo	Opción	Sub opción	Descripción
Edición			Permite realizar funciones específicas sobre algún registro y pantalla, dependiendo su contexto
	Limpiar		Limpia la pantalla para un nuevo despliegue
	Buscar		Permite encontrar un registro en particular.
	Exportar		Genera la información de la pantalla a un documento texto u hoja electrónica.
	Textos Propuestos		Relaciona un tema a un texto propuesto por el consultor
	Gráficos Propuestos		Relaciona un tema a un gráfico propuesto por el consultor
	Textos Usuario		Relaciona un tema con un texto propuesto por el usuario o cliente
	Gráficos Usuario		Relaciona un tema a un gráfico propuesto por el usuario o cliente

Módulo	Opción	Sub opción	Descripción
Reportes			Consulta / imprime / genera información de clientes
	Temas		
		Propuestos	Temario con relaciones a textos y gráficos propuestos por el consultor
		Usuario	El mismo temario anterior, relacionado a textos y gráficos propuestos por el consultor y por el cliente
		Cuadros	Especificar los 5 cuadros actuales: 1.Eficiencia y Eficacia 2.Costos de Inventario 3.Políticas y prácticas de compra 4.Costeo por órdenes de trabajo 5.Costeo por procesos Todos estos programas utilizan variables de trabajo ingresadas por pantalla, realizan cálculos internos y despliegan índices.
	Balance		Resultados de balances de los últimos 4 años por cliente o usuario
	Encuesta		
		Encuesta	Cuestionario planteado por el consultor con las respuestas dadas por cada cliente o usuario
		Evaluación	Evalúa la encuesta de un cliente determinado tomando uno de los 4 años de balance como fuente de información, y los compara con datos referenciales
	Producción		
		Productos	Despliega los productos de un cliente determinado, especificando la rama a la que pertenece, para luego, seleccionando un producto, descomponerlo en sus factores porcentualmente
		Factores	Despliega en una sola pantalla los productos y su descomposición de factores.

Reportes	Indicadores	Prod.Productos	Despliega datos de producción y el destino de la misma, por productos de un cliente seleccionado en un año y período dado, puede seleccionarse un producto para descomponer sus factores
		Fact. Producto	Presenta en una sola pantalla todos los productos de un cliente en un período determinado, desglosando por factores porcentualmente
		CIIU	Despliega los indicadores y sus valores referenciales de una rama CIIU seleccionada previamente
	Productividad	Evaluación	Evalúa cada índice de la rama CIIU a la que pertenece un cliente previamente seleccionado en un período determinado, y lo compara contra los valores referenciales de la rama.
		Factores	Permite comparar los índices de productividad por factores de un cliente y de un producto determinado, entre diferentes períodos, para medir su evolución, calcula índices de PTF de año base y año seleccionado.
	Encadenamiento	Genérico	Similar a la opción anterior, pero es un resumen por cliente y período.
		Por compañía	Despliega los productos de un cliente seleccionado, especificando sus encadenamientos hacia atrás y hacia delante, la rama CIIU a la que pertenece, y su origen (nacional / importado)
		Por Producto	Presenta la misma información que en la opción anterior, pero permite seleccionar un producto específico de un cliente.

2.4. Reestructuración de datos y de código:

En este punto se abordarán varios temas que nos llevan al mismo fin:

- Levantamiento de información con el usuario del sistema, validando que los requerimientos por él planteados sean entendidos por el informático, con firmas de responsabilidad. Los mismos que se detallan en el capítulo 3.
- Acorde a los requerimientos se plantean las modificaciones necesarias a las bases de datos, de forma que se pueda manejar nuevos datos para nuevos procesos.
- Una vez planteados los requerimientos y los cambios necesarios a las bases, estamos en posibilidad de replantear los cambios a los programas de forma que nos permita satisfacer las necesidades del usuario.

Tanto los cambios a las bases de datos como la nueva estructura de los procesos del sistema se verán reflejados en diagramas UML, utilizando la metodología OMT, los mismos que se desarrollan en el capítulo 4.

2.5. Ingeniería Progresiva:

La teoría define a la Ingeniería Progresiva como la "*actividad de manejar sistemas antiguos con tecnología del presente para las necesidades del futuro*".

Bajo esta óptica se hará lo posible para dejar documentado los procedimientos de forma que puedan más adelante aplicarse cambios acorde a las necesidades del usuario sin tener que afectar mayormente las bases y procesos actuales.
A pesar de ser un sistema específico para las actividades planteadas, es decir Industrial y Agrícola se parametrizará lo más posible.
Igualmente se plantea la migración tanto de la base como de la aplicación a las últimas versiones del SQL Anywhere como del Power Builder.

Capítulo 3 Análisis de Requerimientos

Los requerimientos del usuario están enfocados a poner operativo el sistema en un ambiente diferente para el que fue diseñado.

A continuación se exponen brevemente los puntos a considerar en el rediseño, y que están sustentados en el anexo 3:

- *Implementar un control de acceso al sistema por usuario:*
  De forma que exista dos tipos de usuarios con restricción de opciones según especificaciones del cliente.
- *Modificar pantallas en las cuales hace referencia a sucres:*
  Esto debido a que originalmente el sistema fue desarrollado cuando la economía utilizaba el sucre como moneda oficial, por lo que debe adaptarse a las condiciones monetarias actuales.
- *Acoplar textos y gráficos a Office de Microsoft:*
  En vista que las herramientas Office de Microsoft son estandar en el mercado es necesario que los textos y gráficos con los que se relaciona el sistema, estén en este formato.
- *Aumentar nuevos cuadros de trabajo:*
  Por la necesidad del cliente de contar con más herramientas que le permitan jugar y proyectar (what if) datos económicos.
- *Poder manejar empresas agrícolas:*
  Si bien el cliente es un consultor que en un momento puede querer trabajar con empresas de cualquier rama CIIU, por el momento está interesado en empresas del tipo industrial y agrícola.
- *Poder generar índices de ramas dinámicamente con cualquier base:*
  Por lo expuesto en el punto anterior, se hace necesario generar nuevos índices tomando como referencia cualquier rama CIIU, lo que reduce drásticamente esfuerzo de digitación con la consiguiente probabilidad de error.
- *Documentación técnica y de usuario:*
  Que les permita manejarse adecuadamente sin depender del desarrollador de software para mantenimientos futuros, y los usuarios del sistema dispongan de un documento que les ayude a operar adecuadamente cada opción del sistema.

Capítulo 4 Desarrollo del Sistema

4.1 Diagramas UML

A continuación se presenta los diagramas UML necesarios para documentar e implementar los requerimientos planteados por el usuario del sistema.

4.1.1 Casos de uso

4.1.1.1 Diagrama de Casos de Uso

Véase Anexo # 4

4.1.1.2 Descripción de los casos de uso

Caso de Uso: Administrar Usuarios
Actores: Consultor
Descripción: Ingreso de usuarios y determinación de su tipo

Acción de los Actores	Respuesta del sistema
1. El consultor ingresa el identificador de usuario, su nivel de acceso (consultor o usuario) , y la clave de ingreso.	2. El sistema registra la información ingresada por el consultor, valida sus niveles 1= para consultor, 0 para usuario operador, comprueba que sean datos válidos, y los graba en la tabla respectiva.

Caso de Uso: Administrar Códigos
Actores: Consultor
Descripción: Ingreso de códigos del sistema

Acción de los Actores	Respuesta del Sistema
1. El consultor parametriza el sistema con información referente a: – Períodos – Ramas CIIU – Unidades – Temas de interés – Cuadros de trabajo – Cuentas de Balance – Factores	

– Origen/ destino de insumos, productos – Indicadores de competitividad – Genera si es del caso nuevos indicadores – Preguntas de la encuesta	2. El sistema valida esta información, y permite el mantenimiento de esta información. (ingreso, cambios y borrado)

Caso de Uso: Parametrizar Clientes
Actores: Consultor
Descripción: Ingresar datos de un nuevo cliente

Acción de los Actores	Respuesta del Sistema
1. Ingresar datos de un nuevo cliente (compañía) identificando la rama a la que pertenece, y datos exclusivos de cada cliente. 2. Registrar en las tablas respectivas los productos que produce el cliente, determinando los insumos necesarios para la producción, identificando su origen, los productos que produce y el destino de los mismos.	3. El sistema valida la información ingresada, y permite su mantenimiento, es decir ingresar, modificar y eliminar registros de insumos, productos y cadenas.

Caso de Uso: Registrar temas y Cuadros del cliente
Actores: Cliente
Descripción: Documentar con temas y cuadros propios de cada cliente
Observación: Esta opción aplica únicamente cuando el sistema ha sido implementado en un cliente determinado, pues sirve para que cada cliente ilustre los temas planteados por el consultor adaptándolos a su propia realidad.

Acción de los Actores	Respuesta del sistema
1. En el módulo de códigos, el consultor ha planteado ciertos temas de interés, los mismos que tienen relacionados textos y	

cuadros (documentos de word y presentaciones PPS de office) , mediante este proceso el cliente tiene la opción de documentar estos temas con sus propios textos y cuadros adaptados a la realidad de su empresa.	2. EL sistema relaciona los nombres de los textos y cuadros ingresados por el cliente, al temario planteado por el consultor, de forma de asociar un tema determinado a los criterios del consultor y del cliente.

Caso de Uso: Cargar datos periódicos del Negocio
Actores: Cliente, Usuario Operador
Descripción: Ingreso de información periódica del cada cliente
Observación: Esta opción puede tener dos variantes, en caso que el sistema esté instalado donde el cliente, puede ser el mismo cliente quien se encargue del ingreso de los datos, caso contrario lo hará a través del usuario operador en las oficinas del consultor.

Acción de los Actores	Respuesta del Sistema
1. Ingreso Producción	2. El sistema registra producción únicamente para los productos asociados al cliente, en los períodos parametrizados, asimismo reparte a sus factores de acuerdo a la parametrización de cada cliente y producto.
3. Respuestas a la encuesta	4. El sistema despliega el temario para un cliente previamente seleccionado, y registra las respuestas a cada pregunta, para su posterior evaluación.
5. Datos de Balance	6. El sistema permite llevar información contable de hasta cuadro años históricos, para lo cual presenta el plan contable previamente parametrizado, y permite su manejo (ingresar, cambiar).
7. Cerrar Año	8. Como se había mencionado, el sistema solo puede manejar cuatro

	años en línea. Mediante este proceso se incorpora un nuevo período contable, y el más antiguo se graba en tablas históricas.

Caso de Uso: Manejar Cuadros de Trabajo
Actores: Consultor, Cliente
Descripción: Calcular índices mediante juego de datos.
Observación: Esta opción puede ser manejada directamente por el cliente en caso de tener instalado la aplicación en sus oficinas, caso contrario lo hará a través del consultor.

Acción de los Actores	Respuesta del Sistema
1. El usuario (cliente o consultor) puede escoger entre los siguientes cuadros de trabajo: – Eficiencia y eficacia – Costos del Inventario – Políticas y prácticas de compras – Costos por órdenes de trabajo – Costos por Procesos – Costeo ABC – Políticas de Precios – Calcular ciclo del efectivo – Calcular costo del capital – Calcular el valor de la Empresa – Evaluar Proyectos – Calcular la ineficiencia de la infraestructura	2. Para cada caso, el sistema presenta un set de variables, sobre las cuales el usuario puede realizar cambios y obtener resultados de acuerdo a los “inputs” realizados. Cada cuadro representa algoritmos propios de acuerdo a la naturaleza de los índices que está calculando.

Caso de Uso: Extraer Información
Actores: Consultor, Cliente
Descripción: Extraer información del Sistema
Observación: Esta opción puede ser manejada directamente por el cliente en caso de tener instalado la aplicación en sus oficinas, caso contrario lo hará a través del consultor.

Acción de los Actores	Respuesta del Sistema
1. Consultar el catálogo de temas: de interés planteados por el consultor.	
	2. Despliega el temario especificando aquellos temas que tienen relacionado un tema y/o un cuadro, ya sea los propuestos por
	3. el consultor o los propios de cada usuario, desde donde puede mediante selección del cuadro o tema deseado, acceder a la herramienta relacionada de office (procesador de textos o power point) y presentar el archivo respectivo.
4. Consultar datos de balances: de una compañía o cliente previamente seleccionado.	
	5. El sistema presenta los datos de las cuentas contables del cliente seleccionado de los últimos 4 años que las bases de datos mantienen en línea.
6. Encuesta: al cuestionario planteado por el consultor, de una compañía o cliente previamente seleccionado.	
	7. El sistema despliega el cuestionario con las respuestas que el cliente respondió a cada pregunta.
8. Evaluar Encuesta: de las respuestas dadas al cuestionario, el usuario debe seleccionar la compañía, y el período contable con cuyos datos quiere obtener resultados de su gestión. Si el cliente quiere datos de otro período, debe seleccionarlo y oprimir el botón de Cálculo.	
	9. El sistema aplica un algoritmo basado en los valores de los balances contables, en datos de

	ventas y producción, y datos propios de la compañía, con lo cual obtiene índices por cada tema, los mismos que son comparados con valores referenciales de forma que el cliente pueda determinar qué tan apartado o alejado se encuentra de los valores sugeridos por el consultor.
10. Productos: que produce una compañía o cliente determinado, y de esa pantalla, seleccionar la descomposición de factores de un producto específico.	
	11. El sistema despliega los productos relacionados a una compañía específica, y los factores porcentuales de un producto seleccionado.
12. Productos, factores: de una compañía o cliente de quien se desea en una sola presentación los productos y su descomposición porcentual de los factores.	
	13. El sistema solicita se seleccione una compañía, de la cual se despliegan todos sus productos y sus factores en forma porcentual.
14. Producción: seleccionar compañía y el período del cual el usuario desea saber la producción y su destino.	
	15. El sistema despliega la producción de una compañía, año y período previamente seleccionado, especificando la cantidad producida y el destino de la misma repartida porcentualmente.
16. Producción, Factores: Selecciona la compañía y período del cual el usuario desea saber la producción expresada en factores.	
	17. El sistema presenta en una sola pantalla la producción de una compañía y período previamente seleccionado, desglosando su composición porcentual en factores.

18. Indicadores CIIU: El usuario selecciona de una lista de todas las ramas, aquella de la que desea ver los índices y sus respectivos valores referenciales.	19. El sistema presenta todos los indicadores relacionados a una rama CIIU previamente seleccionada, muestra además los valores referenciales de la rama.
20. Evaluación de índices: El usuario selecciona la compañía, y período sobre el cual desea comparar los índices.	21. El sistema realiza cálculos basados en datos contables del período seleccionado, datos de producción y datos propios de la compañía o cliente. Aplica un algoritmo propio para cada índice, y lo presenta junto con su valor referencial.
22. Factores de Productividad: El usuario selecciona la compañía, producto y período de producción del cual desea ver sus factores.	23. El sistema calcula los factores de productividad tanto de un año base (parametrizable) como de un período seleccionado, y los presenta para evaluar su comportamiento.
24. Factores de productividad genérico: selecciona únicamente la compañía y período del cual desea el usuario saber los índices de productividad.	25. Similar al punto 22, pero no lo hace por producto sino por compañía.
26. Encadenamiento por compañía: selecciona la compañía de la cual desea ver las cadenas de cada producto.	27. EL sistema presenta todos los productos de una compañía previamente seleccionada, con datos de encadenamiento hacia atrás y hacia delante, especifica además la rama CIIU a la que pertenece la cadena.
28. Encadenamiento por producto: el usuario selecciona de una lista el producto específico del cual desea ver el encadenamiento.	

	29. Similar al punto 26. el sistema presenta la misma información, pero de un producto específico.

4.1.2 Diagrama de Clases

Véase Anexo # 5

Descripción de clases

Nombre de la Tabla (clase)	Descripción
RAMASCIIU	Almacenan datos que identifican a las ramas de acuerdo a normas internacionales
INDICADORES	Contienen los índices o temas que nos interesa evaluar por cada rama.
PERIODOS	Definen lapsos de tiempo en los cuales se evalúan los clientes
COMPANIAS	Contienen datos relacionados a un cliente o compañía.
PRODUCTOS	Describen los productos que produce un cliente determinada.
UNIDADES	Contienen todas las medidas tanta de producción como de valores y tiempo.
INSUMOS	Contienen una lista de materiales necesarios para la elaboración o producción.
PRODUCTOINS	Relacionan los insumos que requiere un producto específico
ORIGENDESTINO	Contienen de donde vienen los insumos y hacia donde se orientan los productos producidos.
RELACIONES	Contienen por cada producto qué necesita para su fabricación, y hacia donde va una vez obtenido.
PRODPRODUCTO	Almacenan la producción de un producto determinado en un lapso de tiempo.
CONSUMOINSPROD	Almacenan los insumos utilizados para la producción de un producto en un lapso de tiempo.
TEMASP	Listado de temas agrupados lógicamente propuestos por el consultor.
TEMASU	Contienen relaciones a textos y gráficos

	de los temas de la tabla TEMASP
TEMASC	Contienen relaciones a cuadros de trabajo propuestos por el consultor.
ENCUESTA	Catálogo de preguntas lógicamente agrupados a ser respondidas por los clientes.
RESPUESTAS	Contienen las respuestas de cada cliente al cuestionario planteado en la tabla ENCUESTA
CUENTAS	Almacenan el catálogo de cuentas contables regulado por la Superintendencia de Compañías.
VALBALANCES	Contienen datos financieros de los últimos 4 años acorde a la tabla CUENTAS
HISBALANCES	Guardan un histórico de balances de años previos a los 4 años.
USUARIOS	Contiene los usuarios autorizados a manejar el sistema, especifica su clave y nivel de acceso.

4.1.3 Diagrama de Objetos

Véase Anexo # 6

4.1.4 Diagrama de Secuencia

Véase Anexo # 7

4.1.5 Diagrama de Colaboración

Véase Anexo # 8

4.1.6 Diagrama de Actividad

Véase Anexo # 9

4.1.7 Diagrama de Componentes

Véase Anexo # 10

4.1.8 Diagrama de Despliegue

Véase Anexo # 11

4.2 Pruebas del Sistema

Sistema de Evaluación de Competitividad EVACOM.

Con la finalidad de garantizar la calidad del software que se está desarrollando, se han realizado las siguientes pruebas de caja negra junto con el usuario las opciones del sistema más importantes y principalmente de aquellas opciones nuevas y que han sufrido modificaciones.

1. Ingreso al Sistema.

Propósito de la prueba
Verificar que únicamente ingresen al sistema aquellos usuarios autorizados, y que una vez en el sistema, este permita o restringa las opciones de los diferentes menús según los requerimientos del usuario.

Procedimiento
Digitar los dos tipos de usuario (uno a la vez) que tiene el sistema, esto es, *digitador* y *administrador* del sistema, probar también con un usuario diferente de forma que valide usuario y contraseña.

Resultado
Si el usuario no existe despliega el mensaje correspondiente, caso contrario ingresa al sistema, en el cual restringe ciertas opciones cuando se trata de un usuario tipo *digitador*, entre ellos el módulo de usuarios.

Si el usuario es del tipo *administrador*, entonces habilita todas las opciones de todos los menús del sistema.

2. Generación automática de Índices

Propósito de la prueba
Verificar que el procedimiento de generación automática de índices, trabaje adecuadamente, de forma de evitar el ingreso manual.

Procedimiento
Previamente debe cargarse nuevas ramas CIIU, por la opción de *Códigos*, *Ramas CIIU*, luego tomamos la opción de *Genera índices* también del módulo de *Códigos* en la cual se despliega en la parte superior las ramas ciiu nuevas que aún no tienen índices, y en la parte inferior aquellas ramas que sí tienen índices.

Se probo seleccionar la o las ramas nuevas sobre las que deseamos generar índices, y la rama base o modelo.

Se realizaron, además, pruebas intencionales de error para validar que una y solo una rama pueda servir de base, o que si se selecciona una rama base, se seleccione al menos una rama ciiu nueva.

Resultado
Una vez pasados los filtros de control, el sistema genera índices nuevos para las ramas que hemos seleccionados de acuerdo a la rama base seleccionada, y despliega un mensaje del número de registros (índices) generados.

3. Cuadros de Trabajo

Propósito de la prueba
Comprobar que los nuevos cuadros de trabajo estén realizando los cálculos según las especificaciones provistas por el usuario.

Procedimiento
Para la prueba se ha tomado al azar el cuadro de trabajo de *Valor de la Empresa,* al cual se accede a través del módulo de *Reportes, Temas, Cuadros,* y de la lista de temas que tienen cuadros relacionados, tomamos aquel especificado.

Al hacer doble clic en la línea que contiene el nombre del cuadro de trabajo, se despliega una pantalla en la cual debe ingresarse una serie de campos con los cuales deben realizarse cálculos para obtener lo siguiente: *flujo de efectivo, y valor actual.*

Resultado
Los resultados concuerdan con las especificaciones proporcionadas por el usuario a momento de realizar el requerimiento del cuadro de trabajo respectivo.

4. Temas Propuestos por el Consultor

Propósito de la prueba
Comprobar que los *links* hacia textos y gráficos estén funcionando con Office de Microsoft.

Procedimiento
Ingresar al menú de *Reportes, Temas, Consultor,* entonces despliega una pantalla con un temario el cual indica si tiene un texto y / o gráfico relacionado.

Hacemos clic sobre un tema que tenga *S* bajo la columna de *Texto,* tomamos el menú *Edición, Texto Consultor.*

El mismo procedimiento para un tema que tenga *S* bajo la columna *Gráfico* pero en el menú *Edición* tomamos la opción de *Gráfico Consultor.*

Resultado
Despliega una pantalla con una imagen del texto o gráfico relacionado, al hacer doble clic sobre esta imagen, abre la herramienta respectiva del Office y edita el documento en Word, o la imagen en PowerPoint, de Office de Microsoft según los requerimientos del usuario.

5. Evaluación de Indicadores

Propósito de la prueba
Comprobar que las formulaciones aplicadas a ciertos índices de ramas agrícolas, (cambios a existentes y nuevos índices), trabajen según especificaciones proporcionadas por el usuario.

Procedimiento
Configurar una empresa de forma que pertenezca a una rama agrícola, es decir asignándole un CIIU agrícola, luego ingresar datos de extensión y producción en el perfil de la compañía, cargar los balances necesarios y por último tomar la opción *Reportes, Indicadores, Evaluación de indicadores.* En esta pantalla escogemos *Año, Empresa,* y presionamos el botón *Cálculo* para obtener y revisar los índices respectivos.

Resultado
Se pudo determinar que los algoritmos trabajan de acuerdo a los requerimientos del usuario.

4.3 Instalación del sistema

Requerimientos Básicos:

EL sistema está desarrollado en Power Builder, el mismo que recomienda la siguiente configuración mínima:
- Procesador 486 o superior.
- Windows 95 , 98 2000, XP, NT workstation
- 32 MB en RAM.
- 20 MB de espacio en el disco duro.
- Monitor VGA.
- Debido a que el sistema tiene conexiones con textos y gráficos desarrollados en Microsoft Office 2000 recomienda tener cargado esta versión del Office, caso contrario estas opciones no funcionarán adecuadamente.

Pasos de Instalación:

Utilizando el CD de instalación, seguir los siguientes pasos:

1. *Instalación de la base de datos:* Escogiendo setup.exe de la carpeta sqlany5504 del disco de instalación y utilizando el wizard .

2. *Instalacion del runtime de Power Builder.* Escogiendo setup.exe de la carpeta DDDK6 del disco de instalacion y utilizando el wizard .

3. *Crear el DSN:* en administrador de ODBC dentro del Panel de Control. Para SQL Anywhere, direccionandolo a la base de datos del sistema: c:\evacom\archivos.db

4. Verificar que en el autoexec.bat y/o en el path del sistema se encuentre lo siguiente:

   C:\PWRS\shared;C:\Archivos de programa\Sybase\SQL Anywhere 5.0\win32;

5. Copiar la carpeta evacom al disco C.

6. Copiar la carpetas graficosp y textosp al disco C.

7. Crear el ícono de acceso directo en el escritorio de Windows
   Destino : C:\evacom\evacom.exe
   Iniciar en: "C:\sybase\dddk6\Deployment DLLs"

4.4 Manuales usuario

Véase Anexo 12

4.5 Disco con el sistema

Adjunto en la pasta del manual

Conclusiones y Recomendaciones

Una vez concluido este trabajo, puedo establecer algunos puntos relevantes que pueden servir a aquellos desarrolladores de software que en un momento dado se encuentren en una situación de dilema entre rehacer aplicaciones de software y reconstruirlas empleando técnicas de Reingeniería de Software.

La analogía que un autor hace en referencia a la compra de una casa que requiere ser readecuada antes de ser habitada, es de lo más ejemplificadora, pues en términos de software es perfectamente aplicable el principio, es decir si las estructuras y diseño de una aplicación están vigentes y pueden ser utilizadas como base del sistema nuevo al que se pretende llegar, es aconsejable y viable utilizar reingeniería.

Por otro lado debemos considerar que cada caso es diferente y requiere de un estudio individualizado para poder determinar la aplicación o no de Reingeniería de Software, no es posible aplicar patrones o estándares como argumentos de comparación y evaluación.

Existirá casos en los cuales es recomendable rehacer el sistema, pues la reingeniería puede resultar más costosa y complicada que desarrollar una aplicación nueva.

Debe además existir un entendimiento completo de la problemática que un proceso de este tipo conlleva por parte del usuario del sistema, pues este pasa a ser una pieza clave en el procese de reingeniería.

En base a lo expuesto y a mi experiencia en este proyecto de Reingeniería de Software, considero recomendable su aplicación, previo a un estudio minucioso tanto del funcionamiento y diseño del sistema actual, como del producto final al que se desea llegar.

Bibliografía

Ingeniería de Software, Roger S. Pressman,

UML Grady Booch, James Rumbough, Ivar Jacobson

UML SAMS Teach Yourself, Joseph Schmuller

Trabajo de Titulación “Análisis y Diseño de Sistemas Orientado a Objetos” Ing. Xavier Palacios UDLA 2001

Publicaciones en Internet

ANEXOS