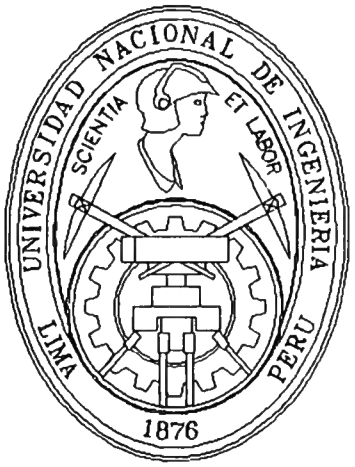
Universidad Nacional de Ingeniería
FACULTAD DE INGENIERIA INDUSTRIAL Y DE SISTEMAS




# Elementos de una Estrategia de Desarrollo de Software con Tecnología de Objetos

**T E S I S**

Para Optar el Título Profesional de:

**INGENIERO DE SISTEMAS**

**MARCO ANTONIO VIDAL HUAMAN**

Lima - Perú
1997

**UNIVERSIDAD NACIONAL DE INGENIERIA**
**FACULTAD DE INGENIERIA INDUSTRIAL Y DE SISTEMAS**

# "ELEMENTOS DE UNA ESTRATEGIA DE DESARROLLO DE SOFTWARE CON TECNOLOGIA DE OBJETOS"

Trabajo realizado por

**MARCO ANTONIO VIDAL HUAMAN (814003-B)**

para optar el Título de INGENIERO DE SISTEMAS

ASESORES:

**Ing. Juan Carlos Sotelo**
**Ing. Luis Zuloaga Rotta**
**Ing. Celedonio Méndez**

UNI, febrero de 1997

*A mi tía Mercedes,*
*por su cariño y apoyo*

**DESCRIPTORES TEMATICOS**

- OBJETOS
- DESARROLLO DE SOFTWARE
- ESTRATEGIA DE MIGRACION
- METODOLOGIA
- LEGACY SOFTWARE
- CICLO DE DESARROLLO
- WRAPPERS
- METODOLOGIAS DE ANALISIS Y DISEÑO
- PROYECTO DE DESARROLLO

# *PROYECTO DE TESIS*

# *ELEMENTOS DE UNA ESTRATEGIA DE DESARROLLO DE SOFTWARE CON TECNOLOGIA DE OBJETOS*

## TABLA DE CONTENIDO

# 1. OBJETIVO

La tecnología de objetos merece en nuestros días un gran interés de la comunidad informática. Si bien los inicios y fundamentos de esta tecnología se remontan por lo menos veinte años atrás, es en la actualidad que cobra vigencia y se convierte en una necesidad para el replanteo de los esquemas de desarrollo de proyectos de software.

Esto no significa que lo que venimos haciendo ahora esté completamente errado, de lo que se trata es de ampliar la cobertura de los sistemas de información en las organizaciones y de mejorar los sistemas vigentes usando esta nueva óptica. Esto se traduce en hacer el desarrollo más rápido mejorando la calidad del mismo , lo que significa bajar los costos tanto durante el análisis, diseño y construcción y sobre todo durante la etapa de mantenimiento por cambios. Además, este enfoque debe contemplar la fácil adaptación a los cambios y evolución del sistema.

El objetivo de este proyecto de tesis es proporcionar los lineamientos para el desarrollo de proyectos bajo el enfoque de orientación a objetos. Nuestro objetivo no es desarrollar un estudio teórico de lo que representa la tecnología de objetos o alguna metodología en particular; no está orientada hacia la filosofía sino hacia la implementación, es una revisión práctica y orientada hacia el proceso de desarrollo, que involucra los aspectos técnicos (metodologías, herramientas, lenguajes de programación, migración) y los aspectos de gestión de proyectos (recursos humanos, etapas en el desarrollo, conceptos de calidad de software, capacitación).

Sin embargo, el presente trabajo no puede dejar de lado el marco conceptual y sustentatorio del enfoque, por lo que es necesario contar con un capítulo (Capítulo 2) orientado a cubrir este aspecto. Este capítulo se concentra en los conceptos fundamentales del enfoque.

El Capítulo 3 constituye el centro de nuestra investigación y analiza cada uno de los componentes en el desarrollo de proyectos de software usando la tecnología de objetos.

Así pues, en el Capítulo 3 se tratarán los requisitos o perfiles de los miembros del equipo, algunas pautas para el control del proyecto, las posibilidades de migración de nuestros sistemas tradicionales y algunos conceptos asociados con la calidad.

El Capítulo 4 ofrece una visión general sobre las metodologías de desarrollo que en la actualidad son las más aceptadas, y algunos criterios para la selección de una metodología.

El Capítulo 5 muestra de manera clara los pros y contras de la tecnología de objetos, de modo que podamos evaluar dentro de nuestro contexto la conveniencia de su adopción en el corto plazo. Además se intenta hacer una síntesis de los principales aspectos a tener en cuenta en el desarrollo de proyectos de software que hagan uso de la tecnología de objetos y se proponen alguna recomendaciones.

Dentro del presente trabajo se ilustran algunos ejemplos basados en mi experiencia en el campo profesional dentro del sector bancario, y especialmente con el producto SARA BANK propiedad de CosapiSoft S.A. Una breve introducción a los conceptos manejados por este producto se incluye en los anexos 1 y 2 del presente trabajo.

# 2. CONCEPTOS DE LA TECNOLOGÍA DE OBJETOS

## *2.1 EL ESPACIO PROBLEMA VERSUS EL ESPACIO DE LA SOLUCIÓN.*

El espacio problema es aquel ambiente que requiere ser informatizado. Es un ambiente que tiene connotaciones organizacionales y técnicas, pero sobre todo semánticas o culturales.

Es el problema en sí mismo, y dentro del cual el analista no permanece mucho tiempo, pues se traslada rápidamente a lo que se denomina espacio de la solución, que se caracteriza por el modelamiento basado en estructuras de representación y limitada por categorías a priori. Es por eso que Vauquier refiere que *modelar* es *mentir*.

Las categorías mentales y sus estructuras de representación difieren de método en método. Debemos ser conscientes que nunca llegaremos a abarcar la totalidad de un espacio problema y debemos resignarnos a ver la realidad escaparse, sin embargo lo que debemos buscar son aquéllos métodos de modelamiento que reduzcan la brecha o la distorsión entre el problema y la solución.

La solución que surge como muy próxima de la representación natural es aquella que percibe el usuario dentro de su práctica y lejos de marcos técnicos y conceptuales ajenos a su lenguaje. El utilizador percibe entidades estables, dotadas de autonomía, y que se manifiestan por contener la información accesible y un comportamiento. Estas entidades son los objetos. Es por eso que la tecnología de orientación a objetos ofrece una perspectiva diferente pues simplemente hablamos de un *salto semántico,* al pasar del espacio problema al espacio de la solución.

## *2.2 OBJETOS*

Citaremos algunos conceptos que definen a un objeto:

*Un objeto es cualquier cosa que tenga un límite claramente definido* según B. Cox.

Para Smith y Tockey, *un objeto representa un elemento individual, identificable, unidad o entidad, sea real o abstracta, que posea un rol bien definido dentro del dominio del problema..*

Rumbaugh dice algo parecido, pues señala que *un objeto es una cosa, abstracción o concepto con límites definidos y significado dentro del problema.*

Ivar Jacobson define un objeto como *aquello caracterizado por un número de operaciones y un estado que refleja el efecto de estas operaciones.*

Una definición menos filosófica es la que proponen Martin y Odell, para ellos *un objeto es cualquier cosa, real o abstracta, dentro de la cual almacenamos datos y los métodos que los manipulan.*

Considero que la definición que Grady Booch propone es la más adecuada, él sostiene que *un objeto se define como todo aquello que posee una identidad, un estado y un comportamiento.*

Resumiendo

El objeto debe representar más que un dato o una entidad de un modelo entidad-relación , más aún que una agrupación de datos y procesos, es el modelo exacto y completo de un objeto del mundo real desde el punto de vista del modelamiento.

Un modelo orientado a objetos consiste entonces en un conjunto de objetos, los mismos que son parte claramente delimitadas del sistema modelado. Cada objeto contiene información individual (por ejemplo, la cuenta corriente de Juan Pérez tiene un número, un saldo y un historial de movimientos).

Para cada objeto en un modelo orientado a objetos se le asocia un comportamiento que le permita interactuar con el mundo exterior, así tenemos que en el caso de la cuenta de Juan, ésta necesita interactuar con los otros componentes del sistema y así podemos establecer una lista de operaciones que podemos hacer sobre ella (ver figura 2.1):

- actualizar saldo líquido,
- actualizar saldo contable,
- actualizar saldo 24 horas,
- actualizar saldo 48 horas,
- actualizar saldo +48 horas
- consultar saldos,
- consultar 10 últimos movimientos,
- mostrar firmas y condiciones,
- consultar datos generales, etc.

Es importante señalar que lo único que hacemos al ver un objeto por fuera es conocer cuál es el comportamiento y las operaciones que se ejecutan sobre él, mas no cómo trabajan internamente.

Por otro lado, tenemos que la información de un objeto implica la relación con otros objetos. Estas relaciones pueden ser estáticas o dinámicas. Una relación estática consiste en una relación por tiempo considerable, por ejemplo una cuenta tiene relación estática con las firmas asociadas a ella, las firmas no cambian constantemente para una cuenta, salvo cuando uno de los apoderados

de la cuenta cambie. El otro tipo de relación se refiere a lazos volátiles entre objetos, así una relación dinámica puede darse entre una cuenta y la transacción de depósito que opera sobre ella, la relación durará el tiempo que la transacción se ejecute.

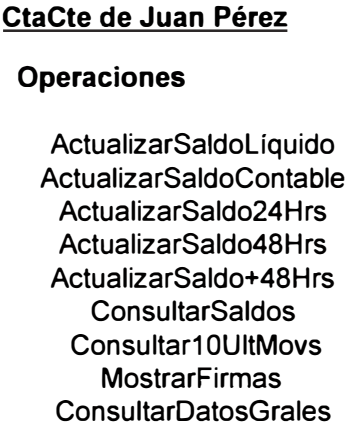



**Figura 2.1** El objeto CtaCte de Juan Pérez

Como podemos apreciar las relaciones dinámicas, son aquellas mediante las cuales dos objetos se comunican entre ellos. las relaciones estáticas describen relaciones entre objetos que componen otros objetos, como el caso de las firmas con las cuenta.

Para esto nos basamos en el uso de jerarquías de partición o agregación (al respecto Jacobson hace diferencias) de esta manera se obtienen relaciones *compuesto de.*

La dinámica del modelo orientado a objetos es creado haciendo uso de las relaciones dinámicas, mediante el envío de estímulos a otros objetos. El envío de este estímulo implica que el receptor es estimulado para comportarse de un modo específico, produciéndose la ejecución de una operación sobre el objeto receptor. En programación, este estímulo se denominará *mensaje.*

Por ejemplo para el caso de la cuenta de Juan Pérez, si deseamos rebajar el saldo, le enviaremos el estímulo "actualizar saldo líquido" con un importe negativo. Cuando la cuenta de Juan Pérez reciba el estímulo, lo interpretará y ejecutará las operaciones que para ese caso han sido definidas. Si su saldo es menor al monto enviado en el estímulo y la cuenta no tiene línea de crédito, la cuenta responderá que no puede sobregirar la cuenta.

Toda la información en un sistema orientado a objetos es almacenada dentro de sus objetos y sólo puede ser manipulada cuando los objetos ejecutan las operaciones. El comportamiento y la información están *encapsulados* dentro del objeto. El único medio de afectar un objeto (variar su estado, propiedades o atributos) es ejecutando operaciones sobre él. Por eso se dice que los objetos soportan *ocultamiento de información (information hiding)*. Para trabajar con un objeto no necesitamos saber cómo se comporta ni cómo está almacenada la información, solo necesitamos saber cuáles son las operaciones que él ofrece. En resumen, *para el mundo exterior todo lo que el objeto muestra es su interfaz.*

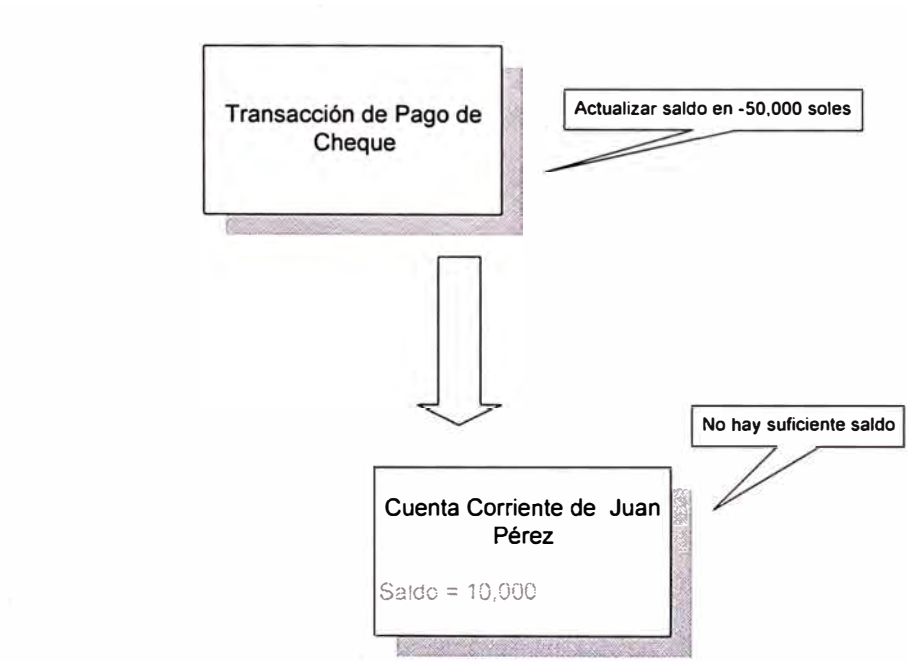



**Figura 2.2** Estímulo ActualizarSaldo sobre CtaCte de Juan Pérez

## *2.3 CLASES E INSTANCIAS DE CLASE*

En el ejemplo de la cuenta de Juan Pérez , las operaciones que sobre ella podíamos ejecutar podían ser válidas si hablamos de otra cuenta, la de José López. Es por esto que su agrupación como objetos con similar comportamiento e información permite definir el concepto de *clase.* Una clase representa una plantilla (template) de objetos y describe cómo estos objetos están estructurados internamente. Objetos de la misma clase tienen la misma definición para sus operaciones y para la estructura de su información.

Las instancias creadas a partir de las clases nos proveen el comportamiento dinámico que deseamos plasmar en el modelo. Cuando las instancias empiezan a comunicarse, es cuando el comportamiento del sistema se pone de manifiesto. Una instancia debe saber qué estímulo enviar a otra instancia. Si una instancia envía un estímulo a otra instancia, pero no tiene que preocuparse de la clase a la que pertenece la instancia receptora, decimos que tenemos *polimorfismo.*

Si usamos el concepto de clase, podemos asociar algunas características al grupo de objetos. Podemos considerar la clase como una abstracción que describe todas las características comunes de los objetos que forman parte de ella. En los sistemas orientados a objetos, cada objeto pertenece a una clase. Un objeto que pertenece a una clase se denomina *instancia* de clase. De modo que para nosotros objeto o instancia son sinónimos.

*Una instancia es un objeto creado a partir de una clase.* La clase describe la estructura de la instancia, mientras que el estado actual de la instancia es definida por las operaciones desarrolladas sobre ella.

En el ejemplo de las cuentas bancarias, la clase cuenta define la estructura de la información y el comportamiento de la cuenta de Juan Pérez. La identidad de esta cuenta la diferencia de la cuenta de José López.

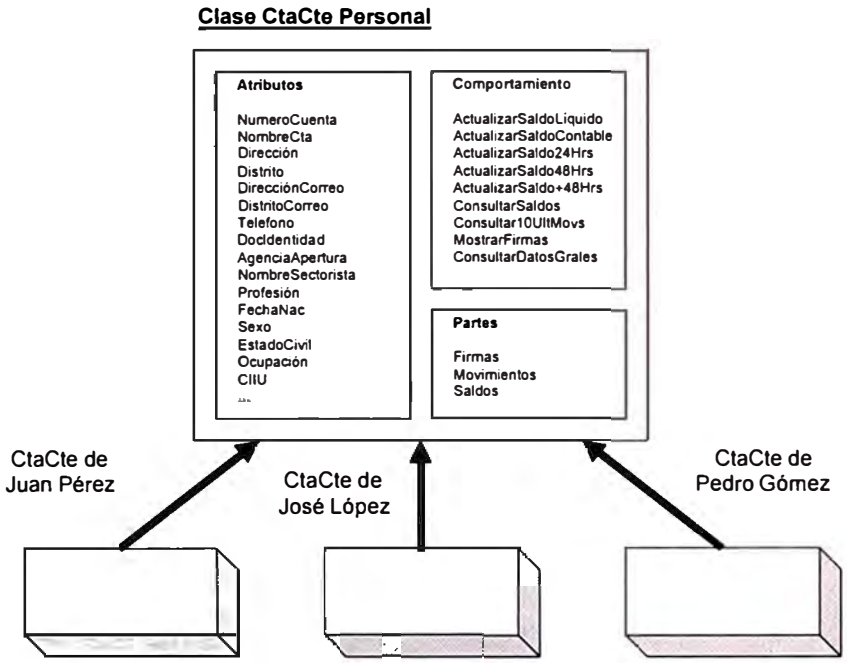



**Figura 2.3** Las CtasCtes de Juan Pérez, José López y Pedro Gómez son instancias de la clase CtaCtePersonal.

## *2.4 POLIMORFISMO*

Las instancias creadas a partir de las clases nos proveen el comportamiento dinámico que deseamos plasmar en el modelo. Cuando las instancias empiezan a comunicarse, es cuando el comportamiento del sistema se pone de manifiesto. Una instancia debe saber qué estímulo enviar a otra instancia. Si una instancia envía un estímulo a otra instancia, pero no tiene que preocuparse de la clase a la que pertenece la instancia receptora, decimos que tenemos *polimorfismo.*

*Polimorfismo significa que la instancia que envía el estímulo no necesita saber la clase de la instancia receptora.* La instancia receptora puede pertenecer a cualquier clase. Ver figura 2.4.

En el caso de las cuentas podemos decir que existen varias clases de cuentas: clase cuenta corriente personal, clase cuenta corriente empresarial, clase cuenta de ahorros, clase cuenta de depósito a plazo, etc. El utilizar la operación ActualizarSaldoLíquido sobre una cuenta corriente y sobre una cuenta de ahorros permite ilustrar la característica de polimorfismo de la operación . En el

primer caso, para un cargo la cuenta se sobregirará si hay suficiente línea, pero en la cuenta de ahorros esto no sucederá nunca pues las cuentas de ahorros nunca pueden estar sobregiradas.


**Figura 2.4** Podemos aplicar la operación Capacidad sobre todos estos objetos. Sin embargo, en cada caso el resultado es diferente.

Un estímulo (operación) puede ser interpretado de diferentes maneras dependiendo de la clase receptora. Esto significa que quien recibe el estímulo determina su interpretación, y no la instancia que envía el estímulo. si la operación ActualizarSaldo es enviada a la cuenta corriente, ésta podría aceptar que el saldo se sobregirara, mientras que si fuera enviado a una cuenta de ahorros, ésto no sucedería pues la cuenta de ahorros no permite sobregiro.

Otra manera de describir el polimorfismo es decir que consiste en la capacidad de una operación de ser implementada de manera diferente en diferentes clases, lo cual mas que una definición es la consecuencia de lo antes mencionado.

Un concepto relacionado (pero no similar) es el *dynamic binding*, que significa que el estímulo no está atado a una operación en la clase de la instancia receptora hasta que éste es enviado.

Es conveniente, sin embargo, no dejar irrestricto el envío de estímulos a cualquier objeto es por eso que se propone un polimorfismo limitado. La implementación de esta restricción se realiza normalmente utilizando la jerarquía de herencia.

En conclusión:

El polimorfismo es muy importante para el desarrollo de los modelos. Es la instancia receptora del estímulo quien determina cómo el estímulo debe ser interpretado, y no la instancia emisora.. La instancia emisora sólo necesita saber que otra instancia puede desarrollar un comportamiento determinado, no necesita saber a qué clase pertenece dicha instancia ni qué operación realmente desarrolla el comportamiento deseado. Esta es una herramienta muy poderosa para desarrollar sistemas flexibles.

De esta manera, sólo se necesita saber qué debe ocurrir y no cómo va a ocurrir. Mediante esta delegación de lo que debe ocurrir a la instancia receptora, se obtiene un sistema resistente a las modificaciones. Si necesitamos añadir una nueva clase, ésta modificación sólo afectará el nuevo objeto, y no aquéllos que le envían estímulos.

Un ejemplo claro de ésto se presenta en las actualizaciones de saldo. Si un nuevo producto bancario sale al mercado, las operaciones que trabajen sobre dicha clase de cuenta seguirán enviando el estímulo de actualización de saldo. La implementación de dicha actualización será transparente para las instancias emisoras.

## *2.5 HERENCIA*

Hemos visto como intuitivamente hemos identificado que la clase cuenta corriente y cuenta de ahorros son variantes de una clase denominada cuenta bancaria. Podemos reunir las características comunes de las clases cuenta corriente y cuenta de ahorros, y ubicarlas en la clase cuenta bancaria. De esta manera en cuenta bancaria describiríamos todo lo que es común a cuenta corriente y cuenta de ahorros. Al agrupar las características comunes en una clase específica, debemos permitir que las clases originales hereden de ésta.

Esto permite que sólo sea necesario describir las características particulares en las clases originales.

Mediante la herencia, podemos mostrar las similitudes entre clases y describir estas similitudes dentro de una clase que puede ser heredada por otras. En consecuencia, podemos reutilizar descripciones comunes. Es por esto que la herencia es promocionada como la idea central de la reutilización en la industria del software.
A través de la herencia, obtenemos otra ventaja. Si necesitamos modificar algunas características de la clase cuenta bancaria, es suficiente hacer esta modificación en un sólo lugar. Si una modificación sobre la cuenta bancaria es hecha, ambas clases, cuenta corriente y cuenta de ahorros, heredarán esta nueva definición.

Otro aspecto importante es la reducción de información en la clase cuenta corriente y cuenta de ahorros. La única información que contienen es aquélla que las diferencia. Es por esto, que la herencia contribuye a la eliminación de la redundancia.

La facilidad de modificación no sólo ocurre dentro de las descripciones de clase. El adicionar nuevas clases puede ser fácilmente realizado mediante la descripción de cambios a las clases existentes.

Mediante el proceso de extraer y compartir las características comunes, podemos *generalizar* clases y ubicarlas encima dentro de la jerarquía. Del mismo modo, si necesitamos añadir nuevas clases, podemos encontrar que ya existe una clase que ofrece la información y el comportamiento requerido por esta nueva clase. Entonces dejaremos a la nueva clase heredar de esta clase y sólo añadiremos aquello que es único para la nueva clase. Estaremos *especializando* la clase.

Las clases ubicadas bajo una clase dentro de la jerarquía de herencia son llamadas descendientes de la clase. Clases ubicadas encima son llamadas ancestros. Dado que las jerarquías de herencia pueden incluir varias clases, generalmente se enfatiza la relación entre dos clases. Si una clase hereda directamente de otra clase se denomina descendiente directo. De igual modo la primera clase es ancestro directo de la segunda clase. En alguna bibliografía encontramos que el ancestro directo es denominado *padre* y el descendiente directo es denominado *hijo*.

Los ancestros creados con el propósito principal de ser heredados por otros son llamados clases abstractas. Generalmente no tienen instancias, aunque es posible. Una clase desarrollada con el objeto de crear instancias de ella es denominada *clases concretas*. esto no impide que una clase concreta pueda ser ancestro de otras clases.


**Figura 2.5** La clase B es descendiente de la clase A y la clase A es ancestro de la clase B.

Generalmente cuando hablamos de superclase nos referimos a aquella clase que se encuentra encima de una clase dentro de la jerarquía de herencia, e incluso llamamos así a todas las clases encima de una clase determinada. Por el contrario subclase es aquélla clase que se encuentra debajo de otra dentro de la jerarquía de herencia, y al igual que el caso anterior, se denominan subclases a todas las clases debajo de una clase determinada.

Esto puede traer confusión pues los prefijos super y sub es estos casos quieren decir algo completamente opuesto. Una superclase no es una clase más completa que la clase bajo ella, por el contrario tiene una estructura de

información y comportamiento más sencillos. Una subclase es una clase más rica en información y comportamiento. Por ello, y a pesar que implementaciones de lenguaje como el Smalltalk trabajan estos términos, es preferible presentar la herencia en términos de *extensión* y *especialización.*

La estructuración de clases se hace posible con la ayuda de las jerarquías de herencia. La estructuración de clases dentro de una jerarquía de herencia nos permite trabajar con clases y definir nuevas clases mediante la identificación de diferencias entre las nuevas y las ya existentes.

Por ejemplo, supongamos que tenemos una jerarquía de herencia para nuestro ejemplo de cuentas bancarias, donde Cuenta Bancaria tiene dos hijos: Cuenta Corriente y Cuenta de Ahorros. Si aparece una cuenta corriente orientada a empresas será necesario añadir una clase. Es obvio que esta nueva clase que denominaremos cuenta Corriente Empresarial tiene todas las características de la clase Cuenta Bancaria y tiene mucho de similar con la clase Cuenta Corriente. Para añadir esta clase existen tres posibilidades: (figura 2.6)

(1) La clase Cuenta Corriente Empresarial sea descendiente de la clase Cuenta Bancaria y se definen las diferencias entre Cuenta Corriente y Cuenta Bancaria.

(2) La clase Cuenta Corriente Empresarial hereda de la clase Cuenta Corriente y nosotros definimos las diferencias; por ejemplo, la exoneración del cobro de gastos.

(3) La creación de una nueva clase Cuenta Corriente Personal, y definir el cobro de manera estándar. De esta manera cuenta Corriente Personal y Cuenta Corriente Empresarial heredarán de la clase Cuenta Corriente.

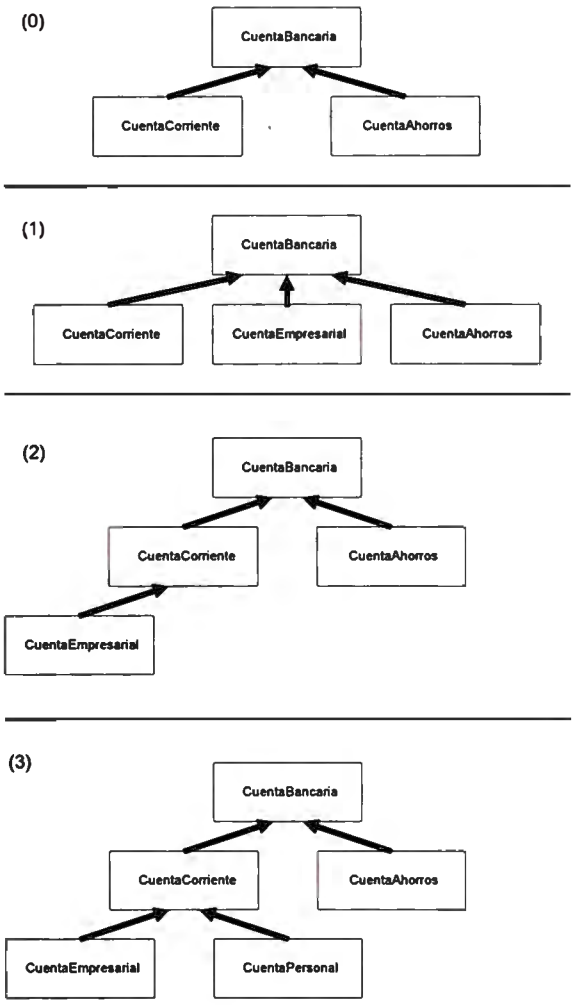



**Figura 2.6** Tres posibilidades para añadir la clase CtaCteEmpresarial

Estas tres posibilidades nos muestran la variedad de posibilidades que se nos presentan al modelar. Cual de esas alternativas utilizar depende mucho de cómo se utilizaran dichas clases. Las preguntas a responder pueden ser : ¿Cómo se manejarán las modificaciones posteriores? ¿Qué es lo que realmente deseamos modelar? ¿Qué tan importante es una clara comprensión del modelo? ¿Qué tan difícil es reestructurar las clases? ¿Cuál es el precio de la reestructuración? la mejor alternativa depende de la respuesta a estas preguntas.

En teoría, para efectos de comprensión deberíamos escoger la tercera alternativa. Des esta manera no tendríamos redefinición, y mantendríamos una

estructura comprensible. El costo es la reestructuración de la estructura de clases y la creación de una nueva clase.

Cuando una nueva clase es añadida, buscamos un candidato a ser ancestro directo. Generalmente tenemos cuatro posibilidades para añadir una nueva clase:

(1) Podemos ir hacia arriba en la jerarquía de herencia para ver si un ancestro se ajusta para establecer una relación de herencia.

(2) Podemos describir la clase desde el inicio como una clase independiente.

(3) Podemos reestructurar la jerarquía de herencia, de modo que obtengamos la herencia requerida. Ver figura 2.7

(4) Podemos redefinir las características que deseamos cambiar.

Las dos primeras soluciones son triviales. La tercera solución es la más aceptable, pues mantiene la jerarquía de clases ordenada y clara. Para reestructurar de esta manera, sin embargo, se requiere bastante trabajo , parte del trabajo consiste en encontrar la mejor estructura y la otra parte ver las consecuencias que la modificación tendrá en el sistema diseñado sobre la base de una estructura jerárquica anterior.

La cuarta solución se denomina generalmente *overriding*. Significa la redefinición de parte del comportamiento y/o estructura de la información a partir del ancestro. El proceso de overriding es muy discutido, pues si bien es fácil y flexible para modificar las clases existentes, puede dañar la comprensión de la jerarquía de clases, como operaciones con el mismo nombre con significados semánticos diferentes en diferentes clases . de este modo, con el overriding la herencia no es transitiva. Pues el descendiente no hereda todas las características; sólo algunas de ellas serán heredadas, mientras que las otras serán redefinidas.

Para comprender cómo debe usarse la herencia de manera adecuada, revisaremos los propósitos principales de la herencia, los cuales fueron anteriormente mencionados.

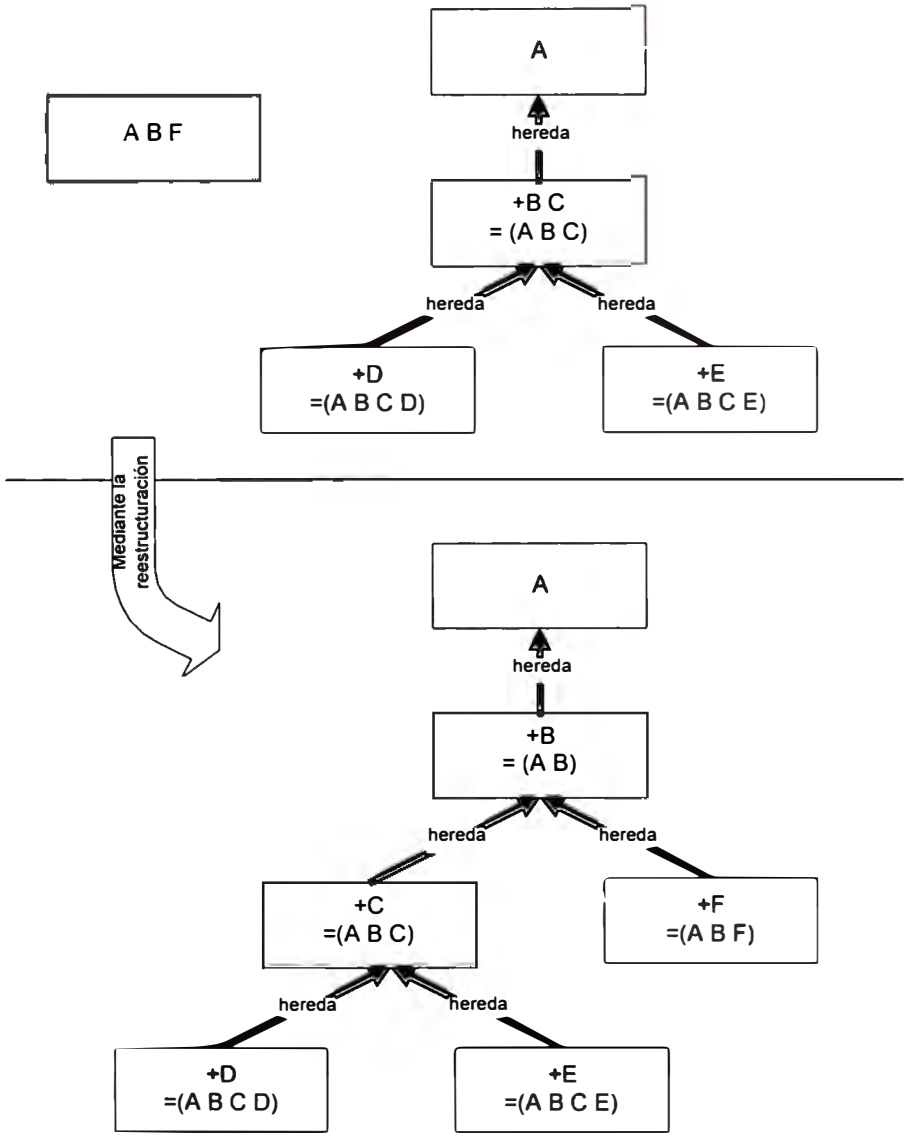



**Figura 2.7** Mediante la reestructuración de la jerarquía de herencia, se puede obtener las clases requeridas.

*Reutilización.* La más importante razón de utilizar herencia es porque simplifica la reutilización del código. La reutilización puede ocurrir de dos maneras diferentes. La primera es que dos clases tienen partes similares; estas partes son extraídas y ubicadas dentro de una clase abstracta, de la cual las dos clases

heredarán. Esta clase representa las partes comunes de ambas clases y no necesita tener sentido por sí misma. La otra manera es partir de una librería de clases. Encontrar las clases con las operaciones que se necesitan y hacer las modificaciones necesarias.

*Subtipos*. Una clase puede considerarse como la implementación de un tipo. Una clase A define cierto comportamiento. Si es posible que un descendiente de A sea utilizado en todos los sitios donde la clase A es utilizado, entonces decimos que las clases son compatibles en su comportamiento. El descendiente representa un subtipo de la clase A. En términos prácticos significa que el descendiente debe tener por lo menos la misma interfaz que sus ancestros. Esto ocurre si la herencia desarrolla una extensión, y no una redefinición de algo ya definido por el ancestro.

*Especialización*. Si el descendiente es modificado de tal modo que no es compatible en comportamiento con su padre, se dice que la clase ha sido especializada. Normalmente, la información y operaciones han sido redefinidas o suprimidas.

*Conceptualización*. Este uso de la herencia corresponde a la semántica intuitiva de la realidad, y nuestra habilidad natural de clasificar los objetos del mundo real. Por ejemplo, decir "un perro es un mamífero", implica que el perro posee todas las características de un mamífero.

Estas diferentes maneras de utilizar la herencia no son exclusivas ni incompatibles. Sin embargo, si nos concentramos en el rol y responsabilidad de cada objeto, en su comportamiento, basado en subtipos, construiremos estructuras de herencia que sean mantenibles y robustas.

Para efectos de prototipeo quizás sea innecesario tanto rigor, teniendo en cuenta que se trata de código provisional. Generalmente, lo que ocurre en estos casos es lo que se viene a denominar *spaghetti inheritance*.

Cuando se describe una nueva clase, si deseamos usar las características de dos o más clases existentes, debemos heredar de todas ellas. Esto se denomina *herencia múltiple*. Esto significa que una clase tiene más de un ancestro directo. La herencia múltiple puede ser justificada si se trata de dos puntos de vista complementarios del modelo, sin embargo su utilización es muy polémica. La principal desventaja proviene de la reducción en la comprensión de la jerarquía de clases. Ver figura 2.8.

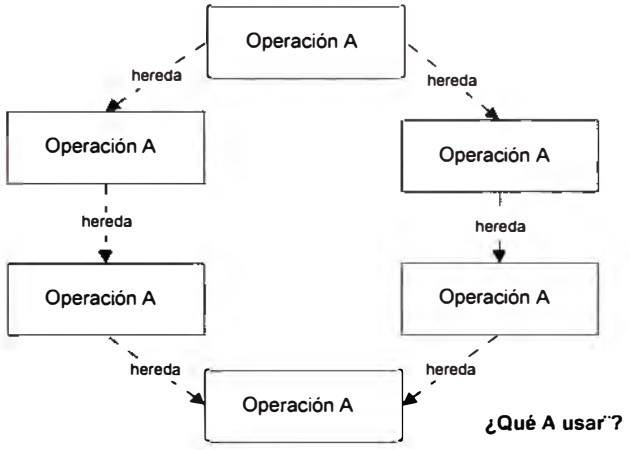



**Figura 2.8** Herencia Múltiple. Dos padres con la misma operación.

# 3. DESARROLLO DE SOFTWARE UTILIZANDO LA TECNOLOGÍA DE OBJETOS

## 3.1 EL PROCESO: EL CICLO DE DESARROLLO

Es necesario distinguir el ciclo de vida del ciclo de desarrollo.

Ciclo de Desarrollo: Se aplica a un proyecto. proporciona un esquema temporal y organizacional donde se combinan las diferentes tareas a cumplir para obtener el sistema o producto.

Ciclo de Vida: Se aplica al sistema o producto. Abarca desde su concepción (idea inicial) hasta su retiro (obsolescencia). Por lo tanto puede abarcar varios proyectos (varios ciclos de desarrollo). Involucra mas que una división de tareas, y engloba consideraciones sobre el producto: etapa de producción o explotación, evolución, mantenimiento, adecuación a nuevos estándares, etc.

Con el objeto de diseñar un buen sistema, las diferentes metodologías de desarrollo de sistemas han propuesto describir la secuencia de pasos del desarrollo del proyecto, en lo que se ha llamado *el ciclo de desarrollo de un sistema*. Los modelos de ciclo de desarrollo de sistemas son aplicables a la tecnología de objetos, consisten en la manera de desarrollar las diferentes etapas del proceso de desarrollo.

El más utilizado dentro del desarrollo convencional es aquel que organiza el trabajo de modo estructurado, secuencial y es denominado modelo cascada (*waterfall model*), como se muestra en la Figura 3.1.

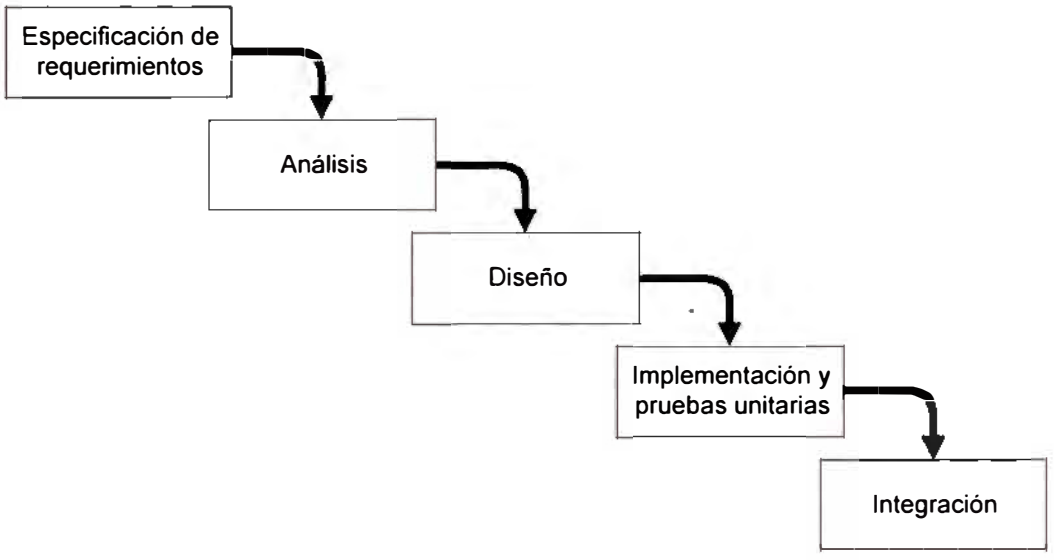



**Figura 3.1** El Modelo en cascada o *waterfall*

Esta cascada describe el flujo del proceso de desarrollo. El trabajo empieza con la creación de la especificación del requerimiento del sistema. Esto es elaborado por la persona o personas que solicitan el sistema o por los desarrolladores en estrecha comunicación con los solicitantes. A partir de esta especificación de requerimientos, el análisis y una descripción lógica del sistema son confeccionados. Este trabajo podría ir llevándose a cabo con la especificación de los requerimientos. El diseño del sistema es luego completado y seguido de la implementación (programación) de módulos pequeños. Estos módulos son probados primero individualmente y luego en conjunto. Cuando las pruebas de integración son completadas, el sistema puede ser entregado.

El modelo en cascada fue el punto de partida de varios otros enfoques sobre el ciclo de desarrollo, que recogían de la experiencia nuevos conceptos. Por ejemplo, no percatamos que en algunos casos (la mayoría), *el agua debía fluir hacia arriba* al final del proceso. Varios modelos fueron diseñados para describir estos nuevos hechos, el más popular es el modelo espiral (Boehm 1986), como se muestra en la figura 3.2. El modelo espiral puede describir cómo el producto se desarrolla para producir nuevas versiones, y cómo una

versión puede ser desarrollada de manera incremental del prototipo al producto completo. Sin embargo, no cuestiona la división de fases, sólo se añade al modelo el carácter iterativo.

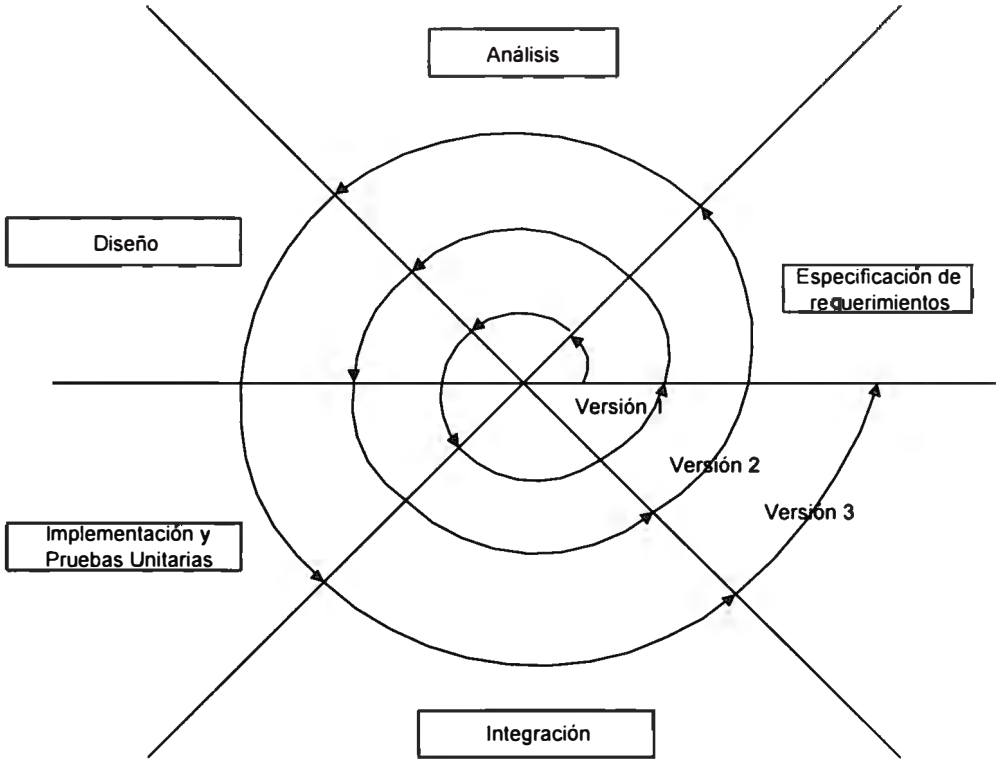



**Figura 3.2** El modelo espiral del Boehm

El ciclo de desarrollo en espiral contiene por lo general todas estas fases, y el desarrollo siempre ocurre de manera incremental alrededor de ellas. Generalmente el desarrollo de un sistema se caracteriza por fases iniciales turbulentas con estabilizaciones posteriores, en ese sentido el método de desarrollo debe ayudarnos a salir lo más pronto posible de esa turbulencia dentro del proceso de desarrollo. Por eso es necesario que dentro de la etapa de análisis nos quedemos el tiempo suficiente para comprender el sistema, pero no tanto como para empezar a considerar detalles que luego serán modificados en el diseño. Esto significa que gran parte del esfuerzo se concentra en la fase de análisis. A pesar de lo que parece este ciclo de desarrollo es esencialmente secuencial, pues no permite ningún paralelismo.

Una división típica del tiempo para los proyectos es la sugerida en la figura 3.3, donde el eje horizontal representa el tiempo y el vertical representa la cantidad de esfuerzo. Inicialmente un número reducido de personas desarrollan el análisis y a continuación el diseño. Estas actividades son desarrolladas iterativamente. Cuando la estructura del sistema se estabiliza, más personas intervienen en la implementación y pruebas. Sin embargo, las actividades de análisis y diseño pueden producirse aún cuando las pruebas han empezado.

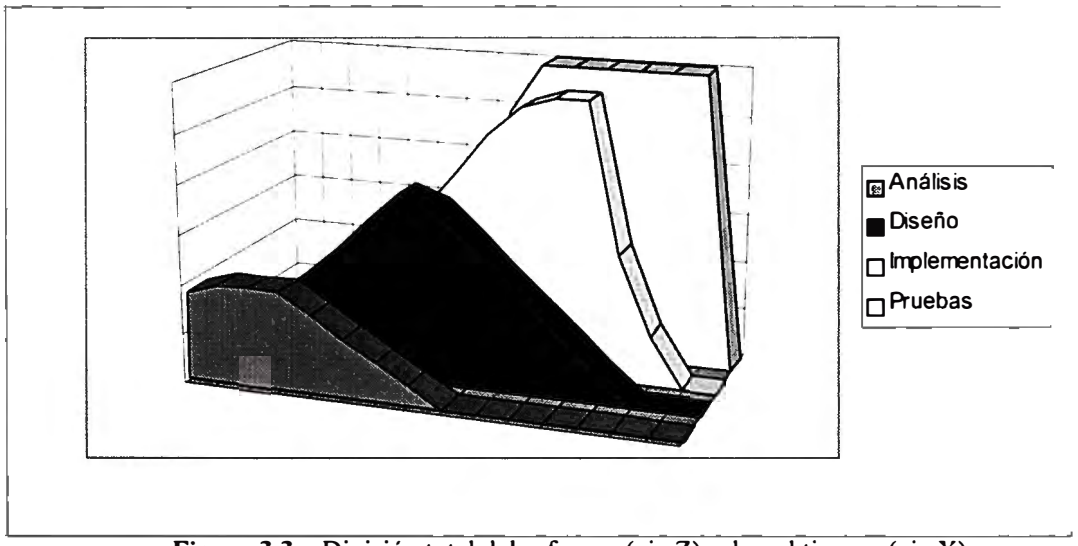



**Figura 3.3** División total del esfuerzo (eje Z) sobre el tiempo (eje X) entre las diferentes actividades

Dentro de variantes de estos modelos de desarrollo podemos hablar del desarrollo por prototipeado. Al respecto existe una confusión con la entrega de un prototipo como parte del análisis y/o diseño. En nuestro caso nos referimos a que todas las tareas se organizan alrededor de la construcción de una aplicación. El concepto clave dentro del ciclo por prototipeado es el denominado retroacción, que viene a ser el retorno a la construcción del prototipo luego de una demostración y decisión. Se debe tener en cuenta que la iteración excesiva acarrearía un costo excesivo para el constructor. Para el éxito del ciclo por prototipeado, es la etapa de evaluación la que preocupa más al jefe de proyecto. Las etapas productivas (especificación, desarrollo del prototipo, pruebas y documentación) se manejan del mismo modo que en los proyectos clásicos.

Todo descansa sobre el buen funcionamiento de las sesiones en las cuales el equipo del proyecto somete el prototipo a la crítica y observación del cliente. Frente al prototipo, el trabajo de validación es bastante complicado debido a la superposición de varios planos: aquel de las funcionalidades, aquel de las características requeridas (implícitas o explícitas), aquel de la interfaz, aquel de la ergonomía. Como muestra la figura 3.4, podemos definir el proceso por prototipeado, como una sucesión de etapas de especificación y de etapas de implementación, a cada cambio corresponde una decisión. Por esto, este modelo de ciclo de desarrollo no sólo se orienta a la programación, sino también al afinamiento de los requerimientos y especificaciones.

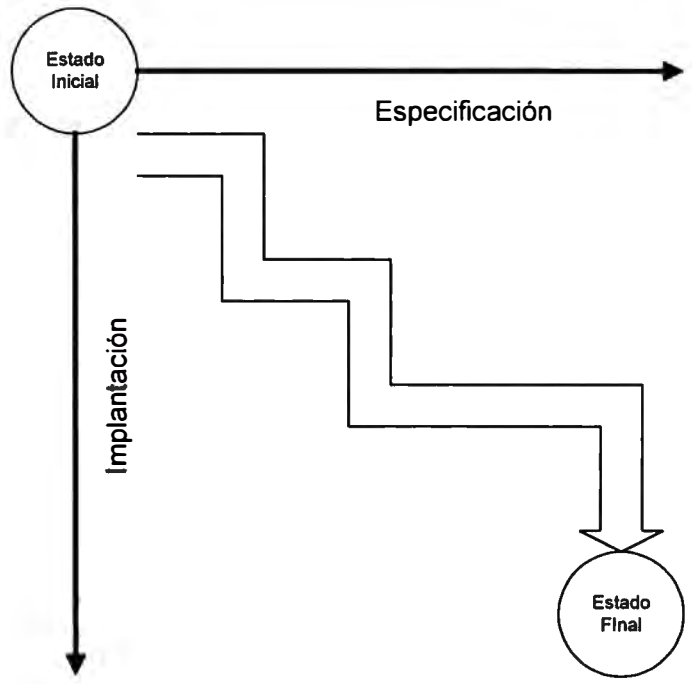



**Figura 3.4** Proceso Iterativo del Prototipo

El prototipeado tal cual lo acabamos de describir representa uno de los tipos de prototipeado que existen y expresa el ciclo de desarrollo de un sistema. A continuación pasemos a revisar los que consideramos tres tipos de prototipeado: el evolutivo, el de especificación y el exploratorio.

*Prototipeado Evolutivo*

Es utilizado cuando el sistema objetivo será construído en el lenguaje con el cual se desarrolla el prototipo. Es del tipo al que hacemos referencia líneas arriba. Es apropiado cuando los requerimientos son continuos, pues las evaluaciones permanentes nos permitirán ir corrigiendo el alcance del sistema final.

*Prototipeado de Especificación*

Es aquel que más que un ciclo de desarrollo representa una técnica dentro de las fases de análisis y diseño. Se utiliza cuando no está definido el lenguaje o herramientas de desarrollo y por lo tanto tomamos una herramienta diferente para el desarrollo de un prototipo. La desventaja de este modo de trabajo es que tendremos que programar nuevamente de cero cuando hayamos identificado nuestro lenguaje o herramienta de desarrollo. Otra desventaja es que el prototipo desarrollado puede contener funcionalidad demasiado compleja para el lenguaje escogido, lo que hará más cara la solución. Por lo que el uso de esta técnica debe contemplar sus posibles consecuencias al momento de desarrollar el prototipo.

*Prototipo Exploratorio*

Este método es considerado como un generador de ideas en los ambientes de desarrollo e investigación, pero considero que pocas veces podrá ser usado en el desarrollo comercial donde se manejan presupuestos claros y el control del ciclo de desarrollo es esencial. El Smalltalk es un ambiente excelente para desarrollar este tipo de actividades. Sin embargo por constituir un método para evaluar la viabilidad técnica de soluciones, generalmente no contempla conceptos como requerimientos de hardware, software y performance, y debe ser restringido a labores de investigación. Podemos considerar al prototipeado exploratorio como un subtipo del prototipeado de especificación.

Existen otros modelos de ciclos que intentan plasmar el proceso de desarrollo de manera formal. Sin embargo, considero que simplemente son variantes de

los tres anteriormente mencionados. Respecto a como intervienen los conceptos del ciclo de desarrollo cuando utilizamos la tecnología de objetos podemos mencionar lo siguiente:

- Está claro que el ciclo de desarrollo de un sistema contempla las siguientes fases:
  - Análisis
  - Diseño
  - Codificación o Programación
  - Integración

  Estas son las fases estándar y en general en cualquier modelo de ciclo de desarrollo lo que existen son matices y nomenclaturas particulares.

- El ciclo de desarrollo en cascada refleja un desarrollo secuencial de fases. Sin embargo en la práctica lo que hacemos es adelantar algunas labores de diseño e incluso de programación durante la fase de análisis, obteniendo una superposición de fases que permite acortar los tiempos de desarrollo. Esta práctica, que es común en los proyectos es válida en la medida que no tengamos que rehacer por adelantarnos al resultado de fases anteriores y sólo en casos donde el conocimiento del aplicativo o sistema es bueno. A este respecto, considero que podemos trabajar con un ciclo de desarrollo en cascada en aquellos proyectos donde el aplicativo o sistema a desarrollar sea de nuestro amplio conocimiento o su dinámica (cambios en el tiempo) sea mínima, pues la iteración será mínima y las correcciones también.

  Un ejemplo de utilización del ciclo de desarrollo en cascada se presenta a continuación.

  El banco X cuenta con un sistema central en un ES/9000 que permite la explotación de sus sistemas básicos (cuentas corrientes, ahorros, cuentas a plazo, cartera, contabilidad, tesorería, etc.). El banco para procesar sus

transacciones en las agencias utiliza terminales con emulación 3270. La decisión del banco consiste en modernizar sus agencias bancarias con sistemas de correo electrónico, herramientas de automatización de oficina, e integrar dentro de la agencia el ambiente transaccional (emulación) con aplicaciones de venta de servicios. Esto involucra el uso de redes de microcomputadoras y productos de software con interfaz gráfica. Para proveer el front-end gráfico se utilizará el VisualAge, que provee facilidades de HLLAPI.

En este caso como se aprecia, los cambios en los ambientes transaccionales del banco son conocidos, el set de transacciones no cambia constantemente, los objetos VisualAge a utilizar son básicamente de interfaz y su dominio es bastante rápido. Este proyecto aplica perfectamente al ciclo de desarrollo en cascada.

- El ciclo de desarrollo en espiral permite desarrollar los aplicativos haciendo uso de ciclos en cascada iterativos. La desventaja de este modelo consiste en que si el análisis y diseño de la primeras iteraciones son errados, tendremos que modificarlo y no enriquecerlo, como debiera ser. Se aplica en proyectos de desarrollo de gran envergadura, de modo que las iteraciones añaden complejidad a la arquitectura diseñada en las iteraciones iniciales.

  Dentro de SARA BRANCH, el componente que ejecuta las transacciones se denomina programa MONITOR. Este programa, actualmente en COBOL, se encarga de hacer llamadas (calls) a funciones en una secuencia que viene determinada por lo que denominamos esquema de transacción (almacenada en un archivo). Así cada transacción para completarse utiliza el programa monitor, quien es el encargado de ejecutar las funciones, una tras otra.

  El ciclo de desarrollo en espiral permitiría planificar el desarrollo de SARA BANK con tecnología de objetos de manera incremental. Así en la primera iteración se desarrollaría un programa MONITOR que permita ejecutar las funciones una tras otra, procesar los extornos de transacciones y procesar las

transacciones con autorización. A continuación una segunda iteración permitiría contar con un MONITOR capaz de seleccionar la ejecución de bloques de funciones dependiendo de condiciones de la transacción, trabajar encadenamiento de transacciones para operaciones más complejas como depósitos múltiples cheques, etc.

- El ciclo de desarrollo por prototipeado permite el desarrollo de sistemas cuando es muy difícil obtener toda la información del usuario desde un inicio, ya sea por falta de definición, o por situaciones cambiantes, o porque el sistema está acompañado de una consultoría que será la que defina las características finales del aplicativo.

  Un ejemplo de aplicación de un ciclo de desarrollo por prototipeado es la personalización del módulo de Venta Cruzada para un banco. SARA BANK cuenta con un módulo de Venta Cruzada (cross-selling). Las características de la venta cruzada son conocidas, sin embargo la manera de presentar las interfaces, las modalidades de acceso a la información y las fuentes de obtención de información se van definiendo en el camino sobre la base de una estructura ya definida (la versión estándar del componente Venta Cruzada), y teniendo en cuenta que cada banco desea distinguirse de su competencia.

- No existe ciclo de desarrollo modelo que aplique a la tecnología de objetos. El ciclo de desarrollo se adaptará como en el caso de los sistemas de tecnología clásicos a las características de la aplicación a desarrollar.

## 3.3 EL EQUIPO DE DESARROLLO

### 3.2.1 Características

Los candidatos a trabajar en los proyectos iniciales con la Tecnología de Objetos deben cumplir ciertas características. Teniendo en cuenta que este tipo de actividad se basa en el factor humano y su adecuada

conducción, el reclutamiento de los mejores elementos es fundamental. A continuación enumero algunas características que considero son importantes al momento de seleccionar el personal de trabajo.

Buscar elementos con "mente abierta" a nuevas tecnologías y paradigmas, de otro modo, lo único que se conseguirá es tener dentro del equipo, personas que siempre busquen resolver los problemas de manera tradicional. Es por eso que el personal elegido debe ser capaz de asumir como suyo el reto de desarrollar con TO y trabajar en equipo en una misma dirección.

Buscar elementos con gran capacidad de autoaprendizaje. En nuestro medio es muy difícil buscar instituciones que proporcionen capacitación en TO y si esta se ofrece, es de carácter introductorio e informativo. La mayor cantidad de información deberá ser asimilada vía el autoestudio. Por otro lado, si no existe esa vocación personal de actualizarse permanentemente, el desarrollo de proyectos con componentes tecnológicos nuevos siempre será un problema.

Buscar elementos capaces de trabajar con información vaga e incompleta. Es casi imposible conseguir buena documentación de los proyectos iniciales pues nos encontramos en proceso de aprendizaje. Luego, la documentación residirá en la lectura del código. Otro inconveniente radica en la necesidad de aplicar metodologías que son nuevas para nosotros y algunos de cuyos conceptos, hasta llegar a un nivel de madurez, son gaseosos o volátiles, y han sido manejados de manera intuitiva.

### 3.3.2 Integrantes del equipo

A continuación se propone una estructura del equipo de desarrollo, esta es la estructura ideal, el número de personas de cada tipo varía según el tamaño del proyecto.

- *Constructor de Métodos*: Encargado de implantar de manera independiente métodos a partir de un diseño estrictamente trabajado. Deben ser monitoreados por el diseñador de clases o el arquitecto. Las características técnicas de estos miembros del equipo se enumeran a continuación:
  - ☑ Aprendizaje de conceptos de ingeniería de software orientada a objetos, así como de un lenguaje y ambiente de desarrollo orientado a objetos.
  - ☑ Aprendizaje de las librerías de clases del proveedor escogido.
  - ☑ Experiencia de desarrollo de un sistema pequeño a modo de prueba.

- *Encargado de las pruebas de las clases*: Al mismo tiempo que surge el diseño, debe aparecer un plan de pruebas. Se concentran en la calidad de las clases. Es responsable de verificar la documentación y el código.

  Perfil o habilidades:
  - ☑ Conceptos de Aseguramiento de la Calidad.
  - ☑ Conocimientos de herramientas para el desarrollo de las pruebas.

- *Arquitectos de la Aplicación y Diseñadores de Clases*: Son los que diseñan el producto, tienen una visión global del sistema de manera clara y con todo el detalle.

☑ Los conocimientos de técnicas y herramientas de prototipeado.

☑ Conocer mas de un entorno y/o lenguaje de programación orientado a objetos.

☑ Lo mismo que el Constructor de Métodos pero con mayor experiencia. Por lo menos haber participado en un proyecto anterior como constructor de métodos.

☑ Gran capacidad de comunicación con el cliente y de entendimiento del negocio.

- *Jefe de Proyecto*: Encargado de la revisión de librerías que eviten inventar la rueda. Revisa periódicamente el código y la documentación del producto. Verifica plazos y presupuestos. El perfil del jefe de proyecto es similar al que se busca para un proyecto de desarrollo con tecnología tradicional. Si bien sus conocimientos sobre la TO deben estar claros, uno de los principales objetivos de su labor es el control de una adecuada distribución del trabajo, un control permanente sobre los esfuerzos y tiempos dedicados a las actividades asignadas a cada miembro del equipo y al mismo tiempo, un manejo cuidadoso del presupuesto. Así mismo, es el encargado de informar, y en algunos casos de alertar, sobre la situación y avance del plan de trabajo a los directivos.

Esta organización responde a una definición teórica de los elementos del equipo de desarrollo, sin embargo debe utilizarse en función al tamaño del proyecto y a la calidad de recursos con que se cuenta. Por ejemplo para un proyecto de 40 meses/hombre, como sería aproximadamente el desarrollo de SARA BANK (SARA Applimatic y SARA Branch) con TO, la estructura del equipo sería de la siguiente manera:

| *Cantidad* | *Perfil* |
|---|---|
| *1* | *Jefe de Proyecto* |
| *1* | *Arquitecto de la Aplicación* |
| *2* | *Diseñadores de Clases* |
| *1* | *Encargado de pruebas de clases* |
| *2* | *Constructores de Métodos* |

Se entiende que todo este personal no es necesario desde un inicio. Se debe contar con una organización básica formada por los tres primeros elementos de la lista (cuatro personas) desde un principio hasta la etapa de diseño. Los mismo que en la etapa de construcción participarán para el apoyo en la construcción de métodos y la implementación del diseño.

### 3.2.3 El entrenamiento

El 90% de los programadores a finales de la década estarán involucrados en el desarrollo de aplicaciones orientadas a objetos. Incursionar en la tecnología de objetos significa:

- Entender los conceptos de objetos
- Practicar la programación orientada a objetos (lenguaje y ambiente de programación)
- Comprender los enfoques de análisis y diseño orientados a objetos
- Conocer las bibliotecas de objetos que proporcionan los ladrillos para la construcción de aplicaciones orientadas a objetos.

A continuación bosquejamos un plan de entrenamiento para el personal de proyecto:

**Conceptos de TO**: Conceptos básicos de la tecnología de objetos y fundamentos del método a utilizar. Duración: 1 a 2 días.

**Gestión de Proyectos**: Características especiales del método. Puntos de corte y evaluación del avance. Definición de algunas métricas. Duración: 1 día.

**Revisión General del Método**: Mediante el uso de ejemplos para introducir el método en el ciclo de desarrollo de la aplicación o sistema. Los participantes deberán estar familiarizados con la TO. Duración 3 a 4 días.

**Análisis**: Un exhaustivo estudio del proceso de análisis. Enfatizado con un ejemplo de alguna magnitud para aplicar el método. Duración: 3 a 6 días.

**Diseño y Construcción**: Un exhaustivo estudio del proceso de diseño y construcción, enfatizado con un ejemplo de alguna magnitud. El lenguaje a utilizar no está cubierto en el entrenamiento, sin embargo su impacto en el proceso de construcción debe tenerse presente. Duración: 3 a 6 días.

Respecto a los costos de este entrenamiento y capacitación, éstos deben contemplar: tiempo productivo perdido, debugging, entrenamiento en hardware y software, e infraestructura de soporte.

En software podemos gastar desde $100 por el Smalltalk V a $6,000 por el Smalltalk-80 de Parc-Place. Una herramienta de análisis y diseño como OMTool desde $500 hasta $40,000 dependiendo de la plataforma. Una base de datos OO como Versant, Gemstone, Ontos u ObjectStore cuesta entre $5,000 y $50,000.

El costo importante también lo representa la capacitación, aunque el nivel de dicha capacitación es muy escasa en nuestro medio. Algunos cursos especializados dictados por instructores de nivel cuestan aproximadamente $1,500 (30 horas) por persona.

De lo anterior se desprende que el autoestudio y el autoaprendizaje son los medios a nuestro alcance. Dicho aprendizaje podrá ser reforzado con cursos especializados en el exterior, si contamos con los recursos. No debemos enviar a capacitar a profesionales para quienes la TO es algo completamente nuevo a cursos en el extranjero. Es mejor familiarizarlos con la TO, de modo que dicha capacitación sea mejor aprovechada.

Otro costo a medir es el tiempo que toma contar con una persona que ya haya superado la curva de aprendizaje. Se cree que los programadores con orientación a objetos podrán ser efectivos a partir del séptimo mes.

Los no informáticos y los profesionales jóvenes adquieren rápidamente los *skills* de la TO, mientras que los programadores experimentados sufren el shock inicial del cambio de mentalidad antes de comenzar a ser productivos. Ambos tipos de profesionales son importantes dentro del equipo y a la larga asimilan homogénea y adecuadamente dichos conceptos.

## 3.3 LOS LENGUAJES O AMBIENTES DE DESARROLLO

En la actualidad deben existir más de cien lenguajes de programación que se califican como orientados a objetos. Hasta la fecha no existen teorías formales que fundamenten la llamada programación orientada a objetos., como sí las hay para la programación funcional y lógica. Por lo tanto, la discusión al respecto, sobre un lenguaje de programación y si es o no orientado a objetos es bastante etérea. A continuación se presenta la clasificación que presenta Ian Graham.

### 3.3.1 Clasificación de los lenguajes de la TO.

Se definen dos clases de lenguajes de programación:

- Lenguajes orientados a objetos.
- Lenguajes con características de orientación a objetos.

Aunque la definición es bastante subjetiva, podemos decir que se denominan lenguajes orientados a objetos a aquellos que soportan formalmente por lo menos las nociones de encapsulamiento, identidad de objetos, abstracción, herencia y envío de mensajes. Dentro de esta clasificación se encuentran:

- Simula
- Smalltalk
- C++

Los lenguajes con características de orientación a objetos son aquellos que no poseen la pureza del Smalltalk, pero que sin embargo poseen características orientadas a objetos. Dentro de este grupo se distinguen:

- Ada
- Object Pascal
- OO-Cobol

Adicionalmente, dentro de esta clasificación se incluyen aquellos lenguajes que se basan en estructuras de objetos reutilizables, generalmente gráficos, denominados de la generación del *componentware*. Tal es el caso de PowerBuilder y Visual Basic.

A continuación revisaremos el Smalltalk, como fiel representante de este primer grupo y al OO-Cobol como la nueva alternativa para el desarrollo en TO.

### 3.3.2 Smalltalk

El Smalltalk es considerado como la representación más pura de la orientación a objetos. Existen dos dialectos: el Smalltalk-80 y el Smalltalk V. Este lenguaje y a la vez entorno de desarrollo es el resultado del trabajo de Alan Kay, Adele Goldberg, Daniel Ingalls y otros investigadores en el Xerox PARC durante los años 70. Como se

indico líneas arriba, el Smalltalk no es un lenguaje sino mas bien un entorno de programación con editores, browsers de jerarquías de clases y muchas cualidades de un lenguaje de cuarta generación.

Es el Smalltalk el que estableció la noción de envío de mensajes como la única manera de comunicar objetos. Dentro de Smalltalk, todos son objetos. La noción de la existencia de metaclases hace un poco difícil su comprensión para los que se inician. De otro lado, para mantener la consistencia del lenguaje trata los números como objetos, sin embargo esto lo hace menos eficiente.

El Smalltalk no maneja el concepto de tipos, es decir, no es *strongly typed*, de modo que siempre se producen errores cuando se envía un mensaje que no puede ser gestionado, y todo esto en tiempo de ejecución.

Otro aporte importante del Smaltalk es su entorno de trabajo, el mismo que ha servido de referencia a muchos otros productos y lenguajes con orientación a objetos.

El concepto de metaclase fue introducido por el Smalltalk. Una metaclase, como se sabe, no es más que una clase de clases que sirve como template para generar otras clases, pero que no puede ser instanciada (*instantiated*).

La mayoría de las versiones de Smalltalk sólo proveen herencia simple, por tal razón, las aplicaciones que requieren de herencia múltiple están diseñadas de manera poco natural, para salvar dicha deficiencia.

Aunque el Smalltalk es un ambiente productivo y comprensible para los programadores, el enlace dinámico (*dynamic binding*), la depuración (*garbage collection*) y la profusa interfaz gráfica para los usuarios lo hacen muy pesado en requerimientos de hardware.

A continuación se muestra un resumen de las ventajas y desventajas del Smalltalk:

*Ventajas*

- Posee una uniformidad conceptual al considerar todo como objetos.
- Posee un excelente ambiente de ejecución que incluye debuggers, browsers y acceso total a la jerarquía de clases.
- El enlace dinámico permite polimorfismo, y por consiguiente, gran flexibilidad.
- Es muy fácil de entender y afinar el diseño global en comparación con un lenguaje convencional.
- Es muy difícil escribir en estilo no-orientado a objetos, lo que constituye al Smalltalk en una herramienta pedagógica.

*Desventajas:*

- No es muy eficiente y consume muchos recursos de memoria.
- No soporta objetos persistentes
- No maneja en la mayoría de casos ambientes multiusuarios y de control de cambios, sin productos adicionales.
- Los errores en el envío de mensajes son detectados en el momento de ejecución.

### 3.3.3 Object Oriented Cobol

En 1995 se culminó el trabajo del Instituto de Estándares de los EE.UU. (ANSI), el resultado de este trabajo es la especificación W371 del Object

Cobol. Al respecto, las casas de software dedicadas a proveer compiladores de este lenguaje como MicroFocus y HP han lanzado al mercado sus versiones de Cobol orientado a objetos como extensiones al Cobol estándar. De alguna manera el Object Cobol responde a las necesidades de los usuarios que quieren beneficiarse de la reutilización y extensión, así como a la necesidad de proteger su inversión en código existente: una estimación señala alrededor de 70 mil millones de líneas de código en todo el mundo. El Anexo 3 muestra la implementación ANSI del Object Cobol. Las versiones de Object Cobol proveen las siguientes características:

- Clases definidas por el usuario basadas en estructuras de registro Cobol
- Encapsulamiento
- Clases como templates de tipos de datos abstractos
- Envió de mensajes vía el uso de llamadas al verbo USING
- Herencia mediante el verbo COPY
- Polimorfismo usando entry points y la sentencia READ
- Habilidad en usar escritos en otros lenguajes
- Recolección de objetos suprimidos (garbage collection) mediante el verbo CANCEL
- Compatibilidad hacia arriba con el Cobol
- Soporte más avanzado para prototipeo

### 3.3.4 Cuadro Comparativo de algunos lenguajes de la Tecnología de Objetos.

A continuación se presenta un cuadro que muestra algunos lenguajes orientados a objetos y las características de la TO que aplican para cad uno de ellos.

| | *Simula* | *Smalltalk* | *C++* | *Ada* | *Eiffel* | *Object Cobol* |
|---|---|---|---|---|---|---|
| ***Binding*** | *Din/Est* | *Dinámica* | *Din/Est* | *Estática* | *Estática* | *Estática* |
| ***Polimorfismo*** | *Si* | *Si* | *Si* | *Si* | *Si* | *Si* |
| ***Encapsulamiento*** | *Si* | *Si* | *Si* | *Si* | *Si* | *Si* |
| ***Concurrencia*** | *Si* | *Pobre* | *Pobre* | *Dificil* | *Prometido* | *No* |
| ***Herencia*** | *Si* | *Si* | *Si* | *No* | *Si* | *Si* |
| ***Her. Múltiple*** | *No* | *No* | *Si* | *No* | *Si* | *Si* |
| ***Garbage Collect.*** | *Si* | *Si* | *No* | *No* | *Si* | *Si* |
| ***Persistencia*** | *No* | *No* | *No* | *= a 3GL* | *Algo* | *Si* |
| ***Generalidad*** | *No* | *No* | *Si* | *Si* | *Si* | *No* |
| ***Biblio. de Objetos*** | *Pocas* | *Gráficos* | *Pocas* | *Algo* | *Pocas* | *No* |

## 3.4 LA CALIDAD

### 3.4.1 Los conceptos alrededor de la calidad en software

Con el objeto de desarrollar sistemas han surgido a lo largo del tiempo muchos enfoques, siempre con la idea de producir buenos sistemas. Pero, ¿a qué llamamos un buen sistema?. Al respecto existen dos puntos de vista, el interno (del desarrollador) y el externo (del usuario). El usuario desea que el sistema sea rápido, confiable, fácil de aprender y usar, eficaz, etc. Por otro lado, el desarrollador desea que el sistema sea fácil de modificar y crecer, fácil de entender, con componentes reutilizables, fácil de probar, compatible con otros sistemas, portable, potente y fácil de construir.

La definición de lo que debe ser un buen sistema tiene más conceptos que varían en importancia dependiendo de las aplicaciones. En algunos casos es más importante la sencillez de utilización que la performance, etc. Lo que sí es común a todos ellos es que necesita siempre de modificaciones.

De acuerdo a la segunda ley de termodinámica, el desorden de un sistema cerrado no puede ser reducido, sólo puede ser incrementado o

mantenido en su mismo nivel. Una medida de este desorden es la *entropía*. Esta ley parece aplicarse de igual modo a los sistemas de software y es llamada *entropía del software*. Al respecto Lehman, sugiere que:

I. Un programa que es utilizado será en algún momento modificado.

II. Cuando un programa es modificado, su complejidad se incrementa.

Si asumimos que un sistema tiene una cantidad de entropía, la experiencia nos muestra que es razonable asumir que el aumento de entropía del software es proporcional a la entropía del momento de la modificación. De ahí se deduce que es más sencillo modificar un sistema ordenado que uno no ordenado. Eso matemáticamente se expresa como

$$\Delta E \sim E \qquad \text{o,} \quad \frac{dE}{dt} = kE$$

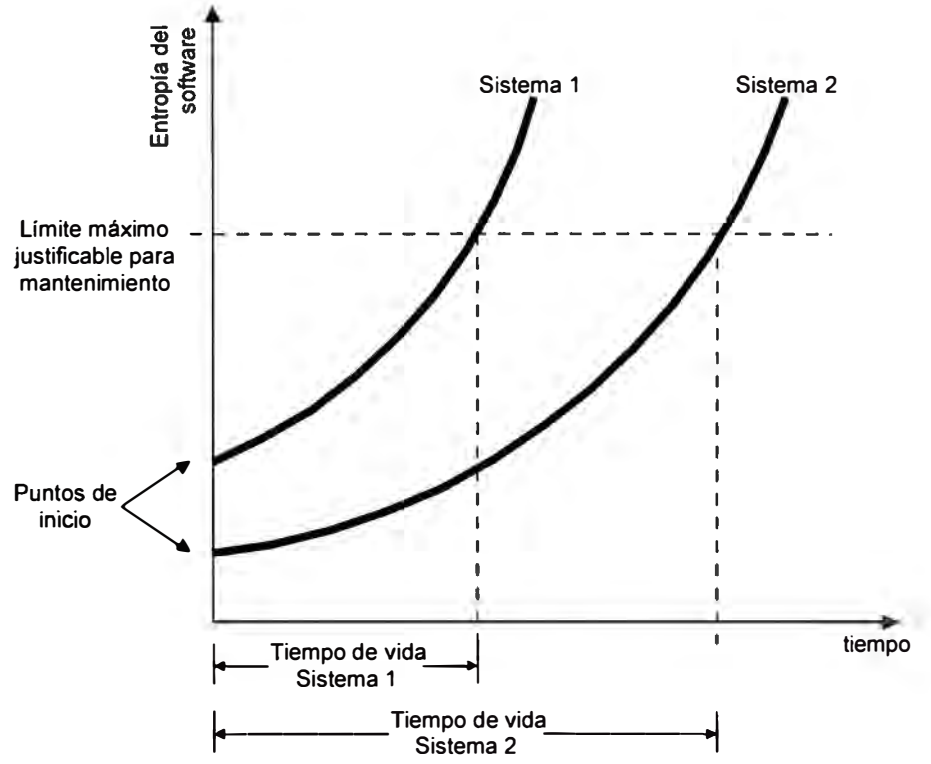



**Figura 3.5** La Entropía de un sistema y cómo se incrementa a diferentes velocidades dependiendo de la entropía inicial

La solución a esta ecuación diferencial se muestra en la figura 3.5. La figura grafica que la prolongación del tiempo de vida de un sistema depende de que tan bien estructurado estuvo inicialmente. Cuando un cierto nivel de entropía es alcanzado, ya no es justificable económicamente continuar con el sistema.

La producción de software está relacionada con la construcción de sistemas de alta calidad con un razonable esfuerzo invertido y por consiguiente con un costo igualmente razonable. Los intentos por incrementar la calidad del software ha pasado por innovaciones en los lenguajes de programación y en los métodos de desarrollo de sistemas. Así comprobamos la evolución de los lenguajes de programación, en lo que se denominan generaciones de lenguajes. Por su parte a nivel de técnicas y métodos hemos visto la implantación de métodos estructurados, herramientas de cuarta generación, herramientas CASE, técnicas de prototipeado, sistemas de bases de datos y generadores de código, todos ellos atacan el problema de la calidad del sistema. Sin embargo, algunas evidencias muestran que en lugar de mejorar la situación, éstas técnicas perjudican el sistema. Así, reportes de la compañía Butler Cox en 1990 mostraba que el diseño estructurado estaba mermando la calidad y productividad del producto final. Lo mismo ocurrió con los lenguajes de cuarta generación que si bien mejoraban la productividad degradaban al mismo tiempo la performance, pues debido a lo restrictivo de los lenguajes de muy alto nivel, era necesario seguir haciendo algunos procesos en lenguajes de tercera generación.

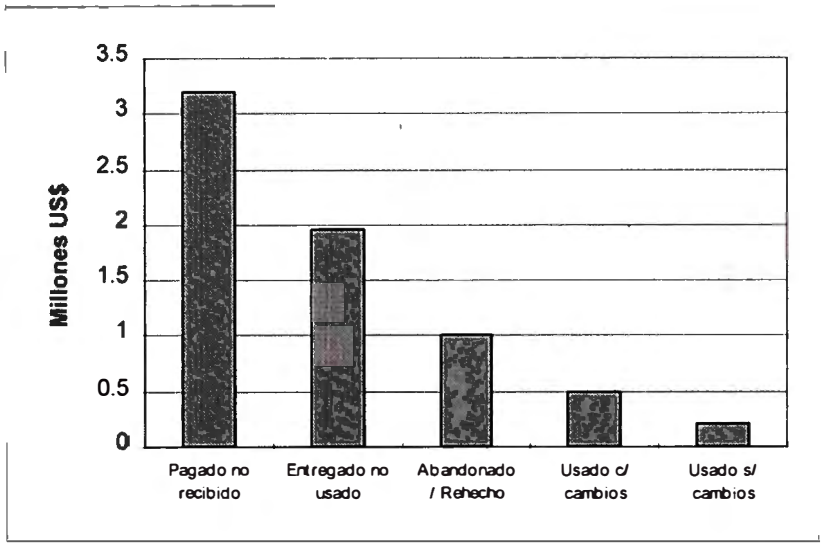



**Figura 3.6** Proyectos del Dpto. de Defensa de los EE.UU.

Una evidencia de las deficiencias que mostraban esta técnicas se puede apreciar en uno de los mas grandes desarrolladores de software, el Departamento de Defensa de los Estados Unidos. (Ver figura 3.6). Este cuadro muestra los proyectos de Defensa en los setentas. Si bien la mayoría de estos sistemas fue hecho en Cobol, y no existe comparación con lo que en los noventas estamos haciendo con la herramientas modernas con las que contamos, el hecho es que algo equivocado se estaba haciendo.

Otro estudio hecho por Lietz y Swanson a principios de los ochenta, se preocupa de averiguar las razones de los requerimientos de mantenimiento a los sistemas. (Ver figura 3.7). El resultado es la comprobación que la mayor parte de los cambios solicitados son inevitables, y reflejan el carácter dinámico de los negocios modernos.

La noción más elemental de calidad que debemos esperar de un producto es la de cumplir con las *especificaciones funcionales*. Esta conformidad funcional es una característica que debe poseer fuera de toda noción de performance o de contexto de ejecución. La naturaleza de la aplicación

puede requerir ciertas cualidades suplementarias. Por ejemplo, reutilización de partes, permitir desarrollo incremental o modular, etc. El contexto de ejecución puede requerir de nuevas características.

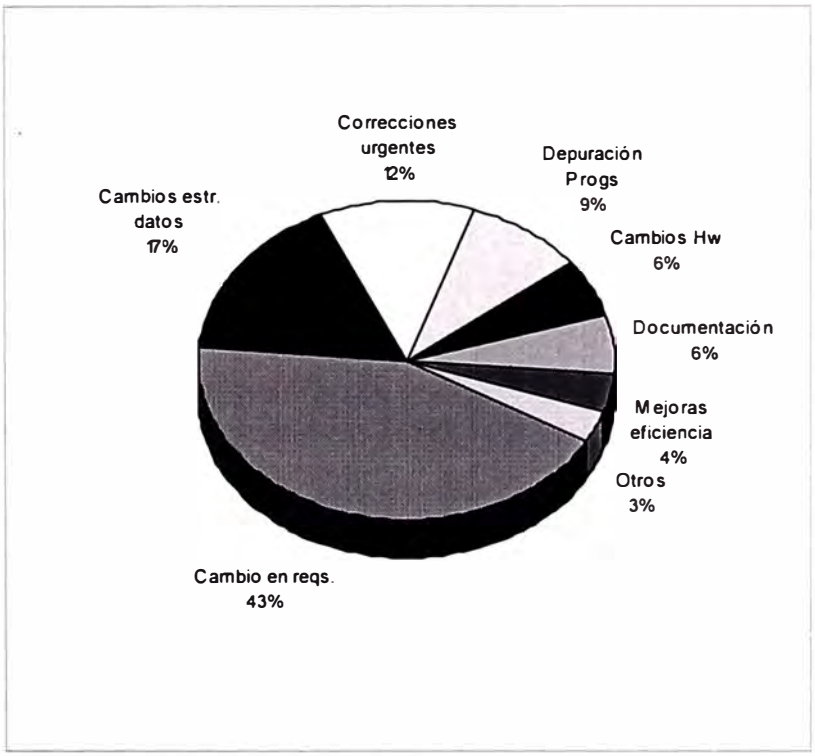



**Figura 3.7** Motivos de modificación de las aplicaciones.

En resumen, la calidad del software se definirá como el respeto a las especificaciones funcionales, así como a la verificación de un cierto número de puntos de calidad propios de la aplicación a desarrollar. Algunos de estos puntos de calidad que se exigirán deberán ser ponderados pues son antagónicos. Por ejemplo, la eficiencia deberá ser relajada o reducida si la portabilidad es exigida.

Anteriormente hemos descrito que los criterios de calidad varían en función del actor dentro del proceso de desarrollo de la aplicación. A continuación se presentan los puntos de vista de calidad de quienes participan del proceso.

El decisor : Aquel que va a comprar o administrar el area involucrada.

El usuario : El que utilizará la aplicación.

El operador : El que se encargará de explotar el aplicativo y supervisar su ejecución.

El mantenedor : El que corregirá los errores residuales y evoluciones.

El desarrollador : El que desarrollará la aplicación, prevee reutilizar algunos componentes.

*El decisor*

Se interesa sobre todo en los problemas del usuario, es decir, portabilidad, eficacia, reusabilidad, fácil de mantener, fácil de explotar, fiabilidad y conformidad funcional.

*El usuario*

Se interesará en el funcionamiento correcto (conformidad y coherencia con el manual de usuario). Se cerciorará que la funcionalidad esté completa. Le interesará que tenga el menor número de fallas (confiabilidad). Buscará mecanismos de seguridad y contingencia (seguridad) y un mínimo impacto en el comportamiento de la aplicación en caso de incidentes (robustez). Deseará que los mensajes sean claros (comprensible) y sugiera acciones a tomar (comunicativo). Los tiempos de respuesta deberán ser razonables (eficiencia y rapidez).

*El operador*

Su principal preocupación será la facilidad de instalación (fácil de explotar) y la facilidad de hacerlo funcionar con otros aplicativos (acoplamiento). Se preocupará además del consumo de recursos (eficiencia) y la protección de acceso a la aplicación (confidencialidad e integridad).

*El mantenedor*

La necesidad de mantenimiento se centra en la facilidad de localizar y corregir fallas (mantenibilidad) así como la facilidad de modificación o mejorar (flexibilidad y extensibilidad) y la facilidad de poner a punto los cambios.

Las cualidades de la aplicación se resumen en nueve características. Dichas características son reactivas o proactivas entre sí. A continuación se muestra un cuadro con las interacciones entre ellas.

Adaptable: ⇒ flexible ⇒ extensible, modificable
Seguro: ⇒ protección ⇒ observabilidad
Conforme Funcional: ⇒ coherencia, completamiento
Eficiencia: ⇒ conciso
Integración: ⇒ reutilisable, acoplable y comunicable
Mantenible: ⇒ fácil de probar, modificable ⇒ comprensible, coherente
Maniobrable: ⇒ expresivo y explotable ⇒ comprensible y coherente
Portable: ⇒ reutilizable
Robusto: ⇒ confiable, seguro ⇒ protegido

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 Adaptable | 1 | | | | | | | |
| 2 Seguro | × | 2 | | | | | | |
| 3 Conformidad Funcional | ✓ | 0 | 3 | | | | | |
| 4 Eficiencia | × | × | × | 4 | | | | |
| 5 Integración | ✓ | × | ✓ | × | 5 | | | |
| 6 Mantenible | ✓ | × | ✓ | × | ✓ | 6 | | |
| 7 Maniobrable | ✓ | × | ✓ | × | ✓ | ✓ | 7 | |
| 8 Portable | ✓ | × | ✓ | × | ✓ | ✓ | ✓ | 8 |
| 9 Robusto | × | ✓ | ✓ | × | ✓ | × | ✓ | ✓ | 9 |

donde :

✓ : correlación fuerte

✗ : antagonismo

0 : sin interinfluencia

Algunos criterios de evaluación de la calidad de software se explican a continuación:

- Cuatro criterios cuantitativos concernientes a la interfaz
- Tres fallas dentro de la topología de dependencias
- Tres fallas en el flujo de control
- Dos defectos en el flujo de datos

### 3.4.2 Criterios cuantitativos concernientes a la interfaz

La interfaz a evaluar contiene los siguientes componentes:

- Los tipos: T tipos (el tipo de interés y T-1 tipos auxiliares, excepto los parámetros genéricos que forman parte del tipo principal)
- Las funciones: F funciones, divididas en S selectoras y M modificadoras (al menos una constructora)
- Las precondiciones: P precondiciones formales
- Los axiomas: A axiomas

Los criterios propuestos son los indicadores siguientes:

Operatividad: $\frac{F}{S}$

Adaptabilidad: $\frac{F}{P}$

Cohesión: xxxx

Abstracción: $\frac{\sum_{A}(numero\_funciones\_en\_composicion)}{F}$

A continuación los explicaremos:

*Operatividad*: ] 1,0 (N!) [

Representa la facilidad de manipular los atributos y estados de cada tipo o clase de interés. Una clase abstracta posee al menos un constructor y N selectores (donde N es el número de atributos). En este caso $\frac{F}{S}$ vale, como mínimo, $\frac{(N+1)}{N}$.

Para una clase provista de numerosos modificadores, $\frac{F}{S}$ puede devenir superior a 2, pero no sobrepasará 3; en teoría, el número de modificadores puede ser del orden de N!, si consideramos todas las combinaciones posibles de atributos.

*Adaptabilidad* : [ 1, ∞ [

Muestra el nivel de adaptabilidad del tipo abstracto a nuevas situaciones o contextos, está en directa relación con la cantidad de premisas, precondiciones y restricciones para que el tipo abstracto se desenvuelva. Un tipo abstracto sin precondiciones (es decir, sin restricciones de encadenamiento o condiciones-limite en los atributos), tiene un indicador de adaptabilidad infinita.

Un tipo provisto con muchas restricciones operativas puede tener una precondición por función, de tal manera que su indicador de adaptabilidad sea un valor de 1.

*Cohesión*: [1, 0 (N!) [

Este indicador muestra el nivel de interrelación de las funciones a partir de axiomas que los rigen. Está construído para otorgar a los axiomas (que describen las relaciones entre las funciones), un peso superior que aquellos que sólo sirven para establecer equivalencias de notación para las funciones redundantes.

*Abstracción*: ] 1, 0 (N!) [

Muestra el nivel de abstracción del tipo abstracto.

### 3.4.3 Defectos de la topología de dependencias.

Los tres defectos a los que hacemos mención se muestran en la figura 3.8 que a continuación presentamos.

Fenómeno fan-in : consiste en la existencia de un módulo con numerosas dependencias. En este caso es conveniente desarrollar la fragmentación horizontal.

Fenómeno fan-out : consiste en la existencia de una secuencia de módulos dependientes los unos de los otros a modo de cuerda con nudos. En estos casos estamos ante niveles de abstracción no significativos y generación de dependencias y módulos innecesarios, donde es necesario un achatamiento de las estructuras de jeraquización.

Fenómeno Spaghetti : consiste en la existencia de una secuencia de módulos dependientes los unos de los otros a modo de *cuerda con nudos*. En estos casos estamos ante niveles de abstracción no significativos y generación de dependencias y módulos innecesarios, donde es necesario un achatamiento de las estructuras de jerarquización.

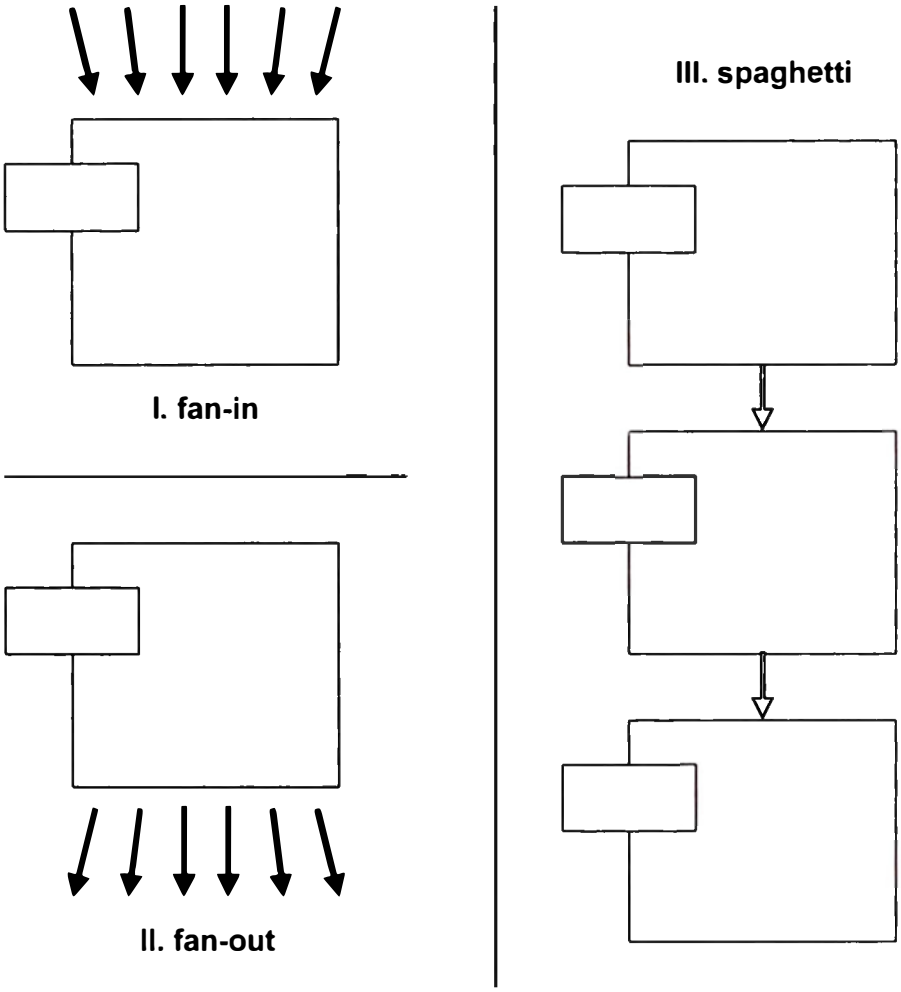



**Figura 3.8** Defectos de diseño dentro de las topologías de dependencia.

### 3.4.4 Fallas dentro del flujo de control.

Los problemas en los flujos de control se producen por una de las siguientes causas:

i. Lanzamiento y/o parada de procesos no especificado
ii. Lanzamiento y/o parada de procesos redundante
iii. Demasiados parámetros en la inicialización

La primera falla respecto a los lanzamientos y paradas de proceso fuera del módulo, es necesario señalar, que un módulo debe contener un mínimo de entradas de lanzamiento o parada, para evitar confusión y garantizar trazabilidad.

De otro lado, la descentralización debe ser jerarquizada, no sólo es necesario conocer el punto de partida y llegada de un flujo, sino también conocer qué otros flujos involucra, cómo activar dichos flujos, etc. Por ejemplo, un objeto O1 envía un control a un objeto O2 y el otro control a O3; adicionalmente O2 envía un control a O3; dentro de este caso, la ambigüedad de saber de quién envió la orden sobre O3, resulta un error de diseño. En este caso, sería conveniente definir dos controles diferentes para O3 y no utilizar el mismo.

El tercer caso resulta de un análisis insuficiente producido de la creación de una tarea confusa; es en general bastante fácil simplificar o sino descomponer para clarificar.

### 3.4.4 Defectos del flujo de datos

Podemos señalar los siguientes defectos relativos a los flujos de datos complejos:

- Demasiados parámetros
- Parametraje repetitivo

Por un lado se debe invertir la perspectiva, es decir fusionar objetos en uno solo agrupando operaciones diseminadas pero utilizadas de manera complementaria. De otro lado, buscar reagrupar los parámetros en abstracciones significativas puede resultar una buena solución.

## 3.5 LA ADMINISTRACIÓN DEL PROYECTO.

Los proyectos de desarrollo como cualquier proyecto requiere de personas encargadas de hacerle seguimiento y control. De otro lado, por ser el jefe de proyecto el interlocutor con los directivos de la organización, es muy importante tener claro el alcance y transmitirlo a los niveles superiores para evitar falsas expectativas. Este proceso de control va de la mano con el método a utilizar, de

modo que se pueda identificar los puntos de corte para el control de las actividades. A continuación se propone un esquema de control del proyecto que concibe cinco fases:

El plan de negocio

- ✧ El estudio de factibilidad y diseño de alcance
- ✧ La elaboración del plan
- ✧ La ejecución del plan
- ✧ El cierre del proyecto

A continuación se explica cada una de estas fases.

### 3.5.1 El plan de negocios

El primer paso en el desarrollo de un sistema es determinar mediante un pre-estudio las características del proyecto en función a objetivos, recursos, costos, alcance inicial y plazos. La elaboración de un documento de este tipo nos permitirá elevar la propuesta o responder a los niveles directivos sobre las características del proyecto. Esta es una estimación muy gruesa que permitirá determinar de qué orden de magnitud estamos hablando.

### 3.5.2 El estudio de factibilidad

En esta fase se hace un levantamiento de información más detallada. Se entrevista a los solicitantes o usuarios, se determina con ellos sus requerimientos, y se define conjuntamente el alcance en función a los recursos con que disponemos y que han sido sujetos de aprobación gruesa en la etapa anterior. En esta etapa lo más importante es definir claramente el alcance del proyecto. Este alcance tiene dos partes: el alcance funcional, es decir, las características que el usuario percibe fácilmente; y la otra parte es el alcance técnico que tiene que ver con plataforma de hardware y software que lo soportan, periféricos especializados, equipamiento adicional, etc.

Respecto a estos dos alcances, es indudable que el alcance funcional debe ser cualitativamente superior al alcance técnico. Pues de lo que se trata, por lo menos en la mayoría de los casos, es que las exigencias usuarias estén siempre como primera prioridad, y no la innovación tecnológica.

### 3.5.3 La elaboración del plan de trabajo

La elaboración del plan de trabajo es una tarea muy importante, pues plasma de alguna manera la fusión de la necesidad de administrar el desarrollo y las características de la metodología utilizada. Adicionalmente, implica un manejo de recursos que debe buscar *colocar al recurso correcto en el momento correcto*, de modo que se minimice los costos. Se propone que la asignación de recursos debe ser anónima, el recurso asignado será tipificado por función y no será nominal. Así, para la actividad A tendremos el recurso Constructor de Clases, y no José López. Esto facilita la tarea de reasignación de actividades a otros recursos.

### 3.5.4 La ejecución del plan de trabajo

La ejecución del plan involucra un conjunto de actividades a realizar durante la duración de esta fase.

- Entrega periódica de tareas a realizar a cada miembro del equipo
- Definición de puntos de corte (milestones) para la verificación de avance
- Ajuste del plan de trabajo
- Reportes periódicos a los niveles superiores
- Reuniones de ajuste de actividades
- Aseguramiento de la calidad de acuerdo al alcance técnico y funcional
- Registro del esfuerzo efectivo de cada miembro del equipo.

Una vez elaborado el plan de trabajo es necesario reunirse con todo el equipo y discutirlo de modo que cada uno de ellos haga suyo el proyecto y sea consciente del proyecto de manera global. El uso de empowerment hacia algunos miembros del equipo es muy útil pues evita que el jefe de proyecto tenga que resolver todos los problemas.

Cada miembro del equipo sabe en todo momento qué tiene programado para hacer en los siguientes 15 días. De esta manera dosifica sus energías y ante problemas surgidos durante la ejecución de alguna actividad, se encargará de alertar las posibles implicancias en el cronograma. De esta manera se asegura tomar las medidas correctivas lo antes posible.

Es conveniente hacer revisiones del plan de trabajo cada cierto tiempo. Al respecto se recomienda que para proyectos de más de un año los cortes sean mensuales. Si son proyectos de seis a diez meses es mejor hacer cortes quincenales. Estos cortes de evaluación del avance no perjudican la posibilidad de reprogramaciones de emergencia en cualquier momento. En estos cortes de evaluación se determina el porcentaje de avance respecto a lo programado, porcentaje de evaluación del esfuerzo respecto a lo programado y porcentaje de esfuerzo que pertenece a actividades no estimadas. Estos indicadores permitirán más adelante tener valores de referencia y permitirán identificar estimaciones no acertadas. El reporte producto de esta actividad deberá explicar el motivo de los atrasos, las actividades realizadas en el periodo y los factores de riesgo que se avisoran para las siguientes etapas del proyecto.

Con relación a las desviaciones en las estimaciones, podemos identificar algunas causas:

- Una mala estimación

- El recurso con el que se ejecutó la actividad no es de la misma productividad o conocimiento si lo comparamos con el recurso asignado inicialmente a dicha actividad.
- Existieron elementos distorcionantes que prolongaron la actividad. Por ejemplo: fallas en el software de base, lenguaje o ambiente de programación, equipamiento de debajo de las especificaciones iniciales, cambio de recurso durante la ejecución de la actividad, etc.
- Especificaciones imprecisas para la actividad

### 3.5.5. Cierre del proyecto

Esta es quizás la fase más importante para el departamento de informática o la empresa de software desde el punto de vista de calidad e ingeniería de software. En esta etapa se elabora el informe de fin de proyecto.

El reporte de fin de proyecto debe ser un documento que recoja un resumen de cómo se llevó a cabo el proyecto, un reporte de los incidentes que se produjeron. De alguna manera representa la bitácora del proyecto.

A continuación se propone algunos puntos que debe tener dicho documento:

- Resumen del proyecto en duración, esfuerzo y costo (real y estimado)
- Resumen de la ejecución del proyecto (bitácora)
- Problemas más importantes que se presentaron
- Recomendaciones para futuros proyectos
- Ratios de productividad por actividad, si fuera posible

## 3.6 EL PROCESO DE TRANSICIÓN DE LA EMPRESA HACIA LA TECNOLOGIA DE OBJETOS.

A este respecto quisiera señalar que existen dos tipos de empresas: las empresas de consultoría y desarrollo de software, y las empresas dedicadas a otras actividades. Es muy importante hacer esta distinción, pues en el mercado peruano las primeras son empresas pequeñas (en volumen de ventas, personal y capacidad de inversión), mientras que para las segundas, un proceso de modernización de sus sistemas de información sólo se justifica si el beneficio que se obtendrá supera los niveles de satisfacción actuales. No debemos olvidar, y éste es un gran error que cometemos, que una empresa textil tiene por objetivo producir y vender textiles y no contar con la tecnología de punta en su centro de cómputo o departamento de sistemas, si ésta no aporta beneficios adicionales.

### 3.6.1 La Empresa en general

El proceso de migración hacia estas tecnologías requiere de inversiones en personal del departamento de informática. Por eso considero que la Empresa a la cual nos referimos, debiera contar por lo menos con 15 a 20 profesionales en informática.

Este es el caso más cercano de algunas experiencias de migración hacia la TO que se publican en revistas y otros documentos. A continuación señalo algunas recomendaciones que aplican a nuestro medio:

- *Usar sólo aquella parte de la tecnología que se encuentra madura.* La tecnología de objetos ofrece en la actualidad una gran gama de productos y técnicas. Se debe evitar utilizar aquellas que no estén lo suficientemente maduras o no se alinean con los estándares que en la actualidad existen (Microsoft, OMG). Utilizar metodologías ya maduras como Wirfs-Brock, Booch, Jacobson, Rumbaugh o Yourdon. Utilizar herramientas de programación como el C++,

Eiffel o Smaltalk o ambientes de desarrollo más elaborados como el PARTS, VisualAge o Envy.

La utilización de bases de datos orientadas a objetos es un tema bastante delicado. De un lado tenemos el problema de la convivencia con otros SGBD relacionales de la empresa y sobre todo, la relación costo/beneficio del uso de dichos SGBD sigue siendo inferior a la unidad. La opción de utilizar SGBDOO puede ser justificada si trabajamos con estructuras de datos muy complejas y donde la solución relacional no es satisfactoria.

Sobre los generadores de código y herramientas CASE existe aún mucho por andar. Desde mi punto de vista, el análisis y diseño aún siguen siendo etapas del desarrollo muy racionales y de gran complejidad. Una herramienta de soporte al diseño de modelos y documentación sí considero que pueda ser útil. Pero lo que nunca debemos hacer es comprar un U-CASE como herramienta de dibujo.

- *Utilizar la tecnología de objetos en todas las fases del proceso*. Es muy importante llevar a cabo el ciclo completo utilizando esta metodología. Algunos programadores se entregan entusiastas a desarrollos utilizando herramientas como el C++, creyendo que de esta manera ya está trabajando con esta tecnología. Considero que lo más rescatable de la TO es el proceso análisis y diseño, la etapa de programación podría ser utilizando herramientas o lenguajes no OO.

  Por otro lado, y como se viene recomendando desde hace mucho, debemos tomarnos todo el tiempo necesario para realizar un buen análisis y diseño.

- *Escoger un proyecto que merezca el uso de la tecnología.* Al respecto, en alguna de las publicaciones sobre experiencias se hablaba de tomar un proyecto "lo suficientemente pequeño como

para que los gerentes no se enteren". Considero que ésta es una decisión equivocada, pues es necesario seleccionar un proyecto donde las bondades de la TO se muestren en toda su magnitud.

- *Debe haber un compromiso y conciencia de la empresa para desarrollar la estrategia de migración.* Esto evitará la búsqueda jacobina de chivos expiatorios a la hora de los problemas, si estos se presentan.

- *El proceso de transferencia de tecnología está a cargo de un grupo de promotores.* Es necesario asignar tareas de estudio e investigación a un grupo de personal, para que luego sean ellos quienes conduzcan el desarrollo de los proyectos iniciales. Dentro de este grupo encargado de realizar el mentoring deben incluirse elementos de un buen nivel y con ascendiente entre el personal de informática, para que este proceso de difusión sea natural y fácil.

- *Se debe utilizar entrenamiento en función a las necesidades ("just in time" training).* Es mucho más productivo capacitar y aplicar dicho entrenamiento de inmediato, para poder aplicar de manera efectiva los conocimientos y técnicas adquiridas.

- *Tratar de identificar mejoras en el proceso a través de la utilización de métricas.* Esto, que es muy difícil, aún en los desarrollos tradicionales, requiere iniciar dichas métricas en los proyectos actuales, de manera que tengamos puntos de comparación y argumentos para cambiar a la TO. Sin pruebas concretas de la mejora y ventajas de esta tecnología no podremos justificar los indudables gastos adicionales de esta metodología.

### 3.6.2 La Empresa de Consultoría y Desarrollo de Sistemas

El caso de este tipo de empresa es completamente diferente. En el Perú hablamos de empresas de no más de 20 a 30 profesionales, incluso menos personal. Los proyectos que desarrollan generalmente son a pedido, existen pocas empresas que se dediquen exclusivamente a la venta de sus productos de software. Para estas empresas el cambio hacia un paradigma de orientación a objetos es mucho más difícil, pero no imposible. A continuación presentamos algunos lineamientos para una migración menos traumática. Lo que a continuación se plantea incluye algunos aspectos antes mencionados que perfectamente se adecuan a la realidad de una empresa de consultoría y desarrollo de software.

- *La gerencia debe estar convencida que el proceso de asimilación de esta tecnología es importante para la empresa.* La promoción de la TO debe ser asumida por el nivel directivo, del mismo modo que el control de los planes debe ser vigilado para que se lleven a cabo dentro de los plazos establecidos. No debemos permitir que una iniciativa de este tipo tenga un apoyo sin objeciones de la gerencia, pues esto significaría en la mayoría de los casos que no se está tomando conciencia de lo que se trata. Por otro lado, tampoco se debe permitir ejecutar el plan de manera extraoficial. Este proceso de transferencia de tecnología tiene sus costos y esfuerzos, los mismos que deben aparecer en los planes y presupuestos.

- *La formación de un equipo mínimo de dos personas para ser los receptores iniciales de la tecnología.* El equipo debe estar formado por profesionales de buen nivel técnico pero que al mismo tiempo posean una adecuada capacidad de comunicación, para que puedan ser luego los transmisores de sus conocimientos y puedan

compartir con los otros miembros del equipo todas sus experiencias y conclusiones.

- *La selección del primer proyecto*. El primer proyecto debe tener dos características principales: ser apropiado a la TO y no ser de carácter crítico. Ser apropiado a la TO se refiere a su capacidad de ofrecer ventajas comparativas respecto a un desarrollo tradicional. No ser de carácter crítico es un requisito básico, pues, no debemos añadir riesgo a un proyecto estratégico o de plazos muy cortos. Es conveniente que la selección del proyecto piloto involucre a la gerencia al momento de la selección. A nivel de desarrollo debe conformarse un equipo que incluya los miembros del equipo receptor, los cuales serán los catalizadores del proceso de aprendizaje.

- *Ir incorporando conceptos de la TO en el proceso de desarrollo de aplicaciones*. Debemos tratar de ir entrenándonos en el camino con los proyectos de desarrollo procedural, organizar pequeñas reuniones para establecer y modelos con TO de nuestros desarrollos tradicionales. Otro aspecto importante es iniciarse en el desarrollo de aplicaciones con interfaz gráfica, generalmente las herramientas que permiten desarrollar con esta tecnología tiene muchos conceptos de la TO incorporados, y así, nos vamos acostumbrando a utilizar componentes encapsulados (cajas negras) e interfaz gráfica.

- *Seleccionar el lenguaje y las herramientas que produzcan el menor impacto en el personal técnico*. En nuestro medio el uso del Lenguaje C como lenguaje de programación es bastante limitado. Sin embargo para aquellos que han trabajado con él, lo más natural

es pasar al C++ o herramientas como el Visual C++ o el Visual Age for C++. En el caso de personal que ha desarrollado con otros lenguajes más clásicos, considero que no existe mucho para escoger. El Object Cobol ya es una realidad y está definido como una ampliación al ANSI Cobol. Si bien es una posibilidad, que muchos calificarían como "jalada de los cabellos", implica un menor tiempo de entrenamiento y adaptación. Sin embargo, si deseamos realizar un cambio radical y contamos con los recursos para hacerlo, entornos lenguajes o entornos basados en Smaltalk o Eiffel son posibilidades más puristas para el desarrollo.

- *Verificar los resultados obtenidos*. Al igual que en caso de la gran empresa, la evaluación de los beneficios obtenidos es muy importante para que los proyectos siguientes adopten esta tecnología como fundamento.

### 3.6.3 Flujo de acciones para la transición hacia la tecnología de objetos

A continuación se propone un flujograma de actividades para realizar una transición "sin traumas" hacia la metodología de objetos. Como podemos apreciar en la figura 3.9, los pasos a seguir en este proceso de transición, requieren en primera instancia de una evaluación de la madurez del área de sistemas o empresas de software. La madurez en el proceso de desarrollo ha sido tipificado por el Instituto de Ingeniería de Software, en cinco niveles:

*Nivel Inicial:* No existen métodos documentados que se utilicen en el desarrollo. Cada desarrollador lo hace a su modo.

*Nivel Repetitivo:* Existe un método pero éste no ha sido formalizado. Sin embargo existe un consenso respecto de cómo hacer las cosas.

*Nivel de Definición:* Existe un proceso documentado y formal de desarrollo de software. Existe un proceso continuo de refinamiento.

*Nivel de Gestión:* Existen mediciones formales de las diferentes características del proceso, las mismas que son continuamente tomadas. No sólo se evalúa en función a costo y tiempo, sino también en función de productividad, eficiencia, calidad, etc.

*Nivel de Optimización:* Las mediciones del nivel anterior son permanentemente usadas como realimentación para optimizar el proceso.

Al respecto, las observaciones de los niveles de madurez en EE.UU. reflejan que a fines de los 80 sólo el 3% de la empresas se encontraban en el nivel 3 (de Definición). Se considera que el nivel mínimo de una organización para realizar una transición hacia la TO es el nivel repetitivo. Esta afirmación se basa en la necesidad una mecánica uniforme de trabajo en el desarrollo de aplicaciones. La utilización de estándares y procedimientos, auque éstos sólo sean de facto, nos indican la existencia de un disciplina, de un orden y formalidad mínimos en el desarrollo de aplicaciones. La adopción de la TO por sus características mismas, abre las puertas a la entropía. Esto se debe a que construímos componentes (objetos) reusables, extendibles y que son utilizados, modificados y reestructurados permanentemente por el total del equipo de desarrollo.

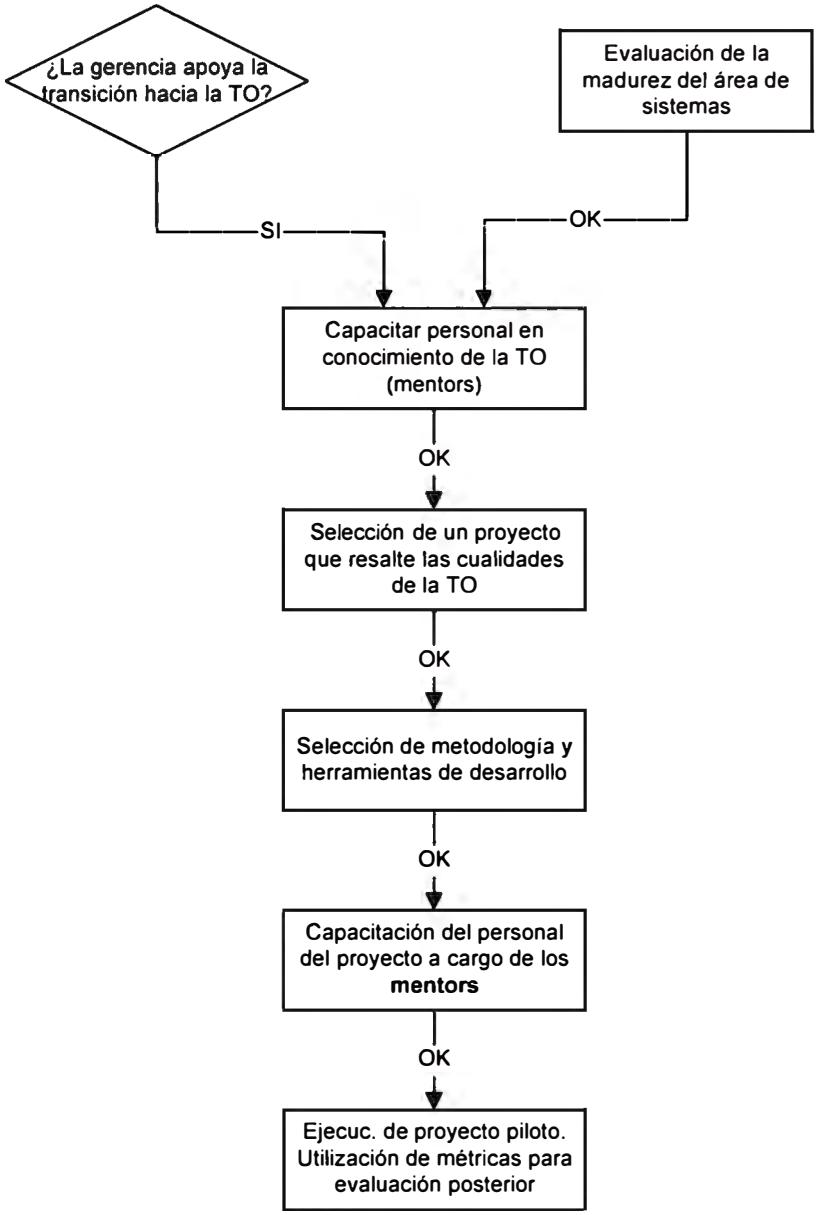



**Figura 3.9** Flujo de transición hacia la TO

Un paso paralelo a la evaluación de la madurez de la organización de sistemas en el desarrollo de aplicaciones es conseguir la decisión de los miembros directivos de apoyar el proceso. Esto que ha sido reiterativo en las recomendaciones para la empresa en general y aquellas de software, es importante pues la adopción de esta tecnología implica incurrir en costos, los mismos que no serán recuperados en un corto plazo. La capacitación del personal propulsor de dicha metodología, que en inglés se denominan *mentors*, previa selección entre los miembros del equipo

con mayor ascendiente y nivel técnico, es la siguiente acción a tomar. Luego ellos serán los auspiciadores e impulsadores de esta tecnología. La siguiente acción a ejecutar en una de las más delicadas, consiste en la selección del proyecto piloto. Más adelante nos referimos en detalle a este punto. La selección de las herramientas de desarrollo fue tratada en secciones anteriores de éste capítulo. Al respecto resumiendo podemos señalar, que lo más importante es elimminar los riesgos que implican la utilización de herramientas completamente nuevas para nuestro equipo de desarrollo. Es vital aprovechar las habilidades y conocimiento de nuestro equipo. La selección de la metodología de desarrollo pasa por una evaluación de aquellas que conoce nuestro equipo de trabajo y que se adapten a nuestra aplicación. El desarrollo de aplicaciones es un proceso que debe ser ecléctico, tomando lo mejor de lo que nos ofrece la TO. Como se mencionó lineas arriba, la capacitación *just-in-time* es lo más conveniente. Los mentors deben participar con el equipo del proyecto en esta labor. La ejecución del proyecto debe contemplar el seguimiento de actividades y desarrollo de métricas de las diferentes fases y actividades, para su posterior evaluación. Se ha diseñado un flujoespecial para la selección del proyecto piloto, el mismo que se presenta en la figura 3.10.

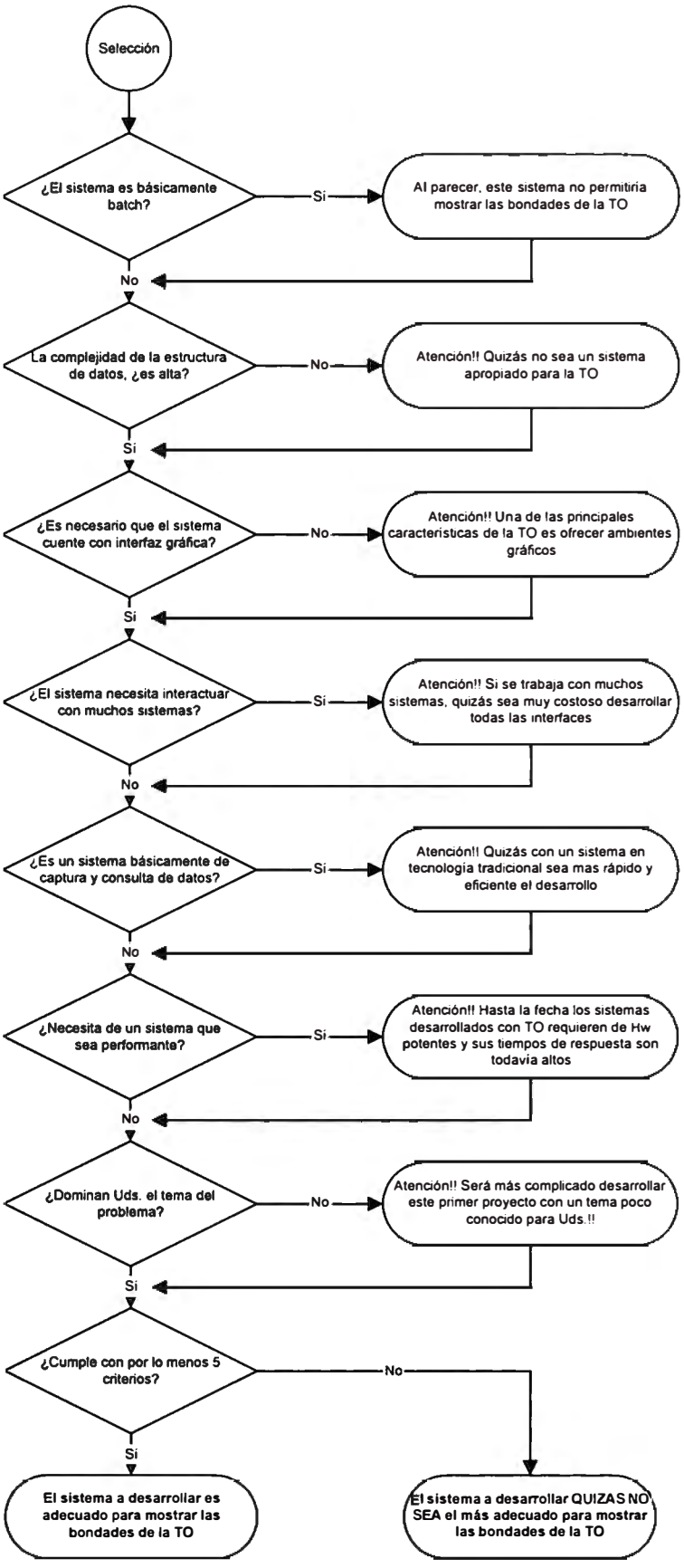



**Figura 3.10** Selección de un proyecto con TO

En el flujo de la página anterior se muestran algunas ideas para la evaluación de la conveniencia de una aplicación para que ésta sirva como proyecto piloto.

El primer paso es descartar casi de plano la posibilidad de utilizar como proyecto piloto el desarrollo de una aplicación batch. Las consideraciones son obvias. En primer lugar dentro de las ventajas que proveen las herramientas para la TO tenemos la facilidad para la creación de interfaces usuario, que en este caso serían desaprovechadas. De otro lado, el esquema de procesamiento en lotes implica una creación y destrucción de instancias de manera masiva, lo que involucra utilizar una eficienta herramienta de TO que se encargue del *garbage collection* o capacidad de memoria considerable.

Otro punto importante a tener en cuenta es la complejidad del aplicativo. Para esto se muestra en la figura 3.10a el tipo de aplicaciones que son más convenientes para una implementación con tecnología de objetos. La figura muestra dos ejes que vienen a ser características de las aplicaciones. Por un lado la complejidad, que es uno de los argumentos mas fuertes que esgrime Booch al momento de fundamentar la idea de una nueva manera de hacer aplicaciones. Por otro lado el uso de la aplicación, es decir si se trata de aplicaciones personales, de grupos de trabajo, departamentales o empresariales. El área con el símbolo de OK es el área de las aplicaciones de nuestro interés.

La facilidad para el desarrollo de interfaz gráfica, como hemos mencionado lineas arriba, es una de las principales vetas a explotar en los sistemas de TO. Es por esto que los proyectos pilotos deberían aprovecharla. Dado que la aplicación a desarrollar representa un esfuerzo piloto para iniciar nuestro camino en la tecnología de los objetos es

conveniente buscar una aplicación que posea el mínimo de interfaces con otros aplicativos. El motivo es evidente, es muy probable que si tenemos que desarrollar interfaces con otros aplicativos el objetivo del proyecto se diluya en esfuerzos colaterales de integración.

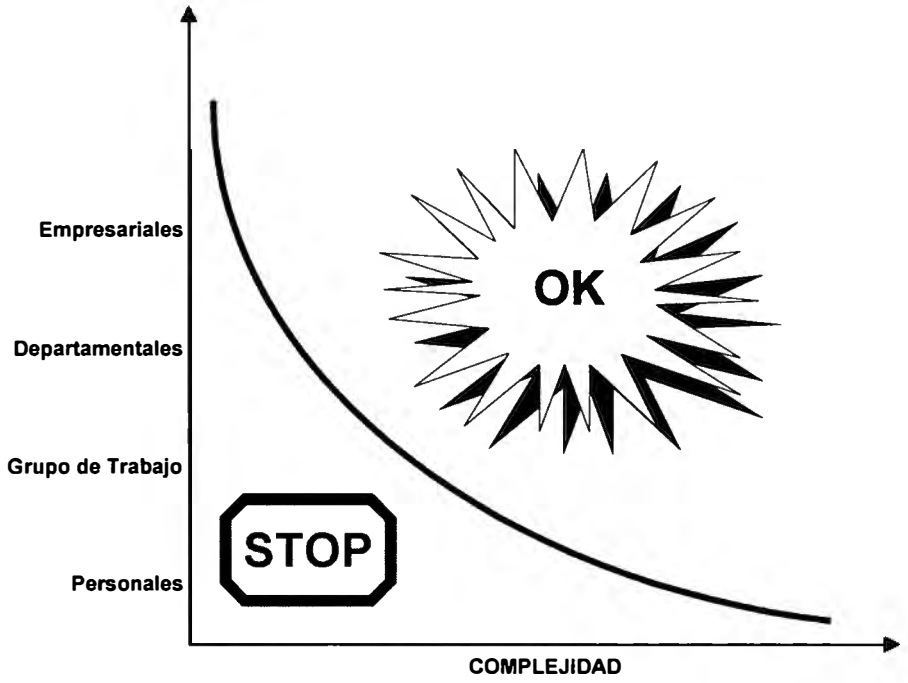



**Figura 3.10a.** Tipos de aplicación para la TO

Sin embargo, si nuestro proyecto piloto se refiere a un aplicativo con interfaces, es conveniente utilizar herramientas de TO que permitan la integración. Por ejemplo, VisualAge es una herramienta que permite una amplia gama de interfaces y protocolos de comunicación.

Uno de los grandes problemas, por el momento, de las aplicaciones con TO es su tiempo de respuesta. No es una tecnología para aplicaciones de misión crítica, por lo que una aplicación de alta performance podría desmerecer las bondades de esta tecnología.

## 3.7 LA MIGRACIÓN DEL LEGACY SOFTWARE A LA TECNOLOGÍA DE OBJETOS

### 3.7.1 El legacy software o infraestructura de software instalada

En estos últimos tiempos se ha venido a denominar **legacy software** a todo ese gran activo que tenemos en nuestras instituciones en forma de programas y datos organizados y estructurados usando las técnicas clásicas de desarrollos de sistemas de información. Estas técnicas se refieren a la programación (programación estructurada), las técnicas de análisis y diseño (deMarco, Yourdan, Martin), e inclusive a la técnica de manejo de datos (archivos indexados, bases de datos jerárquicas, bases de datos reticulares, bases de datos relacionales o bases de conocimiento).

El problema que afrontan las empresas que desean cambiar a la tecnología de objetos consiste en qué hacer con la gran inversión ya efectuada en este tipo de aplicaciones. Es por esta razón que se debe buscar una *interoperación* de la tecnología antigua o clásica con la tecnología innovadora o de objetos. Generalmente, la primera etapa de esta extensión de los sistemas tradicionales consiste en proveerlos de interfaces GUI, algo que la TO hace muy bien. Además de las interfaces GUI, las áreas naturales para la interoperación de componentes clásicos y OO son la reusabilidad, la migración, la extensión y la construcción de front-ends gráficos y/o inteligentes. Debido a que nuestra migración no podrá ser (generalmente) a gran escala por falta de recursos económicos y humanos, el camino a seguir es construir sobre lo existente y migrar paulatinamente hacia la tecnología de objetos.

A continuación mencionamos algunas razones por las que la TO debe interoperar con sistemas no OO:

- Existen sistemas muy grandes y complejos que no pueden ser reescritos de una sola vez.
- La migración hacia la TO será evolutiva para los sistemas actuales, lo que implica que las aplicaciones no serán completamente cambiadas sino que ciertos componentes evolucionarán hacia la TO de manera progresiva.
- Dentro de nuestras aplicaciones contamos con algoritmos altamente optimizados o especializados que será necesario reutilizarlos en los sistemas con TO.
- La explotación de las bases de datos relacionales corporativas que deberán ser compartidas por las aplicaciones antiguas y las aplicaciones nuevas. Pues no es imprescindible usar Manejadores de Base de Datos Orientados a Objetos en los nuevos sistemas.
- Puede existir casos en los cuales la necesidad se limita a proporcionar interfaz gráfica a los sistemas existentes (front-ends gráficos).
- La necesidad de construir sobre las soluciones empaquetadas ya existentes, tales como paquetes adquiridos a terceros.
- Los sistemas expertos embebidos dentro de nuestras aplicaciones cuyo desarrollo con TO es innecesario.

En SARA BANK vemos un ejemplo clásico de necesidad de convivencia de ambas tecnologías. Los componentes de SARA BANK trabajan con información que se encuentra almacenada en archivos tradicionales (tipo VSAM) que son manejados por el software de base LANDP, el mismo que sirve como soporte a las comunicaciones con el computador central. En la actualidad, si quisiéramos pasar a tecnología de objetos, no podríamos dejar de usar este software de base que nos permite trabajar el manejo de archivos y comunicaciones. Es necesario convivir con él. Por otro lado, mucha de la lógica de negocio que contiene un producto como

SARA BANK, orientado a la banca, involucra algoritmos especializados de matemática financiera que sería innecesario rehacer. SARA BRANCH es el componente que debe aprovechar al máximo los beneficios de la TO. Esta afirmación se sustenta en lo siguiente:

- Trata de información en función a parámetros internos (esquema de transacciones) y externos (ingresados por el usuario).
- Maneja componentes (objetos) como el monitor, las transacciones, funciones de negocio, funciones de interfaz, funciones de acceso de datos, funciones de comunicaciones, los esquemas, las cuentas, las cajas, todos ellos imbuídos de comportamiento, identidad, estructura y gran dinámica, características esenciales de los objetos.

### 3.7.2 Los wrappers o envolturas

Los problemas que se mencionaron anteriormente dan como resultado la necesidad de componentes de software que permitan interactuar la parte clásica y la OO de nuestros sistemas. Es aquí que surge el concepto de **Wrapper**. Un wrapper puede ser usado para migrar a una tecnología orientada a objetos, protegiendo la inversión en el código tradicional. El concepto de wrapper consiste en la comunicación mediante el envío de mensajes de la parte orientada a objetos con el módulo convencional (convertido en un objeto por el wrapper). El wrapper es desarrollado con el lenguaje tradicional de programación del antiguo sistema, y representa un esfuerzo inicial, pero que no implicará mantenimiento adicional en el futuro.

El wrapper proporcionará todas las funciones del antiguo sistema pero se comportará como si fuera un objeto. Ante una nueva función a incluir en crear un nuevo juego de objetos para resolver el requerimiento. Cuando una función del antiguo sistema se requiera, este pedido será un mensaje enviado al wrapper, el wrapper descodificará, enviándolo al sistema

tradicional, el sistema responderá al wrapper, quien descodificará el mensaje y lo enviará al requester. A medida que se vayan cambiando parte o funciones del antiguo sistema, sólo bastará con inhibir la ejecución de dicha parte del antiguo sistema (a través del wrapper).

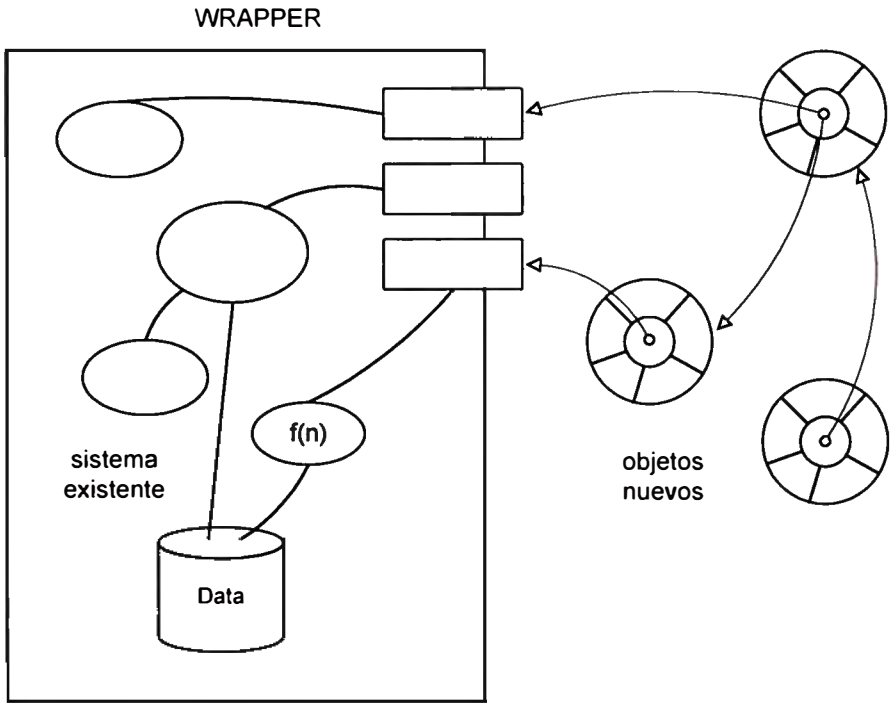



**Figura 3.11** Objetos wrapper

Al respecto, algunos productos como el Visual Age, utilizan el concepto de partes (Parts), que vienen a ser interfaces encapsuladas a manera de objetos que nos permiten comunicarnos con código tradicional. Un ejemplo de la aplicación del uso de wrappers es el encapsulamiento de toda la interfaz con el LANDP (Ver figura 3.11a).

SARA BANK cuenta con funciones especializadas de realizar las interfaces con el middleware LANDP. Las funciones de interfaz que provee SARA BANK son:

| | |
|---|---|
| Programa FBSS.EXE: | Encargado de la interfaz con el servidor de archivos planos. |
| Programa SNA:EXE: | Encargado de la interfaz con el servidor de comunicaciones SNA en LUO. |
| Programa LU62.EXE: | Encargado de la interfaz con el servidor de comunicaciones APPC. |

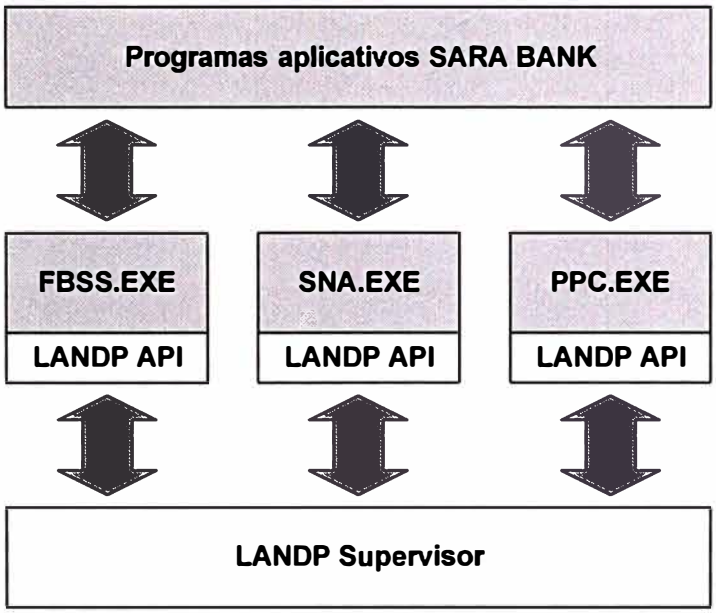



**Figura 3.11a** Funciones de interfaz con middleware

Por ejemplo, cada vez que un programa desea obtener un registro de un archivo con índice hace una llamada FBSS:EXE y le pasa como parámetros los siguientes datos el índice (ARAD002), la operación GU (get unique), el operando (EQ) y la clave de búsqueda.Ver figura 3.12.

**Programa aplicativo**

```
...
    PERFORM UBICA-REGISTRO.
...
UBICA REGISTRO.
```

```
MOVE "ARAD002" TO FILE.
MOVE "GU" TO OPERATION.
MOVE "EQ" TO OPERATOR.
MOVE AG02-KEY TO DATAREA.
CALL "FBSS" USING FILE,OPERATION,OPERATOR,DATAREA.
```

**FBSS.CBL**

```
...
*   carga de parámetros en registro interfaz...
...
*   llamada al API del LANDP
    CALL "LANDP_API_REQUEST" USING DATA-CPRB.
...
*   verificación y envío de respuesta a programa que lo invocó
```

**Figura 3.12** Llamada a una función de interfaz

Si se deseara pasar a TO dicha interfaz podría ser reemplazada por la Parte Visual Age (por ejemplo) que hace interfaz con LANDP u otro middleware encargado de las comunicaciones. De esa manera si el aplicativo requiere de otro middleware sólo el componente de interfaz requerirá ser cambiado.

Existen algunos inconvenientes y problemas que se afrontan al desarrollar los wrappers. Estos son los siguientes:

- El diseñador no tiene la libertad de elegir la mejor representación para el problema debido a que ya está decidida una gran parte del problema dentro del antiguo sistema.
- El diseñador debe exponer las funciones del sistema antiguo y generar interfaces para el usuario.

- Dado que el sistema antiguo continúa actualizando datos, el wrapper debe preservar el estado de esos datos cuando se llame a las rutinas internas.
- Manejo de memoria debe ser sincronizado entre el wrapper y el antiguo sistema.
- Las referencias cruzadas entre los datos nuevos y antiguos deben ser mantenidas.
- La construcción de un wrapper siempre requiere de un conocimiento detallado del antiguo sistema.

### 3.7.3 Las estrategias de migración

La idea de aprovechar la inversión realizada en software construído tradicionalmente a través de los llamados **wrappers** es atractiva. Los wrappers permiten interactuar componentes tradicionales con componentes nuevos orientados a objetos en el mismo sistema. Sin embargo, el uso de los wrappers bajo el enfoque de reutilización o migración puede generar la necesidad de duplicar datos. Existen cuatro posibles estrategias para enfrentar el problema del manejo de la información:

a) **Estrategia de convivencia o *handshake***, que consiste en mantener dos copias de la información y mantenerlas ambas actualizadas. El problema de integridad en este esquema es muy difícil. Sólo sería válido si la información del componente nuevo (OO) y antiguo (Clásico) no se superpone demasiado.

b) **Estrategia del préstamo o *borrowing***, que consiste en mantener la información en el sistema antiguo y solicitárselo sólo cuando se requiera. Se requiere de envío de mensajes al antiguo sistema para manejar las actualizaciones.

c) **Estrategia *take-over***, que consiste en copiar la data en los objetos y dejar de lado la información en el antiguo sistema. Este enfoque

también producirá problemas de integridad. Tendrá que desarrollarse un wrapper que no sólo reciba mensajes sino que además los responda.

d) **Estrategia de traducción**, consiste en dividir en paquetes la base de datos junto con sus funciones relacionadas. Requiere un método de análisis orientado a objetos capaz de describir el antiguo sistema y sus nuevos componentes.

Estas estrategias son aplicables si es una migración, una reutilización o sólo una construcción de front-ends para el sistema actual. Si nuestro objetivo fuera la migración, las estrategias de take-over y traducción son las más convenientes. La estrategia de traducción es posible si el sistema se puede descomponer alrededor de conjuntos de datos coherentes y además, si a partir de los DFDs y sus almacenamientos de datos se puedan crear objetos. Si no es posible, se tendrá que desarrollar un complejo wrapper usando la estrategia take-over. Una estrategia alternativa es utilizar una forma avanzada de la estrategia de traducción llamada **traducción centrada en datos (data-centred translation)**. Esta estrategia consiste en desarrollar ingeniería reversa al modelo de datos e identificar las operaciones de acceso a archivos. Los modelos de datos convencionales generalmente están normalizados, sin embargo tablas normalizadas no necesariamente corresponden a entidades del mundo real (objetos) por lo que es necesaria una **desnormalización.**

Para el caso de algoritmos altamente optimizados o funciones especializadas, la reutilización puede ser llevada a cabo usando los wrappers. Esto se hace posible mediante la definición de clases a nivel aplicación con métodos que llamen a las subrutinas del antiguo sistema.

Otro requerimiento usual será construir sobre soluciones empaquetadas. Una vez más, será necesario un wrapper que llame a las subrutinas del paquete o que simule dialogo en el terminal.

La traducción centrada en los datos (*data-centred transalation*) es la mejor solución para la migración y reemplazo de sistemas existentes, mientras que si nuestro objetivo es la reutilización de los componentes del antiguo sistema la estrategia de préstamo (borrowing) es la más viable.

Otro caso puede ser el de un sistema existente que funcione bien y cuyo mantenimiento sea moderado, en este caso las nuevas funciones deben ser desarrolladas usando el enfoque orientado a objetos y comunicarlo con el sistema antiguo a través de un wrapper. Las nuevas funciones son definidas como métodos de objetos que encapsulan la data que necesitan más adelante. Cuando la data almacenada por el antiguo sistema es requerida, un mensaje debe ser enviado al wrapper y la rutina apropiada de recuperación debe ser llamada, prestando la data requerida. Cuando el nuevo sistema orientado a objetos vaya absorbiendo más funciones del antiguo sistema necesaria la estrategia de traducción basada en los datos.

La figura 3.13 muestra las estrategias de comunicación de datos versus las cuatro posibles razones para la construcción de wrappers: migración de un sistema tradicional a una implantación orientada a objetos, reutilización de sus componentes sin cambiar el núcleo del sistema. aumentar su funcionalidad mediante el cambio del núcleo, y la construcción de un front-end para proporcionar funciones adicionales.

La palabra SI indica que es bastante factible su utilización , el NO representa que no es conveniente el uso de esa estrategia, y los ?? indican que depende de la complejidad y estructuración del sistema.

| | Migración | Reutilización | Extensión | Front-End |
|---|---|---|---|---|
| **Handshake** | NO | NO | NO | NO |
| **Borrowing** | NO | SI | SI | SI |
| **Take-over** | ?? | NO | NO | NO |
| **Traducción** | ?? | NO | SI | SI |
| **Traducción/datos** | SI | NO | SI | SI |

**Figura 3.13** Conveniencia de las estrategias de migración de datos para diferentes propósitos

Cuando se manejan versiones lo conveniente es construir wrappers alrededor del núcleo único para todas las versiones, para esto muchas veces es necesario hacer un modelo de análisis orientado a objeto que nos permita identificar dicho núcleo y la manera cómo nos comunicaremos con el componente orientado a objetos.

En general propongo una secuencia de pasos para la migración no violenta de los sistemas tradicionales:

1) Construir uno o varios wrappers para comunicarlo con los componentes orientados a objetos usando la estrategia borrowing o de préstamos.
2) Desarrollar un análisis orientado a objetos del antiguo sistema.
3) Usar la estrategia de traducción centrada en los datos para migrar.

Podemos concluir que mientras vayamos adquiriendo experiencia, la mejor solución a la migración del gran legado de sistemas con manejo de datos complejo es la de construir wrappers que soporten los front-ends orientados a objetos y construir las funciones nuevas requeridas bajo

estos front-ends. Las herramientas para la construcción wrappers no son numerosas, sin embargo, la primera herramienta a utilizar en este desarrollo debe ser el modelo basado en el análisis orientado a objetos.

A continuación presentamos la aplicación de dicha estrategia en el caso del producto SARA BANK. Como se sabe SARA BANK cuenta con un conjunto de componentes orientados fundamentalmente al tratamiento de transacciones bancarias (ventanilla, back-office, autoservicio, plataforma). La necesidad de automatizar más funciones dentro de la agencia o sucursal bancaria, nos conduce a la decisión de desarrollar un componente adicional denominado SARA PLAT cuyo objetivo es atender el frente de atención denominado plataforma de negocios. Este nuevo desarrollo será desarrollado usando la TO. Se ha escogido Visual Age para Smaltalk como herramienta de desarrollo por las facilidades que ofrece para la integración con tecnología tradicional vía PARTS y su gran potencia en el desarrollo visual.

En este esquema SARA PLAT tiene que convivir con los otros componentes (fundamentalmente con SARA BRANCH, componente que corre en la agencia). Dado que los archivos estarán en LANDP, la interfaz para acceder a ellos estará constituída por un wrapper que encapsule la función FBSS.EXE (la misma función usada por SARA BRANCH). Lo mismo en el caso de las comunicaciones con el HOST.

Es necesario para el SARA PLAT contar con un módulo transaccional de consultas de información del host. Este desarrollo aprovechará los servicios que ofrece el servidor de comunicaciones del SARA BRANCH (componentes tradicionales).

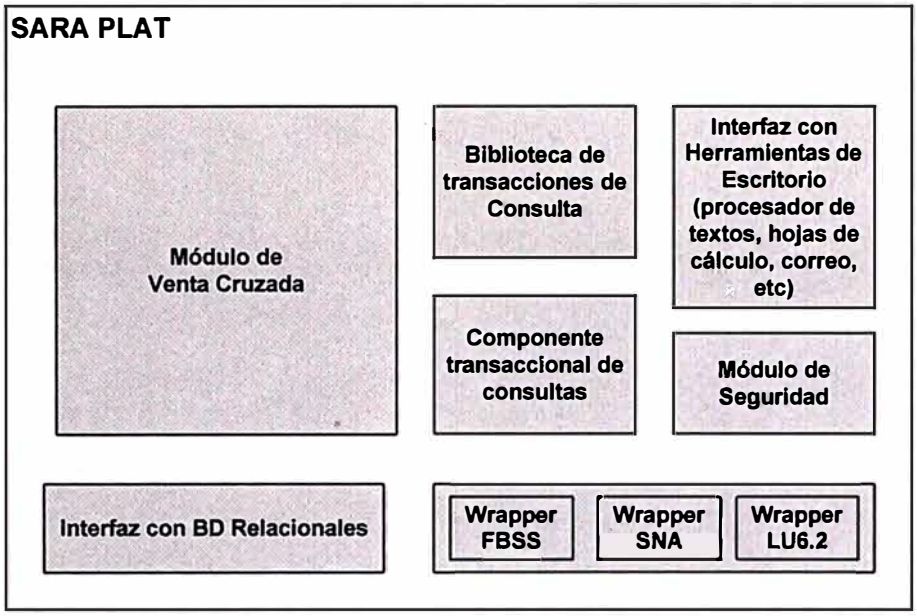



**Figura 3.14** Estructura de SARA PLAT.

La información que manejará el componente SARA PLAT es toda aquella con la que pueda contar para vender los productos y servicios a los clientes o prospectos. Por lo tanto la información se refiere a productos y servicios bancarios, e información del cliente (general, financiera, situación de todas sus cuentas, históricos, etc.). La mejor manera de almacenar esta información no transaccional y en un 90% de consulta es bajo una estructura relacional.

Como podemos constatar, se trata de una extensión al producto SARA BANK, y como tal, lo más apropiado para convivir con el resto del producto es aplicar la estrategia de borrowing o de préstamo, que implica el reuso de interfaces mediante wrappers. Una revisión del SARA BRANCH nos permitiría luego migrar este componente TO, siempre y cuando lo amerite. El SARA PLAT será íntegramente desarrollado en TO sin embargo la información adicional requerida estará en bases de datos relacionales. Ver figura 3.14.

### 3.7.4 Las bases de datos relacionales

La situación actual de las empresas habla de hasta un 90% de información manejada por las empresas que no reside en el computador. Esto hace ver que los sistemas y las bases de datos usadas hasta la fecha han resuelto sólo una parte del problema de información de los negocios. Sin embargo, no se puede negar que existe una gran inversión realizada en bases de datos relacionales que impide una migración masiva hacia bases de datos orientadas a objetos, y por que además, en los sistemas o aplicaciones orientadas a registros, las bases de datos relacionales cumplen su función adecuadamente. Es por estas razones que coincido con muchos autores en pensar en una coexistencia de ambos tipos de bases de datos. Esta interoperación objeto-relacional nos conduce a algunos problemas y a su vez a algunas soluciones tentativas:

Los problemas son:

- La construcción de aplicaciones orientadas a objetos que accedan a bases de datos relacionales.
- El uso de aplicaciones basadas en BD relacionales con bases de datos orientadas a objetos.
- Utilizar un lenguaje de consulta similar al SQL dentro de un ambiente orientado a objetos.

Como soluciones a estos problemas y en general a la interoperación objeto-relacional podemos señalar:

- Pasar completamente las aplicaciones y bases de datos a tecnologías de objetos.
- Usar facilidades estándar de importación y exportación de datos en ambos sentidos.

- Acceder a bases de datos relacionales desde lenguajes de programación orientados objetos.

### 3.7.5. Mapeo de Bases de Datos Relacionales a partir de Modelo de Objetos

A continuación se presentan las diferentes estructuras que se presentan dentro de los modelos de objetos y su correspondiente correlato en las estructuras de bases de datos relacionales.

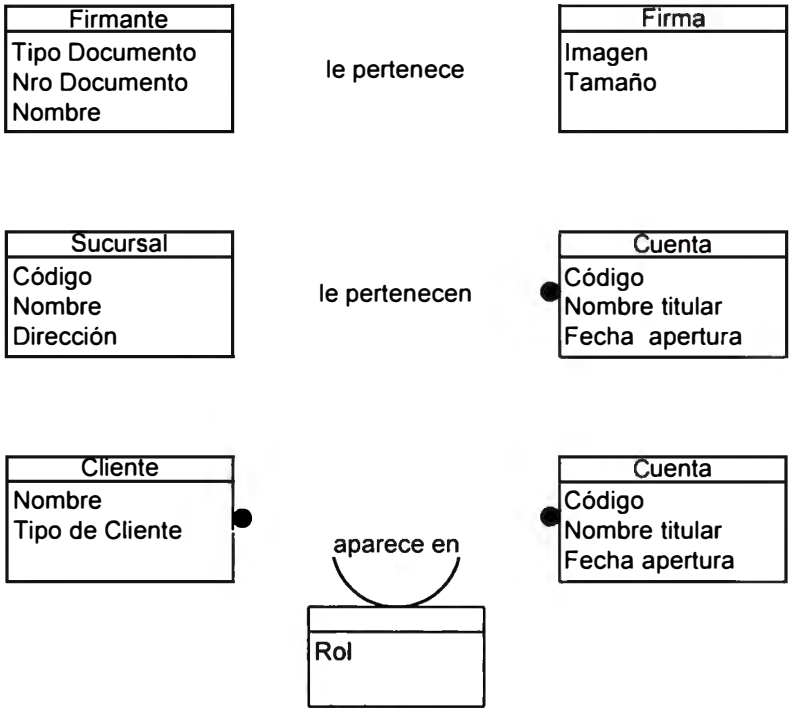



**Figura 3.15** Asociaciones entre clases

#### Clases y Asociaciones

El primer juego de ejemplos muestra como las asociaciones de clases de varios niveles pueden ser llevadas a un modelo relacional, incluyendo la inserción de claves. Las asociaciones son de tres niveles básicos: 1:1, 1:m, y m:n.

La asociación 1:1 está representada en la figura 3.15 por "*un firmante tiene una firma o rúbrica*". Generalmente una asociación 1:1 entre dos

clases debe desaparecer puesto que las dos clases se combinan en una sola entidad dentro del modelo físico de datos. Así tendremos simplemente el Firmante como la única entidad, como se muestra en la figura 3.16. Esta contiene todos los atributos de ambas clases participantes. También, una única clave (primary key), TipDocFir+NumDocFir ha sido insertado. La asociación 1:m está representada en la figura 3.15 por la relación "*una sucursal tiene cero o más cuentas*". Esta estructura se mantiene básicamente igual dentro del modelo físico de datos (figura 3.16). Las claves CodSuc y CodCta han sido añadidas a las entidades Sucursal y Cuenta, respectivamente. CodSuc se convierte en foreign key dentro de la entidad de Cuenta.

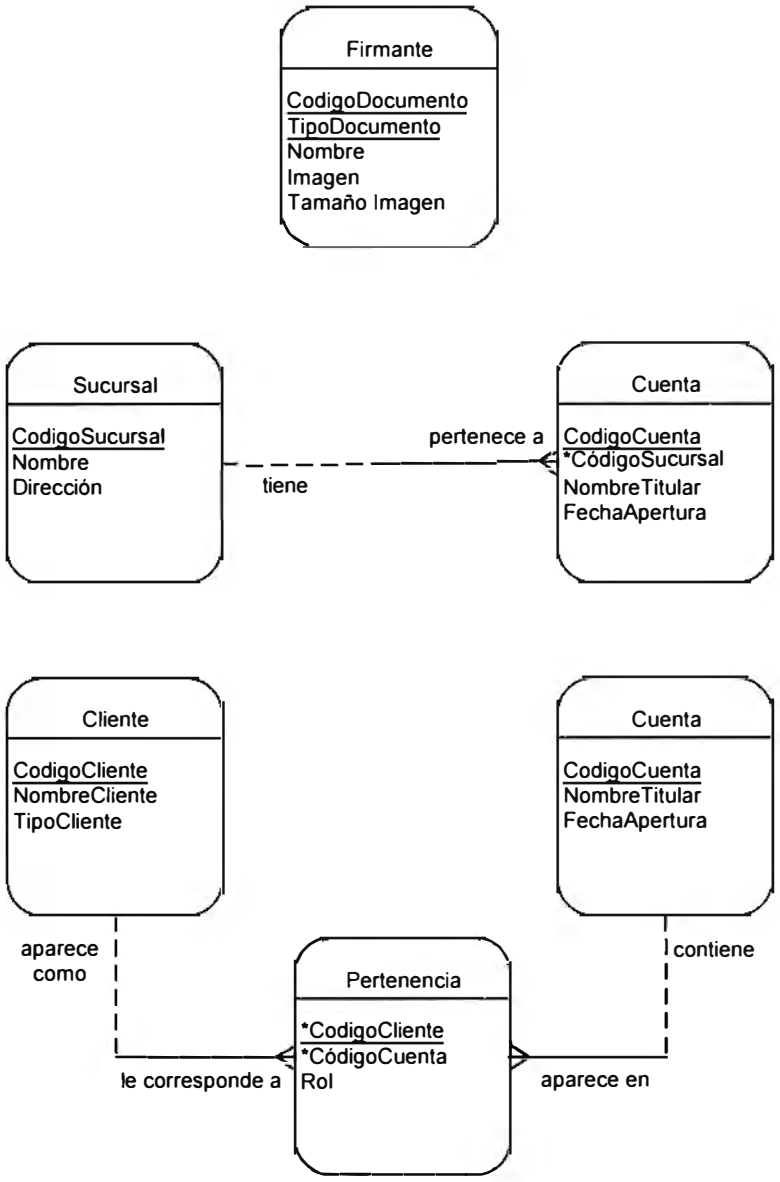



**Figura 3.16** El modelo físico de datos

La asociación m:n está representada en la figura 3.15 por "*un cliente puede tener cero o más cuentas, una cuenta puede tener cero o más clientes*". La asociación tiene un atributo de enlace denominado Rol, que no puede ser colocada como atributo dentro de ninguna de las clases. El mapeo involucra aquí la creación de una nueva entidad, Pertenencia, que contiene el atributo de enlace (figura 3.16). El nombre de la asociación siempre da una idea del nombre de la nueva entidad de enlace. Las clases Clientes y Cuenta mapean directamente a entidades de los mismos nombres con claves primarias CodCli y CodCta. La asociación m:n es

reemplazada por enlaces 1:m desde cada entidad a la nueva entidad de enlace. Las claves primarias de Cliente y Cuenta forman de manera conjunta la clave compuesta de la nueva entidad.


**Figura 3.17** La clase Cuenta posee varios firmantes

En algunos casos, nos encontramos con situaciones en las que tenemos que desdoblar una clase en dos o más, para evitar duplicidad de datos. La figura 3.17 muestra una situación de este tipo, la clase Cuenta puede contener las Restricciones como una entidad separada con una relación muchos a uno con Cuenta. Sin embargo, ¿cuál de las entidades tiene relación con Firma? No existe una solución única, las figuras 3.18 y 3.19 muestran dos posibles soluciones.

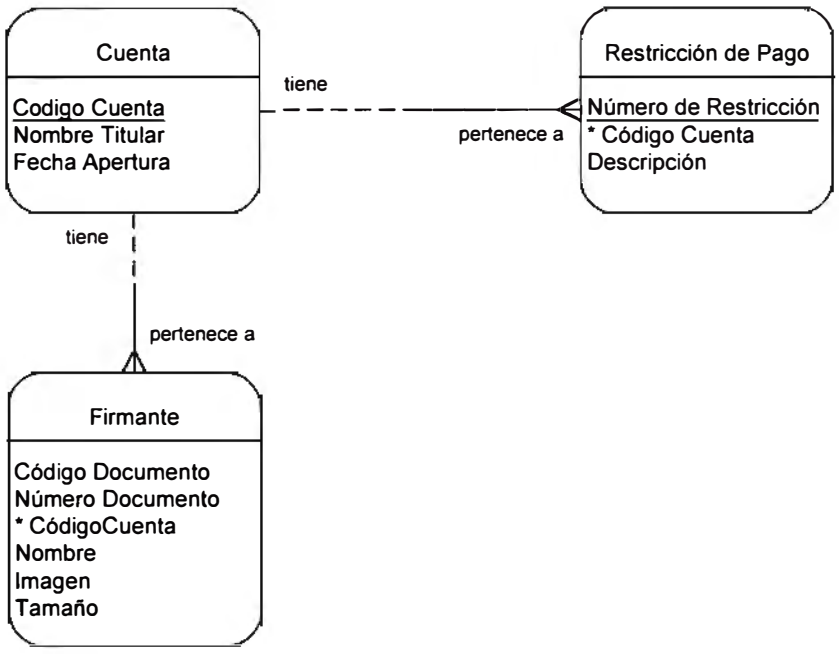



**Figura 3.18** Existe una relación Firmante y Cuenta pero no entre Firmante y Restricción

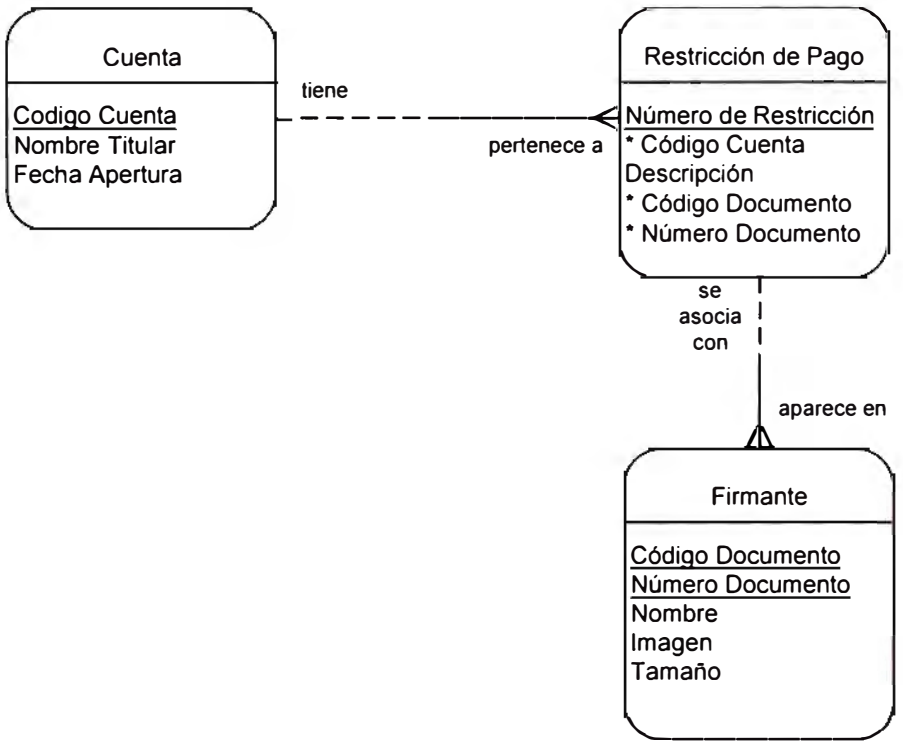



**Figura 3.19** Existe una relación Firmante y Restricción y a través de este último entre Firmante y Cuenta.

## Generalización

La generalización es algo diferente de la asociación y agregación. En la figura 3.20 Cliente es una clase abstracta. Cuando un CltePersNat oClteEmpresa son creados, cada uno heredará todos los atributos de la clase Cliente. Varias técnicas existen para mapear estas estructuras de generalización a modelos de bases de datos relacionales. A continuación una breve reseña de algunas de ellas.

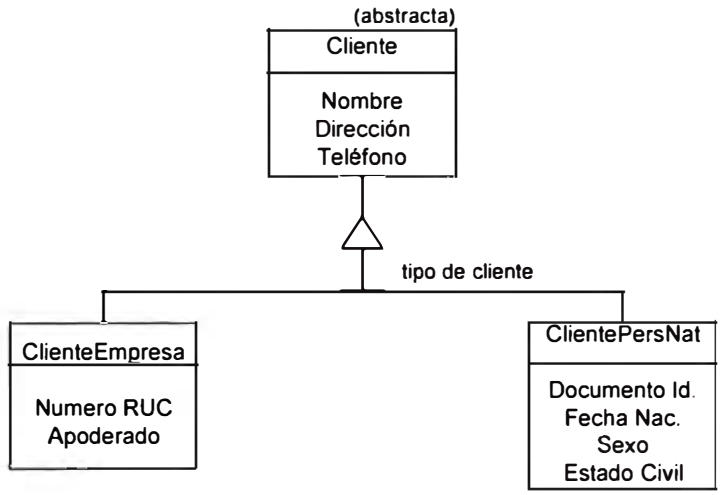



**Figura 3.18** La clase Cliente modelada como clase abstracta

Esquema *One-to-one*

Cada clase dentro de la estructura de generalización es mapeada a su propia identidad. Por lo tanto, una instancia de objeto explota en varias entidades, usando el mismo valor de clave primaria. El discriminador para la generalización, en este TipClte, se convertirá en un atributo adicional de la entidad superclase, como lo muestra la figura 3.21(a). Esta es probablemente la manera más directa de aplicar un mapeo sobre un modelo de objetos, pero el tener que navegar por la estructura de clase puede penalizar la performance.

Esquema *Rolled-Up*

Esta técnica involucra "recoger" todos los atributos de cada subclase hacia la superclase, y por lo tanto, toda la estructura de generalización está implementada como una única entidad (Ver figura 3.21(b)). Dependiendo en el tipo de objeto a ser creado, algunos atributos contendrán valores nulos. Esto es un enfoque muy útil si hay relativamente pocas subclases con pocos atributos.

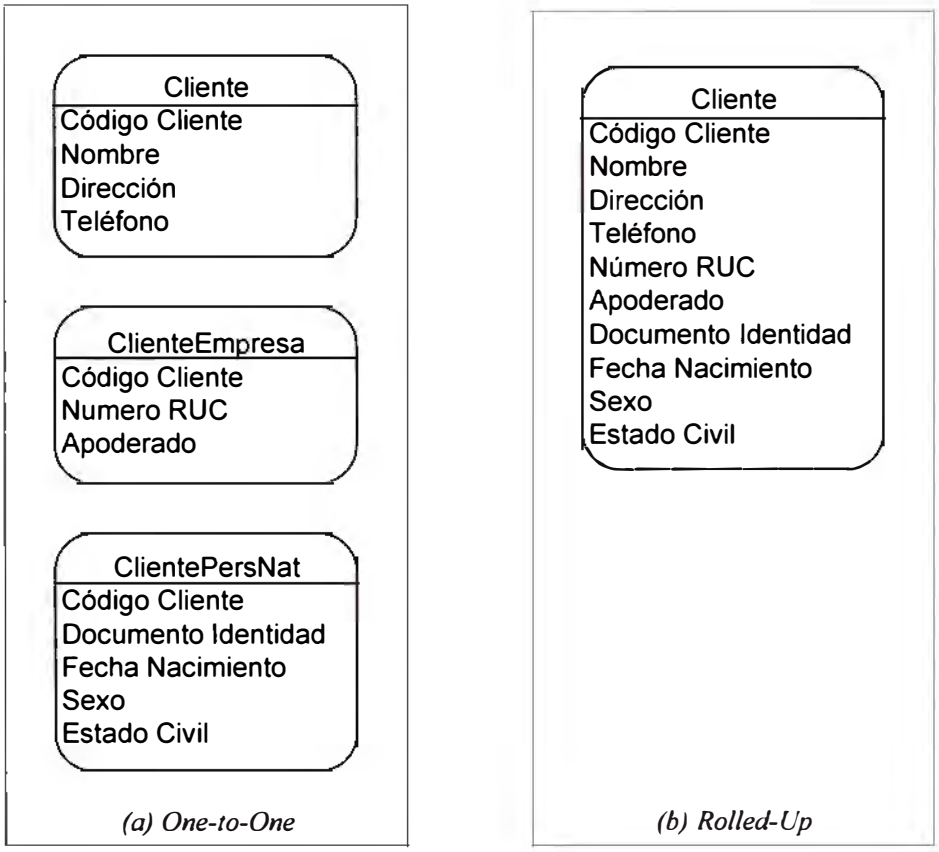



**Figura 3.21** Diferentes mapeos para la estructura de generalización

Esquema *Rolled-Down*

Bajo este esquema los atributos de la superclase son delegados a cada una de las subclases "concretas" (Ver figura 3.21(c)). Este es un enfoque muy utilizado cuando la superclase tiene pocos atributos y existen muchas subclases, o cuando las subclases tienen muchos atributos.

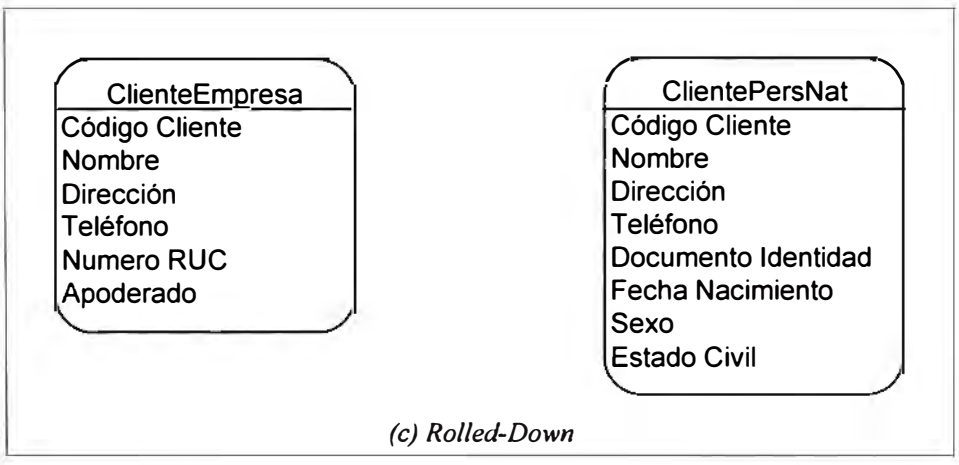



**Figura 3.21 (Cont.)** Diferentes mapeos para la estructura de generalización.

# 4. LAS METODOLOGIAS DE LA TECNOLOGIA DE OBJETOS

## 4.1 INTRODUCCION

Los métodos de desarrollo de sistemas pueden ser definidos básicamente en orientados a funciones y datos, y aquellos orientados a objetos. Los primeros tratan los datos y funciones como dos aspectos algo separados mientras que los segundos tienen una visión altamente integrada. Algunos de los métodos orientados a funciones y datos son el SADT (Structured Analysis and Design Technique), RDD (Requirement Driven Design basado en SREM) y SA/SD (Strucutred Analysis y Structured Design) de Yourdon y Constantine.

Dentro del enfoque de los métodos orientados a funciones y datos (que llamaremos a partir de este momento, métodos OFD) se asignan roles a las funciones y datos. Es así como las funciones son activas y tienen un comportamiento; mientras que los datos son recipientes pasivos de información afectados por las funciones. El sistema generalmente es dividido en módulos o funciones, donde la información es pasada de una función a otra. La división de datos y funciones tiene su origen en la arquitectura de hardware de von Newmann, que enfatizaba la separación del código de programa y los datos.

Un sistema desarrollado con un método OFD generalmente deviene en difícil de mantener. Uno de los principales problemas es que las funciones deben saber cómo los datos son almacenados, es decir, la estructura de datos. Existen muchas estructuras funcionales IF-THEN o CASE que no tienen que ver con la funcionalidad. Todo esto hace que los programas sean difíciles de leer, si sólo estamos interesados en la funcionalidad y no en la estructura de datos. Este tipo de sistemas generalmente son inestables ante los cambios, basta una modificación para que la arquitectura diseñada tenga que ser repensada.

Otro problema con los métodos OFD es la estructura de la solución que no refleja la estructura del pensamiento del ser humano. Inicialmente se describe el *qué hacer,* qué funcionalidades el sistema soportaría y que ítems deben existir en el sistema. Esto debe ser reformulado para describir el *cómo hacer*, para identificar la descomposición funcional del sistema. De esta manera creamos un *salto semántico* entra la visión externa e interna del sistema. Ahí radica una de las fortalezas de los métodos OO , pues el modelo se enriquece con el paso de una etapa a otra.

Si tenemos en cuenta que el sistema diseñado con métodos OFD son diseñados alrededor del comportamiento que deben tener, un cambio en algunas características producen mayores complicaciones en su implementación. Los métodos OO plantean la noción de definir el sistema en base a los componentes o ítems que lo conforman. Esto produce sistemas más estables pues los componentes son más estables que los procesos que los afectan. Al respecto Coad y Yourdon nos ofrecen una tabla donde muestran la tendencia al cambio de los componentes y procesos de un sistema.

| Ítem | Probabilidad de cambio |
|---|---|
| Objeto de la aplicación | Baja |
| Estructuras de información de larga vida | Baja |
| Atributos pasivos de un objeto | Mediana |
| Secuencias de comportamiento | Mediana |
| Interfaces con el mundo exterior | Alta |
| Funcionalidades | Alta |

A continuación revisaremos algunos de los más importantes aportes metodológicos en tecnología de objetos. Para graficar la aplicación de las

metodologías usaremos como ejemplo una aplicación de venta cruzada (cross selling), cuya fundamentación conceptual se explica en el anexo 2

## 4.2 PETER COAD Y DE YOURDON

Los libros titulados *Object-Oriented Analysis y Object-Oriented Design* de los autores mencionados fueron de los primeros documentos de la corriente metodológica de objetos. Esta metodología se caracteriza por su simplicidad y pragmatismo.

Los autores fundamentan su metodología sobre los tres principios de organización del pensamiento:

- Diferenciación entre los objetos y sus atributos (apariencia, calidades, etc.)
- Distinción entre la parte y el todo
- Constitución de las clases (jerarquías o clasificaciones)

Existen tres tipos de comportamiento que orientan la búsqueda de las clases:

- El fenómeno causa-efecto
- La similitud genética
- La similitud funcional

### 4.2.1 El Proceso

Los autores de esta metodología consideran cinco actividades dentro de la etapa de análisis:

1. Buscar las clases y objetos.
2. Identificar las estructuras.
3. Identificar los temas.
4. Definir los atributos.

❺ Definir los servicios.

Luego pasamos de manera *natural* a la etapa de diseño, que incluye fundamentalmente la incorporación a nuestro modelo de atributos y servicios impuestos por la elección de la arquitectura, hardware y software.

### 4.2.2 Identificación de los objetos

Dentro del modelo debemos considerar aquellos elementos externos que tienen comunicación con algunos de nuestros objetos. Se parte de una lista de candidatos a objetos, y a continuación, se evalúa cada uno de ellos. Debemos asignarles un nombre. Debemos quedarnos con aquellos objetos a los cuales podemos afectar un servicio o atributo.

### 4.2.3 Definición de estructura

Para establecer las relaciones de estructura entre los objetos, aplicamos los métodos de organización lógica:

- Agregación, que consiste en la abstracción de las características dentro de la definición de un objeto.
- Ensamblaje de los componentes dentro del todo
- Clasificación, a través de la generalización y especialización de los objetos.

### 4.2.4 Identificación de los temas

Se refiere a la identificación de subdominios, de modo que sea fácil la repartición del trabajo. Las fronteras no deben cortar los enlaces de estructuras. Estos dominios definidos deben mostrar una cohesión interna y un acoplamiento débil.

### 4.2.5 Definición de atributos

Durante esta tarea se fija la cardinalidad de las conexiones entre los objetos. Una conexión portadora de un atributo se convierte en un objeto. Es conveniente convertir las propiedades multivalor en objetos, y eventualmente, asociarlos a un objeto padre mediante el lazo de ensamblaje. A este nivel el carácter de público o privado del atributo no es importante.

### 4.2.6 Definición de los servicios

Podemos clasificarlos en tres tipos:

1. Los constructores: crear objetos, actualizarlos, destruirlos.
2. Los informadores: que suministran una información
3. Los transformadores: modificar, lanzar, parar

El analista deberá aplicar esta tipología sobre cada uno de los objetos. Los mensajes abarcan los eventos, las respuestas y los flujos de datos que intercambian los objetos. Transitan a lo largo de las conexiones de instancias.

La especificación de los servicios es completa cuando se indican por cada servicio: la función que realiza, los parámetros de entrada y de salida, el grado de concurrencia y paralelismo interno.

### 4.2.7 Definición de estados

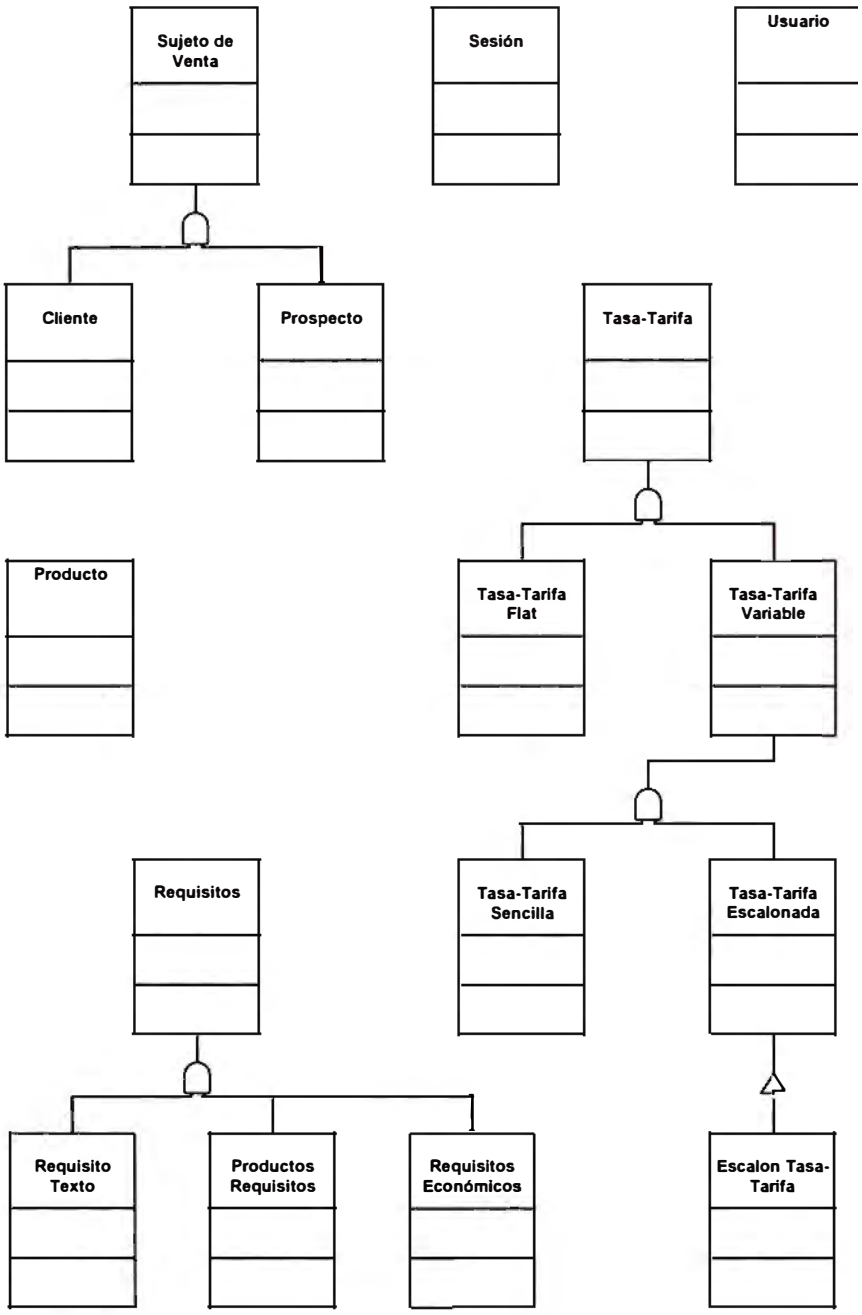
Para este fin la metodología recurre a la teoría de autómatas de estado finito, modelados a través de un diagrama de estado-transición.

### 4.2.8 Caso Práctico : Módulo de Venta Cruzada SARA BANK

*Lista de objetos*

| *Objeto* | *Descripción* |
|---|---|
| **Sujetos de Venta** | todos aquellos individuos que se acercan al funcionario de negocio. |
| **Clientes** | todos aquellos individuos que tienen o usan productos o servicios del banco. Ejemplo: alguien con Cta. Cte., o de ahorros. |
| **Prospectos** | todos aquellos individuos que no tienen vinculación alguna con el banco. |
| **Productos** | Ejemplos: Ctas. Corrientes, Ctas de Ahorro, Ctas a Plazo, Préstamos, Cargo Automático en Ctas por Concepto de Serv. Públicos, etc. |
| **Productos Primarios** | Es aquel demandado u ofrecido al cliente, sobre el cual se realiza la gestión de venta. Ejemplo: Tarjeta de Crédito. |
| **Productos Complementarios** | Acompañan a otros productos Ejemplo: seguro de vida por la adquisición de una cta a plazo. |
| **Productos Suplementarios** | Aquellos productos que ofrecen al cliente una alternativa a productos de características similares. Ejemplo: Tarjeta de Débito |
| **Productos Requisitos** | Aquellos productos cuya adquisición previa es necesaria para la obtención de otro. Ejemplo: la cta. cte. es requisito para tener tarjeta de crédito. |
| **Tasas-Tarifas** | Porcentajes o Montos fijos que se aplican sobre los montos solicitados para la adquisición de un producto. Ejemplo: la tasa de interés de un préstamo. |
| **Sesión de Venta** | El ambiente y los procesos realizados durante la venta a un sujeto de venta. |
| **Vendedor** | Aquel que opera el módulo de venta cruzada. |

## *Estructura de Objetos*

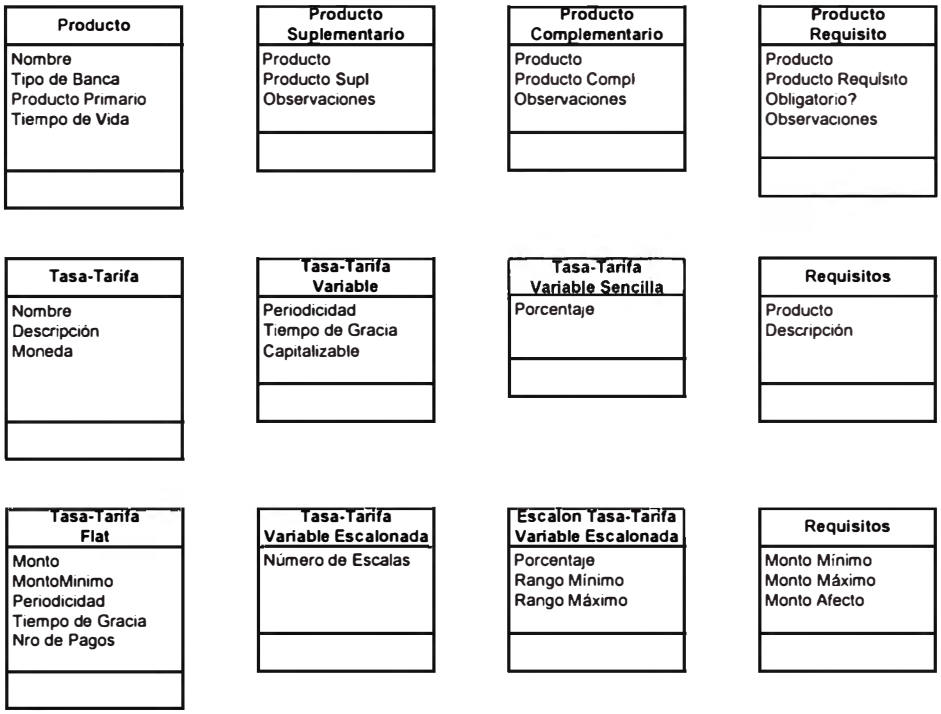



## *Identificación de Temas*

- Productos
- Clientes y Prospectos
- Venta

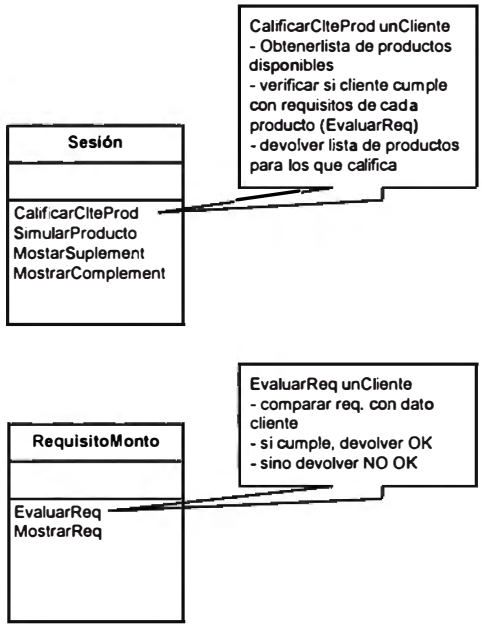
## ***Identificación de Atributos***

A continuación se presentan los atributos de los objetos del tema Productos.



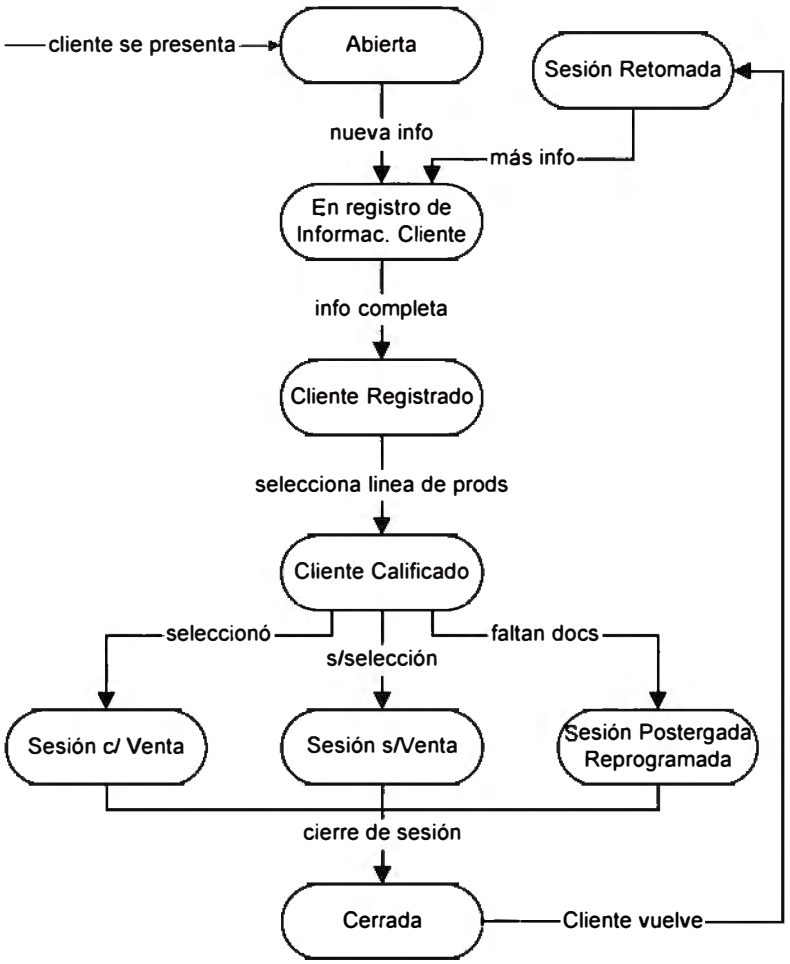

## ***Definición de Servicios***

A continuación mostramos los servicios de los objetos Sesión y Requisito-Monto.



***Definición de Estados***

Definamos los posibles estados para el objeto Sesión.




## 4.3 OMT Y JAMES RUMBAUGH

La metodología OMT (Object Modeling Technique) desarrollada y planteada en el libro Object-Oriented Modeling and Design de James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy y William Lorensen, representa un gran paso hacia la formalización del enfoque orientado a objetos.

### 4.3.1 Las bases teóricas

Las características de los objetos que los autores consideran centrales son:

- La identidad, dentro de un mundo real un objeto simplemente existe pero en un lenguaje de programación dicho objeto debe tener un handle o una dirección que lo referencia.
- La clasificación, que es posible el agrupamiento de diferentes objetos en clases.
- El poliformismo, una operación puede tener comportamientos diferentes dependiendo del objeto sobre el cual actúa.
- La herencia, que no es más que el compartir atributos y operaciones entre clases dentro de una jerarquía.

Para la OMT, el término operación es la abstracción de comportamientos análogos sobre diferentes objetos. La implantación de una operación es un método.

### 4.3.2 Los modelos

La estructura de dicha metodología se centra en la elaboración de tres tipos de modelos del sistema o espacio problema: modelo de objetos, modelo dinámico y modelo funcional; y de cuatro etapas: análisis, diseño del sistema, diseño de objetos e implantación. Cada uno de estos modelos son enriquecidos a medida que vamos avanzando en las etapas de desarrollo.

Los modelos de objetos, dinámico y funcional se encuentran interrelacionados entre sí y representan una visión parcial del todo, sin embargo podemos señalar que el más importante de los tres es el modelo de objetos, pues debemos estar seguros de lo que cambia o se transforma antes que el cuándo y el cómo. Este es el mismo orden que debemos respetar al hacer el modelamiento: primero el qué, luego el cuándo y al final el cómo. Este esquema favorece la evolutividad.

### 4.3.3 El modelo de objetos

Describe la estructura de los objetos dentro del sistema (su identidad, sus relaciones con otros objetos, sus atributos y sus operaciones), es un modelo que captura la estructura estática del problema o sistema. Proporciona el marco de referencia esencial dentro del cual los modelos dinámico y funcional serán ubicados. Es un modelo que abstrae la realidad y la plasma en una representación natural, pues podemos afirmar sin temor a equivocarnos, que los objetos son las partículas que conforman el mundo real.

Un objeto dentro de este modelo está compuesto de atributos y operaciones. Adicionalmente, los objetos forman enlaces a nivel ocurrencias y asociaciones a nivel clases. Se denomina un enlace a una conexión física o conceptual entre ocurrencias de objetos. Mientras que llamamos asociación a un grupo de enlaces con una estructura y significado común.

Los enlaces y asociaciones entre objetos son representados a través de los diagramas de clases y diagramas de ocurrencias. Un diagrama de clase representa la relación entre dos o más objetos, mientras que el diagrama de ocurrencias muestra la relación entre ocurrencias particulares de las diferentes clases.

Una característica importante de los enlaces y asociaciones se refiere a la multiplicidad. Este concepto se refiere al número de ocurrencias de una clase que se relacionan con una sola ocurrencia de la clase asociada. La multiplicidad puede ser: de uno a uno, de uno a muchos, de muchos a muchos, de uno a ninguno.

Las asociaciones pueden tener atributos al igual que las clases; así, podemos decir que una asociación de dos objetos contiene atributos particulares para cada enlace entre sus ocurrencias. En algunas casos, es conveniente considerar una asociación como una clase. De modo que cada enlace se convierte en ocurrencia de una clase. Es útil considerar una asociación como una clase cuando los enlaces intervienen dentro de asociaciones con otros objetos, o cuando los enlaces son objeto de operaciones.

Un tipo especial de asociación son los agregados, que representan la relación "todo-parte" o "es una parte de". Este tipo de asociación se caracteriza por su transitividad (si A es parte de B, y B es parte de C, luego, A es parte de C), y por ser antisimétrica (si A es parte de B, entonces B es parte de A).

Dentro del modelo de objetos debemos además señalar la estructura jerárquica de herencia y generalización, mediante la cual se definen clases que agrupan los atributos y operaciones comunes, y bajo ellas, clases que refinan, adicionan, eliminan o modifican dichos atributos y operaciones. La utilización de generalización y herencia de los objetos produce en muchos casos la creación de clases abstractas y clases concretas. Las primeras se refieren a clases que no poseen ocurrencias pero cuyas clases descendientes tienen ocurrencias directas. Las clases concretas son aquellas que tienen ocurrencias directas en el mundo real.

### 4.3.4 El modelo dinámico

El modelo dinámico es aquel que representa los aspectos del sistema concernientes con el tiempo y los cambios. Para ello se identifican eventos y estados de los objetos previamente definidos en el modelo de objetos. Los valores de los atributos y enlaces contenidos en un objeto se

denomina estado. Denominamos evento a un estímulo individual de un objeto hacia otro. La respuesta a este evento depende del estado del objeto receptor, y ésta puede ser un cambio de estado o envío de otro evento al emisor inicial o a un tercero. Los patrones de eventos, estados y transiciones de estado para una clase dada son representados por un diagrama de estado. Justamente un modelo dinámico consiste en el conjunto de diagramas de estado, uno por cada clase cuyo comportamiento dinámico sea importante..

Los eventos poseen atributos al igual que los objetos. Cada evento es único sin embargo se pueden agrupar en clases de eventos si tienen una estructura y comportamiento comunes. Un escenario es una secuencia de eventos que ocurren durante una ejecución particular de un sistema. Un evento corresponde a un intervalo de tiempo mientras que un evento representa puntos de tiempo.

Un diagrama de estado permite visualizar la dinámica de los objetos en el sistema. Una actividad es una operación que se realiza mientras un objeto se encuentra en un determinado estado. Luego, un estado puede estar asociado a una actividad. Por otro lado, los eventos se pueden asociar a acciones (operaciones instantáneas en respuesta a un evento) o estar sujetos a ejecución dependiendo de situaciones o condiciones.
Así, el diagrama de estado ofrece una representación dinámica de los objetos donde se involucran eventos, estados, actividades durante los estados, acciones debidas a los eventos y condiciones para los eventos.

Los eventos y estados también están sujetos a generalización y agregación. Es posible agrupar un conjunto de estados en uno solo con el objeto de mejorar la visión del conjunto. Esto nos permitirá trabajar con diagramas de estado anidados. La generalización de eventos permite

diferentes niveles de abstracción y de esta manera contar con representaciones genéricas o más específicas dependiendo del detalle con el que estemos hablando. Por otro lado, existe la posibilidad de presentar diagramas de estados concurrentes, pues existe casos (como en el de los objetos agregados) en que es necesario representar combinaciones de estados de diferentes objetos que se interrelacionen mucho.

### 4.3.5 El modelo funcional

El modelo funcional describe cálculos u operaciones dentro del sistema. Es el tercer ángulo de observación del problema. El modelo funcional especifica que pasa, el modelo dinámico cuándo pasa y el modelo de objetos especifica a quien o a que le pasa.

El modelo funcional consiste en múltiples diagramas de flujo de datos que muestran el flujo de valores provenientes de entradas externas, a través de operaciones y almacenamientos de datos internos, hacia salidas externas. Un diagrama de flujo de datos contiene procesos que transforman los datos, flujos de datos que mueven los datos, objetos actores que producen y consumen datos, u objetos de almacenamiento que guardan los datos pasivamente.

### 4.3.6 El método de desarrollo

En lo que se refiere al ciclo de desarrollo, esta metodología propone la construcción de un modelo del aplicativo, al cual se le va agregando los detalles de implantación. Las etapas identificadas son:

a) Análisis, a partir de la expresión de un problema, se construye un modelo que sea la abstracción precisa del QUE del sistema solicitado.
b) Diseño del sistema, fase que conlleva a las decisiones de la arquitectura general, los rendimientos y las aptitudes del futuro sistema.

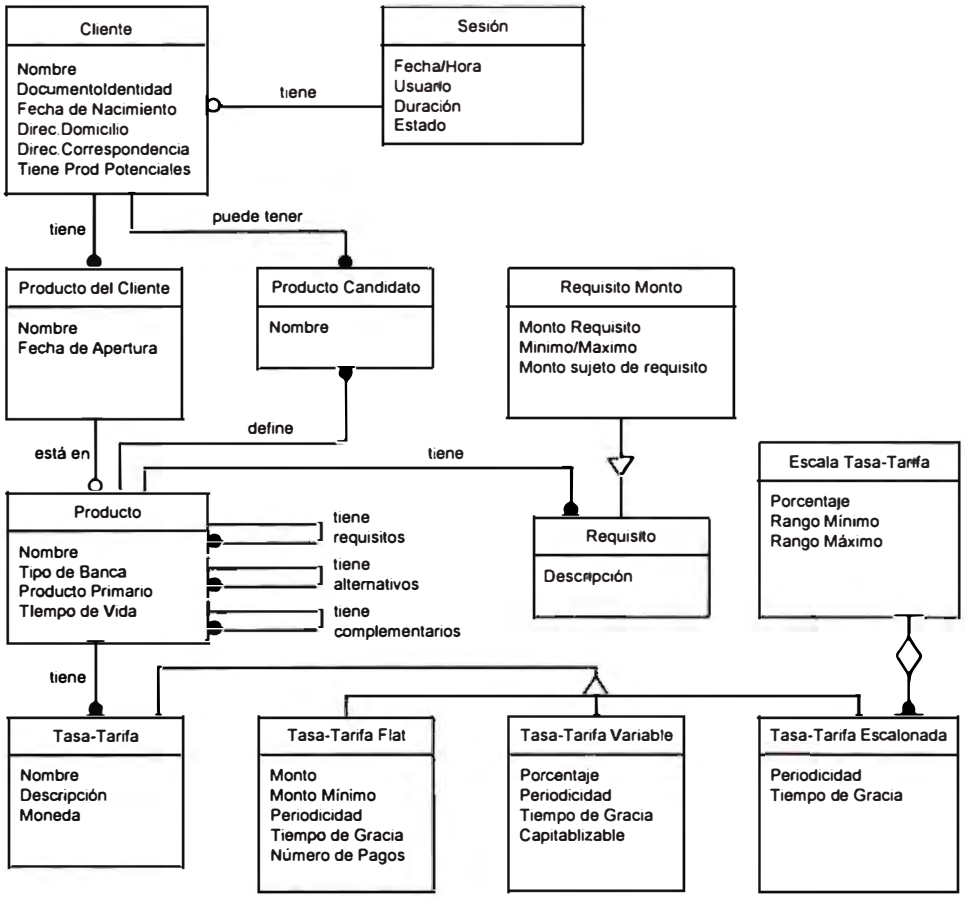
c) Diseño de objetos, aportes de los detalles de la implementación al modelo objeto.

d) Programación o codificación, que idealmente podemos presentar como la transcripción del modelo objeto definitivo al lenguaje y sistema de manejo de datos seleccionados.

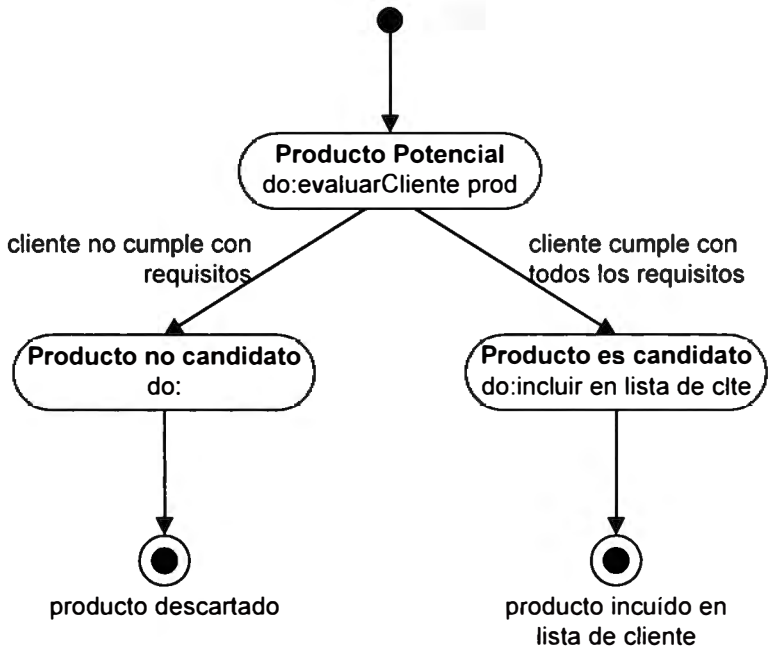
La OMT comparte el dogma del esfuerzo y dedicación a las fases de análisis y diseño antes que a la fase de programación. La OMT precisa las tareas a ejecutar sobre los tres modelos en cada fase, sin embargo no presenta un esquema de trabajo para atacar un proyecto de gran envergadura.

### 4.3.7 Caso Práctico: Módulo de Venta Cruzada SARA BANK

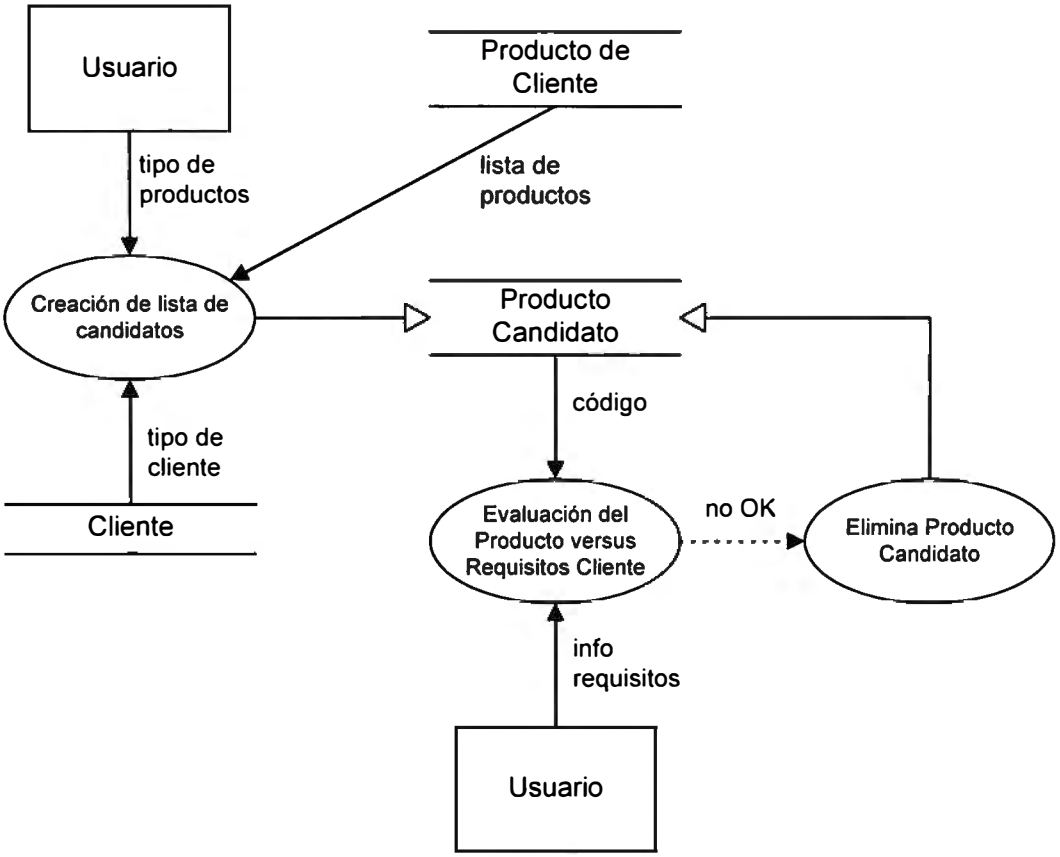
*Modelo de objetos*



*Modelo Dinámico*




*Modelo funcional*



## 4.4 JAMES MARTIN Y JAME J. ODELL

Luego de la serie de publicaciones que hiciera sobre la Ingeniería de la Información, ambos autores se suben al carro de la orientación a objetos con el libro *Object Oriented Analysis and Design* (1992). En esta publicación se presentan cuatro modelos básicos de la metodología, los dos primeros corresponden al análisis y los segundos al diseño:

1. Análisis de la Estructura de Objetos (OSA)
2. Análisis del Comportamiento de Objetos (OBA)
3. Diseño de la Estructura de Objetos (OSD)
4. Diseño del Comportamiento de Objetos (OBD)

Se destaca la capacidad de esta metodología (como de otras con orientación a objetos) de cubrir las etapas de análisis, diseño y, en algunos casos, hasta de construcción (codificación) de una manera natural y con modelos coherentes entre sí.

El proceso que vino a denominarse Ingeniería de la Información y que consistía en la obtención de un modelo de negocio prevalece y se adapta a las nuevas exigencias de la orientación a objetos. Así se asocian con la etapa de Planeamiento Estratégico de la Información y Análisis de las Áreas de Negocio, los modelos producidos por el Análisis de la Estructura y Comportamiento de Objetos (OSA y OBA); y con el Diseño del Sistema, el Diseño de la Estructura y Comportamiento de Objetos (OSD y OBD).

### 4.4.1 Análisis de la Estructura de Objetos (OSA)

Consiste en la identificación de los tipos de objetos y cómo están asociados. Se basa en la identificación de los objetos y su clasificación de acuerdo a los siguientes criterios:

- ☑ Identificación de tipos y supertipos de objetos
- ☑ Identificación de estructuras todo-parte
- ☑ Identificación de relaciones entre objetos

### 4.4.2 Análisis del Comportamiento de Objetos (OBA)

Se concentra en la elaboración de lo que Martin y Odell denominan del *esquema de eventos*, que no es más que una versión personalizada del diagrama estado-transición de los sistemas de procesamiento de datos en tiempo real. Este esquema muestra los eventos que se producen y los cambios de estado que producen en los objetos. Se proponen algunas actividades para analizar la cinética del espacio problema:

- Focalizar el espacio problema, identificar los objetivos y asimismo identificar los eventos tipo objetivo que deben alcanzarse para llegar a cumplir con el objetivo del sistema. Cada una de estas metas está asociada a eventos.
- Identificación del tipo de evento, sus pre y post-estados. Asimismo, denominar el evento.
- Generalización del evento, definición del nivel de abstracción del mismo e integración a la clasificación de eventos del sistema.
- Definición de las condiciones de la operación. Se analiza la naturaleza de la operación para saber si ella es de carácter interno o externo, identificar las condiciones de control que determinan el comportamiento del evento.
- Identificación de los tipos de eventos prerequisito (triggering events) de la operación. En muchos casos, los eventos activadores o prerequisito deben darse en grupo, por lo que es importante distinguir sólo lo que verdaderamente activa o permite la realización del evento. En algunos casos, es importante considerar eventos previos que permiten agrupar ciertos prerequisitos en grupos más pequeños.
- Refinamiento del diseño del evento. Este es un evento con el se culmina la definición y que consiste en una revisión de la labor

realizada. Las actividades más importantes de este paso son la búsqueda y la generalización de tipos de eventos prerequisitos.

### 4.4.2 Diseño de la Estructura y Comportamiento de Objetos (OBD y OSD)

De acuerdo a la metodología propuesta por Martin y Odell ambas fases se trabajan juntas, por considerarse que las estructuras de datos y métodos (clases) deben volcarse hacia los lenguajes de orientación a objetos.

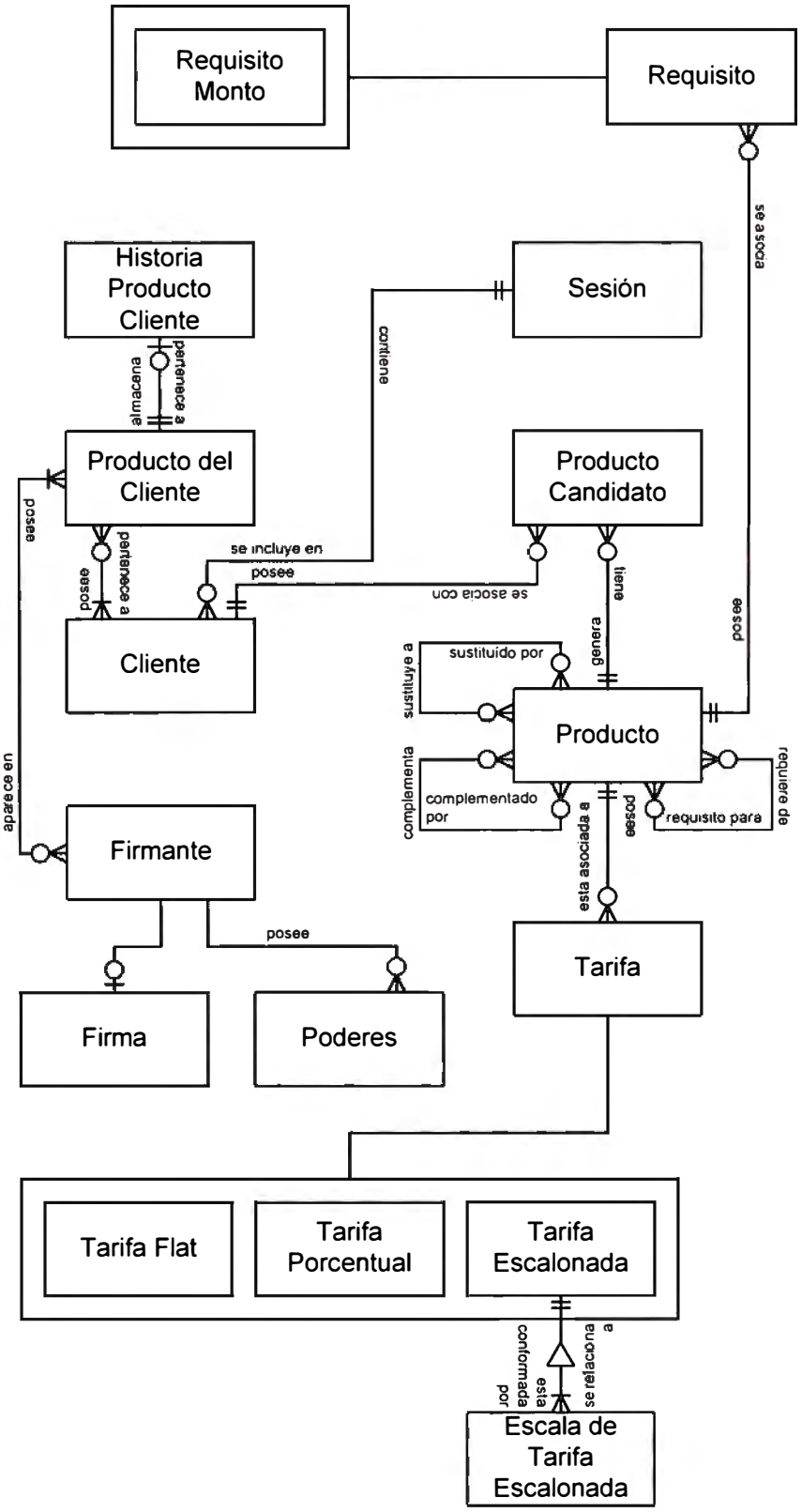
Las principales actividades de esta metodología son:

- ☑ Definir las clases a implantar en base al análisis proporcionado por las fases anteriores. Los tipos de objetos definidos previamente son los candidatos para la implementación de clases.
- ☑ Definir la estructura de datos con la que cada clase contará.
- ☑ Definir las operaciones y métodos que cada clase ofrecerá.
- ☑ Definir la manera de aplicar la herencia a las clases identificadas.

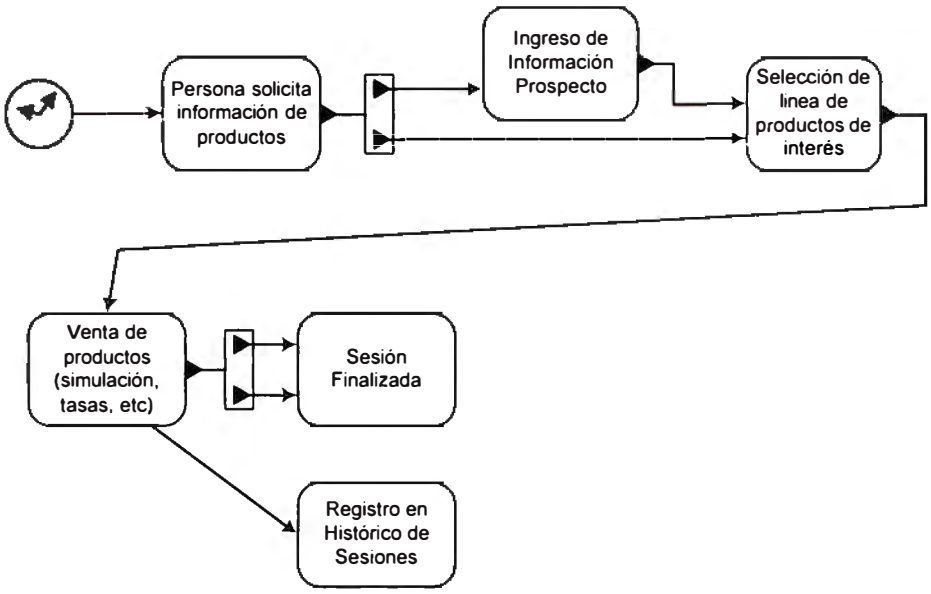
### 4.4.3 Caso Práctico : Módulo de Venta Cruzada SARA BANK

(ver siguente página)

*Modelo de Estructura de Objetos*



*Modelo de Comportamiento de Objetos*




## 4.5 OBJECT-ORIENTED SOFTWARE ENGINEERING

La presente metodología está condensada en el libro *Object-Oriented Software Engineering. A Use Case Driven Approach* de Ivar Jacobson. En ella se describen las fases básicas que considera dentro del desarrollo de sistemas con orientación a objetos. Es interesante señalar que incluye tres fases dentro de su metodología: el análisis, la construcción y las pruebas.

### 4.5.1 Análisis Orientado a Objetos

El propósito es siempre el mismo, el de conseguir una comprensión de la aplicación, teniendo sólo en cuenta los requerimientos funcionales del sistema. Dentro de este objetivo dos modelos son desarrollados: el modelo de requerimientos y el modelo de análisis propiamente dicho. Ambos son modelos lógicos pues no incorporan requerimientos de la implementación física.

El modelo de requerimientos utiliza actores y casos de uso para describir en detalle cada manera de usar el sistema desde la perspectiva del usuario. El modelo de actores permite mostrar los agentes que

interactúan con el sistema. Los casos de uso son flujos que estos actores ejecutan dentro del sistema. Los casos de uso deberán ser entendidos de manera intuitiva por personal no-técnico y es la base para la comunicación y definición de los requerimientos del sistema. Como soporte para el modelo de requerimientos, el modelo de objetos del dominio problema es una herramienta importante para determinar qué objetos están involucrados en el sistema. Generalmente esta metodología aconseja acompañar al modelo de requerimientos de las interfaces del sistema.

El modelo de caso de uso constituye la columan vertebral de la metodología. Estará presente durante todo el trabajo de desarrollo con OOSE, desde la estructuración del sistema para definir las operaciones sobre los objetos hasta su uso como herramienta para las pruebas de integración.

El modelo de análisis es el segundo modelo desarrollado en el proceso de análisis. Este modelo ayuda a formar una estructura lógica y mantenible dentro del sistema. Es lógica en el sentido que el entorno de implementación real no es tomado en cuenta. Su objetivo está focalizado la funcionalidad esencial del sistema. Tres tipos de objetos son utilizados para estructurar el sistema:

- Los *objetos de interfaz* modelan toda la funcionalidad concerniente a las interfaces del sistema.
- Los *objetos de entidad* modelan toda la funcionalidad que maneja la información que residirá en el sistema por períodos largos.
- Los *objetos de control* modelan aquella funcionalidad que no está asociada a otro objeto (generealmente comportamiento).

Estos objetos son identificados cuando los casos de uso son anlaizados y descompuestos. Los objetos deberán proporcionar la completa funcionalidad de los casos de uso. A este nivel podemos dividir el sistema en subsistemas.

### 4.5.2 Construcción Orientada a Objetos

Dentro de esta fase se procede al diseño e implantación en código fuente. Este código fuente deberá ser ejecutado en el ambiente, sin que esto impida que el paso del análisis a la construcción sea prácticamente natural. Es en esta fase donde tomamos conciencia de las limitaciones de memoria, confiabilidad y tiempo de respuesta. Es necesario repetir que es importante la persistencia de los objetos definidos en el modelo de análisis dentro de la implantación, pues estos reflejan el sistema que hemos estudiado. Sin embargo y dado que el modelo del análisis ha sido concebido dentro de un mundo ideal, es en esta fase donde agregamos nuevas dimensiones al problema.

Como fruto de esta fase debemos obtener un modelo de diseño, un modelo que refine y formalice el modelo de análisis teniendo en cuenta el ambiente de implantación. Las actividades dentro de esta fase son:

① Identificar el ambiente de implantación. Implica identificar las consecuencias de la elección del ambiente en el diseño. Aquí todas las definiciones estratégicas deben ser hechas, como por ejemplo, ¿qué SGBD será usado?, ¿qué lenguaje de programación?, ¿cómo se efectuará la comunicación entre procesos?, etc. En general, podemos decir que éste es un paso que se va definiendo en paralelo con el análisis.

② Incorporar estas premisas en el desarrollo del primer modelo de diseño. Tomando como base el modelo de análisis e incorporando todas las limitaciones y ventajas del ambiente de

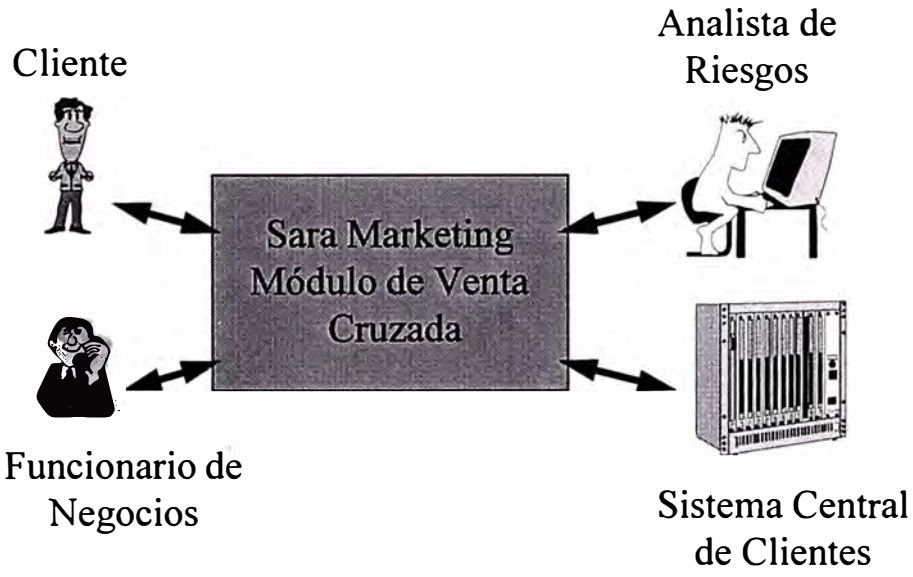
implantación se procede a la construcción del modelo de diseño. A partir del objeto del análisis construímos el objeto del diseño.

③ Describir cómo los objetos interactúan en cada caso. Se definen todos los estímulos enviados entre los objetos y cada una de las operaciones a ejecutar por los objetos. Esta actividad arrojará como resultado las interfaces de objetos.

### 4.5.3 Caso Práctico : Módulo de Venta Cruzada SARA BANK (Análisis)

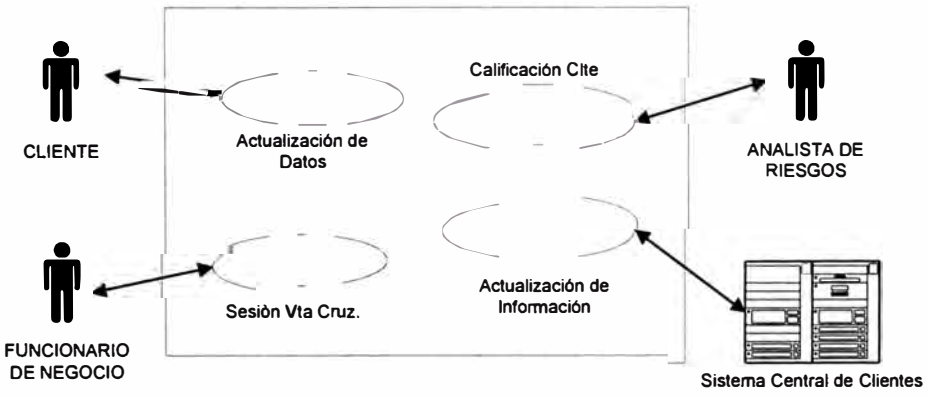
***Modelo de Requerimientos***

*Identificación de Actores*

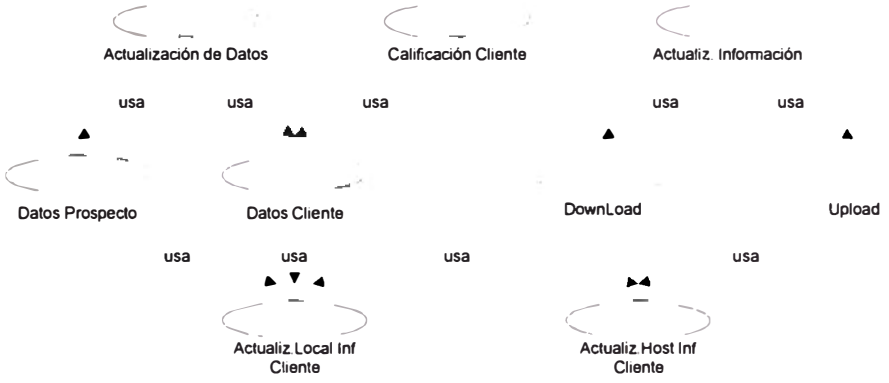



*Modelo de Casos de Uso*

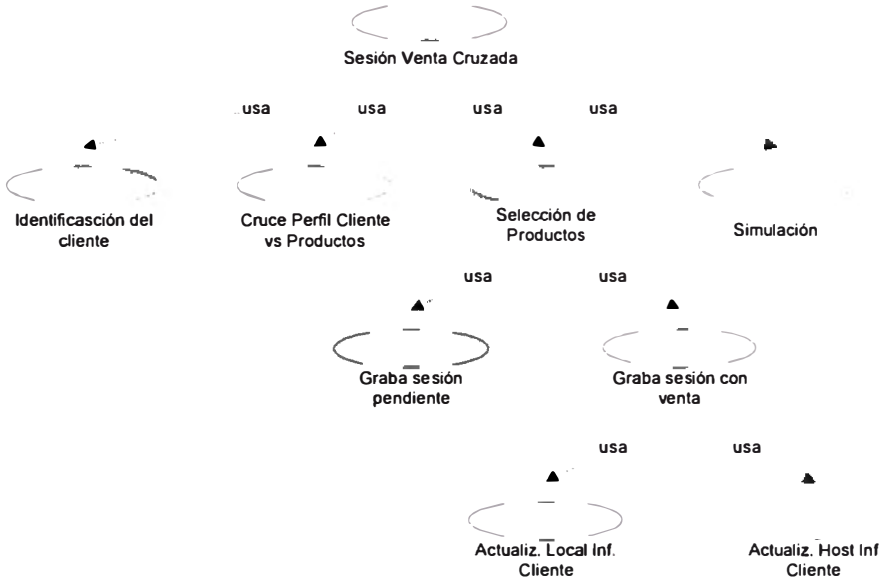
*Primer Nivel*



*Segundo Nivel (a)*




*Segundo Nivel (b)*



*Caso de Uso:*

*Cruce Perfil Cliente vs. Productos*

*(1) El funcionario de negocio selecciona comando para procesar venta cruzada*

*(2) El proceso de cruce lee características de un producto disponible*

*(3) Si las características del cliente aplican para este producto entonces se separa como producto potencial.*

*(4) Vuelve al paso (2) hasta que no queden productos por evaluar*

*(5) Muestra lista con productos a los que aplica el cliente*

*Caminos Alternativos*

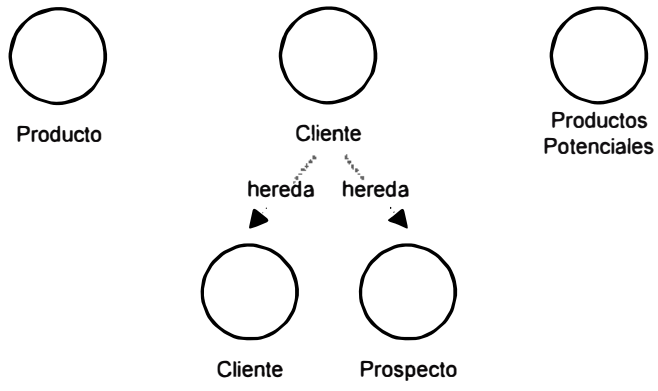
*La tabla de productos está vacía*

*En este caso se informará al funcionario de negocio que existen problemas con el sistema.*

*No aplica a ningún producto*

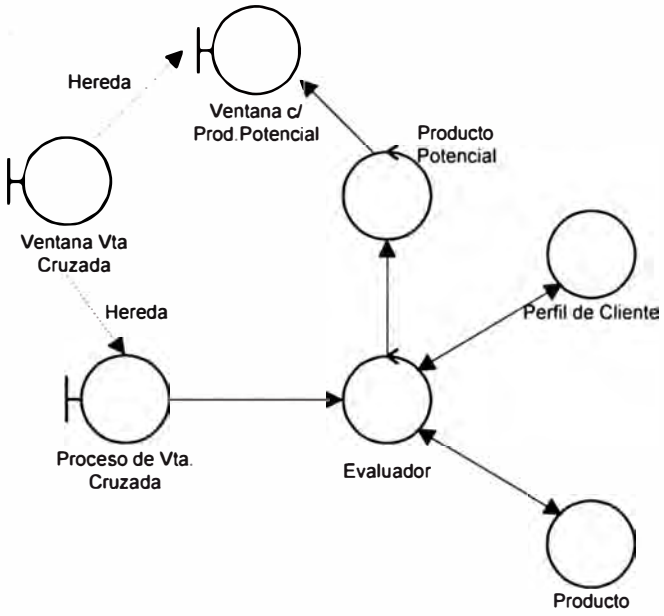
*En este caso se informará al funcionario de negocio del hecho y se sugerirá que revise datos del cliente, pues quizás no se ingresó toda su información.*

*Modelo de Objetos de Dominio del Problema*



***Modelo de Análisis***

*Modelo de objetos que soportan caso de uso: Cruce de Perfil de Cliente vs. Productos*




## 4.6 SELECCIÓN DE UNA METODOLOGÍA

De acuerdo a un último recuento que realizado existen alrededor de 25 metodologías orientadas a objetos, las mismas que pretenden cubrir todo el ciclo de desarrollo o parte de él. (Ver cuadro 4.1).

| | Metodología | Autores |
|---|---|---|
| 1 | ADM 3 | D. G. Firesmith |
| 2 | BON | Jean-Marc Nerson |
| 3 | SEOO | LBMS |
| 4 | Booch | Grady Booch |
| 5 | BOOM | Edward Berard |
| 6 | Coad-Yourdon | Peter Coad y Ed Yourdon |
| 7 | Colbert | E. Colbert |

| | | |
|---|---|---|
| 8 | de Champeaux | Dennis de Champeaux, Doug Lea y Penelope Faure |
| 9 | OSA de Embley | D.W. Embley (Hewlett-Packard) |
| 10 | EVB | John A. Jurik y Roger S. Schemenaur |
| 11 | FUSION | Derek Coleman |
| 12 | HOOD | European Space Agency |
| 13 | IBM | IBM |
| 14 | Martin y Odell | James Martin y James J. Odell |
| 15 | Merise Objet | A. Rochfeld |
| 16 | MOSES | Brian Henderson- Sellersy Julian- M. Edwards |
| 17 | OBA | K.Rubin y Adele Goldberg |
| 18 | OMT | James Rumbaugh |
| 19 | OOSE | Ivar Jacobson |
| 20 | RDD | Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener |
| 21 | Reenskaug | Reenskaug (Taskon AS, Noruega) |
| 22 | Shlaer y Mellor | Sally Shlaer y Stephen Mellor |
| 23 | SOMA | Ian Graham |
| 24 | Syntropy | Steve Cook y John Daniels |
| 25 | Winter Partners | Winter Partners |

Para las organizaciones que se inician en la adopción del paradigma de la orientación de objetos, esta cacofonía de notaciones es definitivamente confusa. La guerra de las metodologías y tendencias se hace evidente y así empezamos a confrontar las diferentes alternativas: ¿C++ , Smalltalk o Eiffel? ¿Booch, Rumbaugh o Coad/Yourdon? ¿Qué debe hacer la organización? A continuación propongo algunas consideraciones para la selección de metodologías.

⇨ **Es necesario tipificar la aplicación a desarrollar**

Al respecto, en aplicaciones sencillas como aquellas administrativas (compras, ventas, inventarios, facturación, etc) donde lo más importante es contar con la información organizada adecuadamente para poder explotarla, las metodologías sencillas y de bastante similitud con el modelo Entidad-Relación de Peter Chen son las mas convenientes. Dentro de esta categoría podemos incluir el Análisis y Diseño Orientado a Objetos de Coad y Yourdon o el de Shlaer y Mellor o el de Martin y Odell. Aplicaciones más complejas requieren de metodologías que permitan representar y modelar de manera más adecuada dicha complejidad. Aplicaciones que tienen que ver con comportamiento de objetos y donde los objetos manejan dinámicas importantes, las metodologías con énfasis en el comportamiento son las mas adecuadas. Dentro de este grupo vemos a Booch, Rumbaugh, Wirfs-Brock y Graham entre otros. Como aplicaciones de este tipo podemos señalar sistemas multimedia, sistemas expertos o basados en conocimiento, aplicaciones de simulación y de tiempo real.

A medida que estas metodologías van madurando vemos como su alcance inicial (puramente técnico) se amplía consiguiendo ser herramientas importantes dentro del nuevo enfoque de las organizaciones. Así hablamos ahora de la tecnología de objetos y su aplicación para la construcción de modelos de negocios basados en redes de agentes (objetos) cooperativos. Este enfoque viene como consecuencia del replanteamiento que la reingeniería del proceso de negocio (BPR) trae consigo. Por lo tanto dentro de este enfoque, metodologías como OOSE de Jacobson o SOMA de Graham están consideradas. Del mismo modo esta metodologías hacen uso de notaciones y herramientas como el Use Case que facilitan el desarrollo de modelos.

⇨ **Utilizar los mejores conceptos de las metodologías que se conozcan.**

Las metodologías de orientación a objetos no son recetas de cocina. Son una herramienta para la construcción del modelo de aquella parte de la realidad que deseamos entender. Por lo tanto debemos aplicar todos los conceptos que sirvan para la mayor comprensión de la realidad incluso aquellos provenientes de una metodología diferente a la escogida. Tal es el caso, por ejemplo, del uso de los Casos de Uso (Use Cases) con la metodología de Rumbaugh o de los diagramas de estado-transición de Harel en la OOA de Coad y Yourdon.

⇨ **No trabajar con una metodología sólo porque se cuenta con la herramienta de soporte para ella.**

Al respecto, la evolución de dicho tipo de herramientas permite contar con paquetes que ofrecen varias alternativas de notación y modelamiento. Esto permite utilizar la más conveniente en cada caso.

A continuación se propone una matriz que trata de resumir las caracterìsticas de los sistemas y cómo las metodologías revisadas atacan este aspecto del dominio del problema. Las características que se enumeran representan algunos de los problemas o dificultades que encontramos en el desarrollo del análisis y diseño orientado a objetos y cómo éstas metodologías nos ayudan a modelar dicha complejidad. En primer lugar se define el problema del modelamiento de relaciones todo-parte y especialización generalización que son el primer nivel de abstración al momento de ientificar los objetos y las relaciones entre ellos. El manejo de estados y su modelamiento es otro de los conceptos con los que nos encontramos en algunos sistemas, sobre todo en aquellos donde el objeto es sometido a procesos que cambian su comportamiento posterior, para superar esta complejidad las metodologías en mayor o menor manera implementan esquemas de modelamiento fundamentalmente basados en diagramas de estado-transición. El concepto de persistencia se refiere a que algunos objetos de nuestro sistema requieren de ser almacenados o recuperados para su tratamiento

posterior, es en este sentido que la metodología OOSE implementa por ejemplo diferentes tipos de objetos para representar al objeto dentro del proceso y el objeto alamcenado en una base de datos. Del mismo modo las consideraciones de análisis de requerimientos y diseño de interfaces usuario no son tratadas por todas las metodologías de manera similar. El comportamiento alternativo de los objetos está asociado a su capacidad de manejar estados y son características que permiten por ejemplo el modelamiento de sistemas expertos o que utilicen lñógica difusa. La complejidad en general representa un problema para las metodologías pues sus ténicas y nomenclaturas deben buscar simplificar y focalizar la atención en la parte del problema en estudio. En este sentido, el uso de temas, frames, manejo de interfaces y otras técnicas para ocultar complejidad de aquella parte que no se encuentra modelando, son todavía problemas latentes en las metodologías.

| | **Coad & Yourdon** | **Martin & Odell** | **OMT de Rumbaugh** | **OOSE de Jacobson** |
|---|---|---|---|---|
| **Estructura de Objetos todo-parte (bill of materials)** | Bien | Regular | Regular | Regular |
| **Estructuras de Objetos generalización-especialización** | Bien | Bien | Bien | Bien |
| **Atributos de Objetos del tipo Estados** | Regular | Bien | Bien | Bien |
| **Objetos Persistentes** | Regular | Regular | Regular | Bien |
| **Modelamiento de Interfaces Usuario** | Regular | Regular | Regular | Bien |
| **Modelamiento Empresarial** | Regular | Regular | Regular | Regular |
| **Modelamiento de objetos con comportamiento alternativo** | Regular | Regular | Regular | Bien |
| **Modelamiento de la Complejidad** | Regular | Regular | Bien | Bien |
| **Modelamiento de Interfaces con otros sistemas** | Regular | Regular | Regular | Bien |

De la matriz anterior y aplicándola a nuestro ejemplo de Ventra Curzada de SARA BANK, obtenemos el siguiente cuadro:

| CARACTERISTICAS VENTA CRUZADA | Coad & Yourdon | Martin & Odell | OMT de Rumbaugh | OOSE de Jacobson |
|---|---|---|---|---|
| Estructura de Objetos todo-parte = SI,POCO | ★★★ | ★★★ | ★★★ | ★★★ |
| Estructuras de Objetos generalización-especialización = SI | ★★★ | ★★★ | ★★★ | ★★★ |
| Atributos de Objetos del tipo Estados = SI | ★ | ★★ | ★★ | ★★★ |
| Soporte aObjetos Persistentes = SI | ★ | ★★ | ★★ | ★★★ |
| Modelamiento de Interfaces Usuario= SI | ★ | ★ | ★ | ★★ |
| Modelamiento Empresarial= NO | - | - | - | - |
| Modelamiento de objetos con comportamiento alternativo = SI,POCO | ★★ | ★★★ | ★★★ | ★★★ |
| Modelamiento de la Complejidad=SI,POCO | ★★★ | ★★★ | ★★★ | ★★★ |
| Modelamiento de Interfaces con otros sistemas=SI | ★★ | ★★ | ★★ | ★★★ |

Como se puede apreciar en el cuadro anterior, las diferencias no son muy grandes entre una u otra metodología. Sin embargo, vemos que existen facilidades en el modelamiento del espacio problema que con la metodología OMT o la metodología OOSE son mejor representados.

# 5. COMENTARIOS FINALES.

## 5.1 BENEFICIOS Y COSTOS DE LA TECNOLOGIA DE ORIENTACION A OBJETOS

### 5.1.1 Ventajas y Beneficios.

Los beneficios que brindan los métodos y técnicas de programación orientadas a objetos, son muestra de un proceso de evolución en la concepción del desarrollo de aplicaciones, como lo fueron el diseño y programación estructurados en su momento. Veamos a continuación algunos beneficios de esta tecnología:

- ☑ Un adecuado diseño en los sistemas orientados a objetos son el sustento para sistemas a ser ensamblados en grandes magnitudes a partir de módulos reutilizables, obteniendo una alta productividad.
- ☑ La reutilización de clases que han sido probadas en proyectos anteriores conducen a sistemas de mayor calidad, se confrontan con los requerimientos del sistema de una manera más sencilla y contienen menos errores o fallas.
- ☑ La tecnología de orientación a objetos, y en especial la herencia, hacen posible la definición de módulos que son funcionalmente incompletos, pero extendibles sin necesidad de ser rehechos.
- ☑ La comunicación entre los objetos se realiza a través de la convención de envío de mansajes. Esta manera de trabajar permite la interfaz entre elementos del sistema e incluso con sistemas externos de manera sencilla. Facilita la definición y construcción de interfaces gráficas de usuario y sistemas distribuídos.
- ☑ El particionamiento de los sistemas en tipos de objetos encapsulados ayuda en el problema de *escalabilidad* (scalability), es decir, la partición del sistema en componentes de un sistema al momento del

análisis y diseño; y su integración al momento de la construcción del mismo. Al igual que con el desarrollo convencional de sistemas, el esfuerzo tiende a crecer exponencialmente con el tamaño y la complejidad del proyecto.

- ☑ La forma de distribuir el trabajo con las metodologías orientadas a objetos es mucho más natural. El análisis y deseño que divide un dominio, formado por componentes del mundo real en vistas orientadas a la solución, es mucho más natural que una descomposición funcional top-down o bottom-up.
- ☑ El ocultamiento de información permite la construcción de sistemas más seguros.
- ☑ La orientación a objetos es una herramienta en el manejo de conplejidad.
- ☑ Los problemas de mantenimiento y evolución de los sistemas son mitigados por la fuerte descomposición en componentes e interfaces uniformes de objetos que contiene el sistema.
- ☑ Los sistemas orientados a objetos son capaces de capturar más significado de una aplicación: su semántica. Dado que los métodos orientados a objetos están involucrados con los sistemas de modelamiento pueden ser usados para llevar a cabo modelamiento de escenarios y facilitar los cambios dentro del negocio. Esta característica proporciona la posibilidad de realizar ingeniería reversa y una retrospectiva hacia los requerimientos.
- ☑ Algunas aplicaciones no han sido exitosas con otras tecnologías, y la tecnología de objetos parece ser la única que se adapta para construirlas eficientemente. Ejemplos a la vista: interfaces gráficas de usuario, sistemas distribuídos y sistemas de flujo de trabajo.

### 5.1.2 Problemas de la Tecnología de Objetos.

La precaución es necesaria cuando queremos utilizar una nueva tecnología. La tecnología de objetos no es una panacea y se debe establecer claramente los componentes que se pueden aprovechar y cuáles no se encuentran lo sufucientemente maduros. A continuación se ofrecen algunos argumentos en contra de esta tecnología:

- ☑ Los proyectos con tiempos muy ajustados generalmente no ofrecen reusabilidad de sus componentes. Las empresas y los gerentes de proyectos generalmente se dejan llevar por el corto plazo y la escasez de recursos, asignando tiempos a las tareas que impiden la construcción de componentes del sistema que en futuro puedan ser reutilizables.
- ☑ Las bibliotecas de objetos tienen que ser construídas y difundidas. En la actualidad exixten pocas librerías de objetos disponibles. Un esfuerzo mayor en desarrollo y difusión de bibliotecas estándares de objetos es necesaria.
- ☑ Existe un creciente desarrollo y nuevos productos que hacen difícil la elección de herramientas o lenguajes para el desarrollo.
- ☑ Cuando la herencia u otras construcciones semánticas son utilizadas, se requiere de un cuidadoso diseño y control para no comprometer la reutilización.
- ☑ Temas como persistencia, concurrencia y performance tienen por el momento que esperar el beneficio del concenso a nivel de todos los constructores y otros implicados. Por persistencia entendemos al hecho que, los objetos, almacenados en disco, continúen sin alteración a lo largo de las diferentes ejecuciones. Al respecto, las bases de datos y los lenguajes de programación requieren ponerse de acuerdo.

- ☑ La topología de mensajes es muy importante, es muy fácil escribir malos sistemas orientados a objetos. Para conocer el comportamiento del sistema es necesario rastrear el paso de mensajes, esto aún posee muchas dificultades y continúa siendo un defecto de la tecnología de objetos..
- ☑ Escribir componentes reutilizables costaría mucho más esfuerzo que lo que tomaría de acuerdo a estimaciones convencionales.
- ☑ La administración de librerías de componentes es difícil y costosa.
- ☑ Se requiere de un cambio de cultura en los equipos de desarrollo, y eso es lo que más odian.
- ☑ Existen inevitables costos asociados con el entrenamiento, reeducación, hardware y software.
- ☑ La tecnología de objetos no es la última palabra en la ingeniería de software. Al respecto, quisiera resaltar que no existe la llamada *bala de plata ("silver bullet")* como la manifestó Brooks en 1986 en su famoso artículo. La necesidad de hablar de la tecnología de objetos como un medio, un paso, una mejora, en la manera de diseñar y construir sistemas, y no en un fin o meta, creo que es el punto de partida para iniciar su utilización.

## 5.2 LAS TENDENCIAS EN EL DESARROLLO DE APLICACIONES Y LA TECNOLOGIA DE OBJETOS.

Está demás decir que el desarrollo de aplicaciones es una de las áreas del conocimiento con mayor evolución. Revisemos las últimas dos décadas de los sistemas de información.

A fines de los setentas:

- Procesos centralizados: La información reside en los grandes computadores (mainframes).

- Grandes empresas: La informatización llega sólo a las grandes empresas y corporaciones (bancos, empresas estatales, etc.)
- Elites Tecnológicas: Sólo un grupo de personas (aquellas del centro de cómputo) controla y procesa la información de la empresa.
- Desarrollos Largos: Los tiempos de desarrollo son prolongados debido a la poca facilidad que ofrecen las herramientas y metodologías.
- Dificultad de acceso a datos: Los usuarios no cuentan con la información a la mano, cada vez que se requiere un reporte o información complementaria es necesario solicitarlo al area de sistemas.

A fines de los ochenta:

- Procesos centralizados con islas de información: La información sigue residiendo en los grandes conputadores, pero aparecen las redes. Estas redes se convierten en islas de información pues generan su información propia, la misma que no es fácil de intercambiar con el computador central.
- Crecimiento acelerado de uso de computadoras personales y su integración en redes: Esta es una reacción natural de todas las areas de la empresa ante la ineficiencia de los centros de computo, asimismo, para procesar información no estructurada y propia del trabajo diario (hojas de cálculo, procesador de textos, etc).
- Desarrollos más cortos: Aparecen herramientas y lenguajes de desarrollo más versátiles. En mainframes y minis: los 4GLs como MANTIS, CSP, Informix, etc. En Pcs: Basic, dBase, FoxPro, Pascal, Clipper y Lenguaje C.
- Información al Alcance de más usuarios: las redes permiten compartir información, los mainframes amplían su cobertura con terminales a más áreas de la empresa.

La presentación de la situación de las últimas dos décadas no pretende ser exhaustiva. Sólo busca resaltar algunos aspectos de la evolución de las tendencias en el uso de los sistemas de información.

En la actualidad nadie duda de la necesidad de contar con aplicaciones y sistemas de información para facilitar el trabajo dentro de las empresas. Es así que los sistemas de información administrativos y de gestión, ya no son la preocupación de nuestras empresas. Esto significa que existe la necesidad de desarrollar nuevos tipos de aplicaciones y por otro lado mejorar los existentes.

Los problemas que tenemos que afrontar en la actualidad son más complejos que en el pasado. Para la construcción de aplicaciones modernas tenemos que tener en cuenta nuevos conceptos:

- Arquitectura Cliente/Servidor

  Tiene que ver con la distribución del proceso de una aplicación de modo que todo el proceso no se realice en un sólo computador.

- Descentralización de la Información

  Es necesario contar con la información descentralizada para evitar cargar el computador central. Aquí se enmarcan conceptos como Bases de Datos Distribuídas, Replicación de Datos y Propagación de Datos.

- Interoperabilidad

  Este concepto se refiere a la necesidad de desarrollar aplicaciones que conversen con otras aplicaciones residentes en otras plataformas de hardware y software.

- Internetworking

  En la actualidad es necesario que las redes dentro de las empresas se comuniquen entre sí, y al mismo tiempo que éstas se comuniquen con los agentes externos con los que interactúan (clientes, proveedores, etc).

- Graphical User Interface (GUI)

  La presentación (interfaz) que proveen los sistemas es cada vez más sofisticada. Este concepto fue introducido por Apple con sus computadoras Macintosh. Incorpora el uso del mouse y de controles

gráficos dentro del aplicativo (botones, scroll bars, íconos, radio buttons, etc).

- Sistemas Multimedia

  Los sistemas actuales deben incluir facilidades para intercalar información textual, sonido e imágenes fijas y animadas.

- Aplicaciones para flujo de trabajo (Workflow)

  El concepto de workflow va mas alla del procesamiento de la información en cada etapa, consiste en la identificación del proceso en sí mismo y su automatización. Asociado a este concepto tenemos el de groupware.

- Sistemas Expertos

  La inteligencia artificial es uno de las tecnologías que por el momento no ha visto aún una acogida por el ambiente informático. Sin embargo, junto con las redes neurales espera su turno en el desarrollo de sistemas por el gran potencial que ofrece para las aplicaciones.

- Lógica Difusa (Fuzzy Logic)

  Dentro de la inteligencia artificial, esta es una de las áreas de mayor avance por la necesidad de contar con técnicas que expresen de manera más natural el mundo real.

- Sistemas de Información Geográficos (GIS)

  Estos sistemas automatizan la cartografía manual y basada en papel. Se concentra en la captura , almacenamiento y presentación de la información de data planar y espacial (bidimencional o tridimencional).

El gran reto de desarrollo aplicaciones incluyendo estos conceptos está latente, y es la tecnología de objetos la gran herramienta para superar este reto. ¿Porqué decimos esto?. A continuación veamos en que fundamentamos esta afirmación. La tecnología de objetos ya se encuentra presente en muchos de los conceptos que hemos presentado. El concepto de arquitectura Cliente/Servidor es una

visión aplicativa de dos objetos que se comunican mediante mensajes, que tienen información propia e interfaces de comunicación. Las arquitecturas C/S son consecuencia natural del proceso de evolución de las aplicaciones hacia estructuras conformadas por agregados de software con identidad y coherencia por sí mismos, con capacidad de comunicación con otros componentes de otras aplicaciones. Esto es lo que se viene a denominar en terminología Microsoft: Desarrollo de Aplicaciones por Componentes. En este sentido, el futuro del desarrollo de aplicaciones se basa en el hecho de construir objetos que se encuentran distribuídos a lo largo de nuestras redes empresariales, los cuales se comunican entre sí para obtener los resultados que esperamos.

Por lo antes mencionado, es necesario contruir arquitecturas de comunicación entre nuestras redes de modo que exista la infraestructura de hardware necesaria. Esta es la orientación de los conceptos de internetworking.

Respecto a los mecanismos de comunicación de objetos que se encuentran alrededor de nuestras redes, es necesario señalar que en la actualidad ya existen estándares de comunicación de objetos. Estos son CORBA (Common Object Request Broker Architecture) y OLE/COM (Object Linking and Embedding / Component Object Model). Si bien OLE/COM no es un estándar, pues ha sido desarrollado por Microsoft, es el más difundido y podemos considerarlo como un estándar de facto. CORBA es un estándar de comunicación que reúne a más de 350 empresas. Esta es la razón que no esté aún listo y OLE/COM le haya tomado la delantera.

Estos estándares buscan ofrecer la posibilidad de contar con objetos distribuídos a lo largo de toda la red que interactúan entre sí, sin importar con qué plataforma nos estemos comunicando. De esta manera proveemos a nuestras soluciones esta característica de las aplicaciones del nuevo siglo: interoperabilidad.

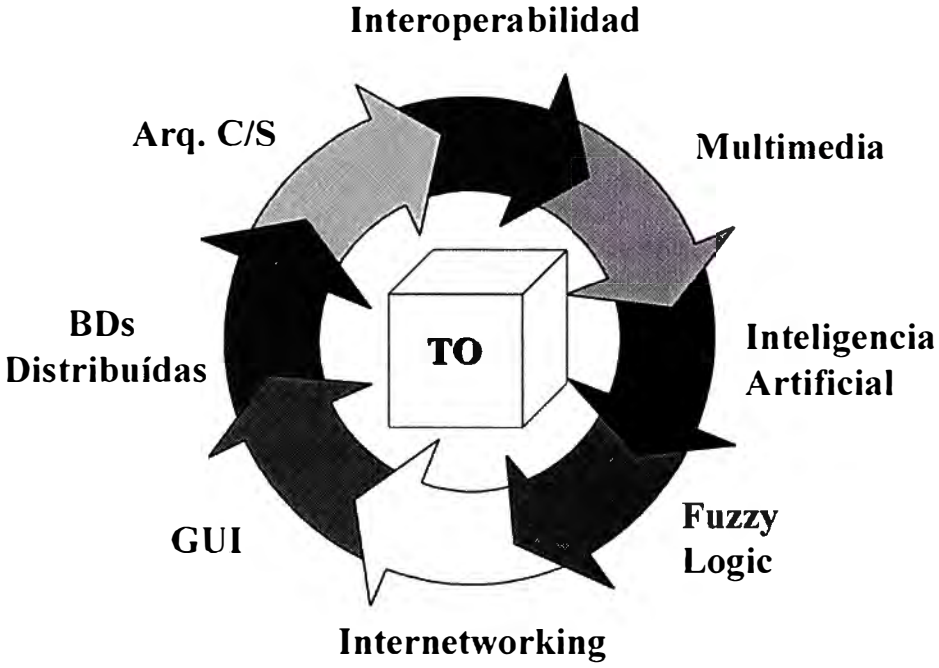
Respecto a GUI sólo diremos que en la actualidad todas las herramientas para desarrollar front-ends gráficos son herramientas que incluyen objetos de interfaz usuaria con atributos cambiables y métodos vacíos que corresponden a la interfaz de los eventos. Este es el caso de herramientas como Visual Basic, Power Builder y Delphi. Otro de los lenguajes con características de orientación de objetos es JAVA. Todos ellos incluyen controles de interfaz.

Los sistemas multimedia son cada vez más populares y de uso masivo. Los avances en este sentido se deben al abaratamiento de los componentes de hardware que los soportan y permiten ofrecer aplicaciones para usuarios finales más amigables y sencillas. Los componentes multimedia dentro de nuestras aplicaciones son piezas de software completamente encapsuladas. El lenguaje JAVA, por ejemplo, permite la creación de objetos multimedia de manera sencilla para el desarrollo de aplicaciones de internet e intranet.

Otro de los campos en el que los avances de la tecnología de información se ha desarrollado, por lo menos en el aspecto conceptual, es el de las aplicaciones de flujo de trabajo. El diseño de workflows y herramientas para groupware ofrecen una gran posibilidad para la automatización de aquellas áreas de las organizaciones aún no automatizadas. En ese sentido, la tecnología de objetos ofrece una posibilidad clara para dicho desarrollo. Dentro del desarrollo de aplicaciones de workflow existen dos aspectos: el aspecto de procesamiento de la información y el del flujo del proceso. Con relación al primer aspecto podemos decir que se refiere al tratamiento de la información de manera tradicional: los datos de la transacción, la manera de almacenarlos y presentarlos. El segundo aspecto es por el contrario novedoso: el diseño e implementación del modelo de proceso dentro de la organización para un seguimiento del proceso. Es en este segundo aspecto en el que la tecnología de objetos aporta su enfoque. Los procesos y su modelamiento representan un

ejemplo claro de aplicación del modelo de orientación a objetos. Los procesos, sus estados, los agentes del proceso, las condiciones de activación de eventos, todo apunta a una visión desde el punto de vista de objetos. Así lo han visto algunas plataformas de desarrollo de aplicaciones de este tipo como Flowmark de IBM. Esta herramienta almacena el modelo del proceso en bases de datos de conocimiento ObjectStore. De igual modo, se almacenan las diferentes ocurrencias del modelo que existen en producción, cada una de las cuales en una instancia de la clase (modelo). En este caso la persistencia de la instancia se hace a través de un SGBDOO.

La inteligencia artificial tiene en la tecnología de objetos una posibilidad de ver efectivamente materializada su disponibilidad. Los conceptos de lógica difusa son una muestra de ello. La implementación de aplicaciones que hacen uso de lógica difusa está basada en tecnología de objetos. Mediante la TO se construyen objetos difusos que almacenan la operatividad y valores de dicha lógica para luego construir procesos de decisión basados en dichos objetos.

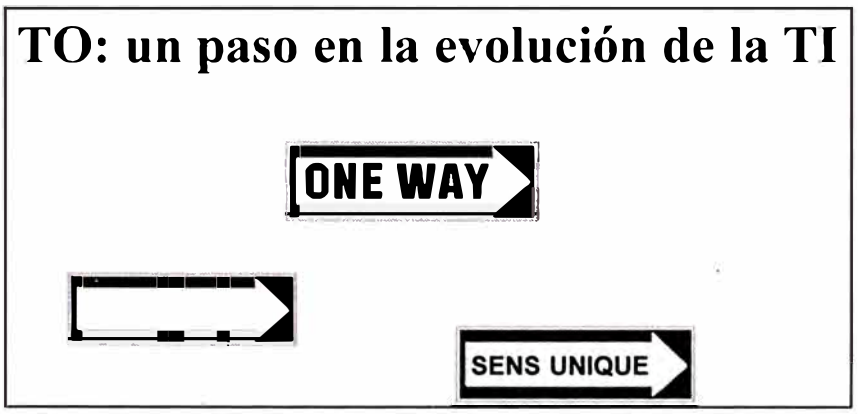


Lo que hemos querido mostrar con este repaso de las tendencias de la tecnología de sistemas de información es justamente que dicha tendencia está orientada hacia la tecnología de objetos. Por lo tanto debemos prepararnos para asimilarla e integrar su paradigma en el diseño de nuestras aplicaciones.

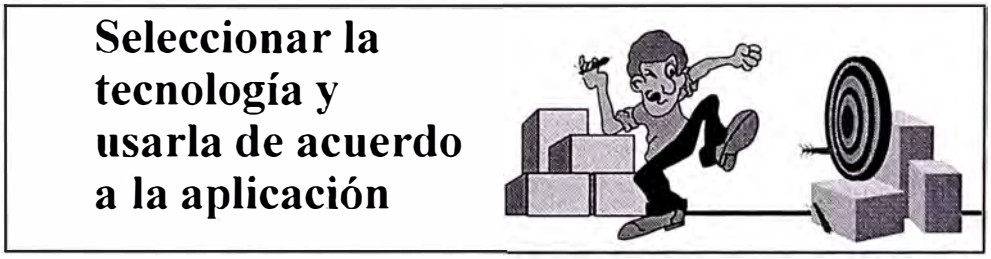
## 5.3 REFLEXIONES FINALES

A continuación pasamos a resumir algunos de los planteamientos que esta tesis propone:

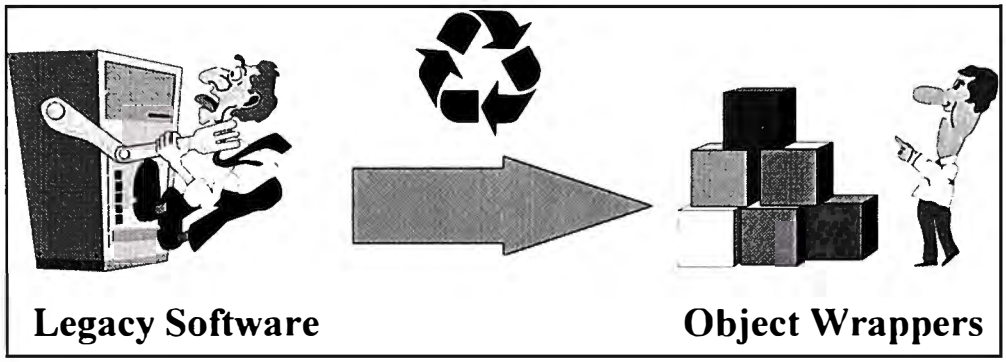
- La necesidad de adoptar la tecnología de objetos por la comunidad informática es vital y no es más que el siguiente paso en la evolución del desarrollo de aplicaciones.



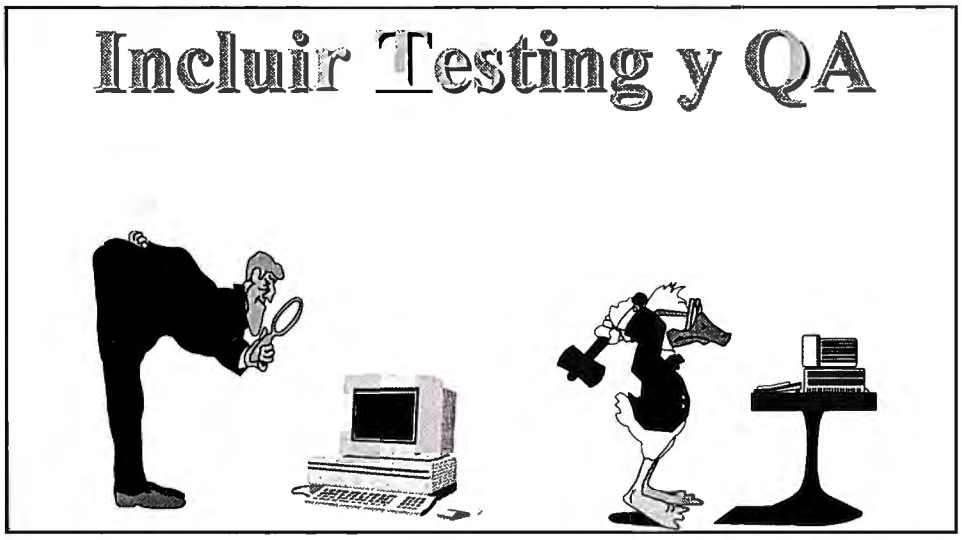

- La tecnología y los nuevos retos de los sistemas de información impiden que las organizaciones que desarrollan sus sistemas y aplicaciones de manera "artesanal" puedan ser competitivos en un futuro cercano. La tecnología de objetos "per se" no es la solución a los problemas. Es necesario tomar en cuenta los siguientes elementos al momento de migrar hacia la TO.
  - ★ Hacer uso selectivo de la tecnología dependiendo de la aplicación, pues no siempre es conveniente su uso.



★ Evaluar el impacto de la adopción de la TO en el *legacy software*, determinar la mejor manera de convivencia. Hacer uso de los wrappers.




★ Aplicar criterios de calidad a todo el proceso de desarrollo e incluir dentro de los equipos de desarrollo personal especializado en *Testing* y *Quality Assurance*. Recordar que es muy fácil hacer sistemas orientados a objetos de pésima calidad.

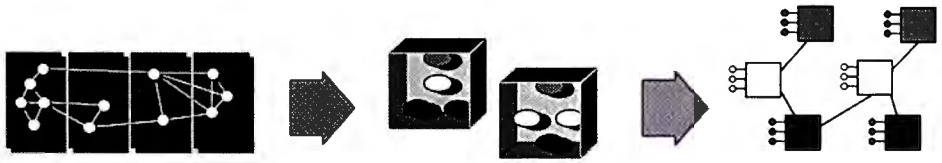



★ La migración hacia la TO es un proceso. Iniciar a algunos en esta aventura, los mismos que serán luego los que enseñen a los demás difundiendo sus ventajas y limitaciones. Quizás el uso de herramientas visuales que usan objetos visuales nos permita ir entrando en esta tecnología. Una organización que permita el desarrollo eficiente de aplicaciones en esta tecnología es necesaria.

# Entrenar a los "mentors"

★ Las metodologías son herramientas para el análisis y diseño de aplicaciones, debemos tratarlas como tales. Lo más importante es comprender las necesidades del usuario y el entorno problema. Un modelamiento del entorno problema incluyendo los componentes y agentes del mismo nos permitirán desarrollar con la TO un proceso de desarrollo "sin costuras" (seamless process).

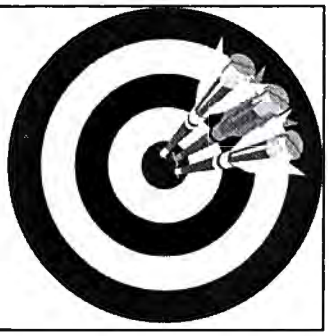
## Seamless Process




➢ La madurez en el desarrollo de aplicaciones se refleja en una adecuada administración del proyecto y una disciplina en la ejecución del mismo. El seguimiento permitirá alcanzar los objetivos de alcance completo, plazo cumplido y márgenes presupuestados.

Un adecuado control de proyecto permitirá cumplir con las tres restricciones:

- •Alcance Funcional
- •Costo y Esfuerzo
- •Plazo



## BIBLIOGRAFIA

### *Libros y Manuales*


Object-Oriented Software Engineering - A Use Case Dirven Approach
*Ivar Jacobson*
Adisson-Wesley, 1992

Object-Oriented Analysis and Design
*James Martin y James Odell*
Prentice Hall, 1992

Object-Oriented Methods
*Ian Graham*
McGraw Hill, 1992

Object-Oriented Modeling and Design
*James Rumbaugh, Michael Blaha, William Prelermani, Frederick Eddy, William Lorensen*
Prentice Hall, 1991

Développement Orienté Objet
*Dominique Vauquier*
Editions Eyrolles, 1993

La démarche objet - Concept et outils
*Max Bouché*
AFNOR, 1994

Discovering SMALLTALK
*Wilf Lalonde*
Benjamin/Cummings Publishing Company, 1994

Object-Oriented Analysis
*Peter Coad, Edward Yourdon*
Prentice Hall, 1991

Object-Oriented Design
*Peter Coad, Edward Yourdon*
Prentice Hall, 1992

An Introduction to Object-Oriented Programming and Smalltalk
*Lewis J. Pinson, Richard Wiener*
Addison-Wesley, 1988

Solutions Development Discipline
Microsoft Press, 1994
Smalltalk/V 286 - Tutorial and Reference Manual
Digitalk Corporation, 1989

Curso de Programación C++
*Francisco Javier Ceballos*
Ediciones RA MA, 1991

***Revistas y otras fuentes e información***

Microsoft Solutions Framework v2.0 (CD)
Microsoft Corporation, 1996

Microsoft Tech Ed 96 (CD)
Microsoft Corporation, 1995

Brooks, Jr. "No Silver Bullet: Essence and Accidents of Software Engineering", *IEEE Computer*, Vol 20, No.4, Apr. 1987

Stuart Frost, "Modeling for RDBMS legacy", *SIGS Object Magazine*, 4(5) September 1994

Edmund C. Arranga y Frank P. Coyle, "Object Cobol", *SIGS Object Magazine*, 4(5) September 1994

David M. Bulman, "An Object-Based Development Model", *Computer Language*, 6(8) August 1989

Darcy Harrison, "Building Financial Software with O-O Technology", *Objects in Europe*, March 1995

# ANEXOS

## ANEXO 1 : SARA BANK® Y SU ENTORNO DE APLICACION

## INTRODUCCION AL PRODUCTO SARA BANK®

SARA BANK® es un sistema orientado a la automatización de agencias bancarias en sus labores de tratamiento transaccional y batch de operaciones. Incluye la gestión de los diversos componentes de la problemática en agencias, como son: las transacciones, la gestión de cajas, la contabilidad en agencias, diseño de refrendos, la seguridad y control de accesos, la gestión y visualización de firmas, la difusión de procedimientos operativos, etc.

SARA BANK® es una solución que involucra un conjunto de productos y servicios que tienen por finalidad solucionar integralmente la gestión de información en las agencias. La gestión integral consiste en atender los diferentes frentes de atención de una agencia. Estos son:

**Ventanilla (Front-office)**

Este frente es el que recibe gran volumen de clientes y no clientes, y tiene como función principal atender transacciones financieras de los productos y servicios del banco; si bien este frente debe estar preparado para procesar gran volumen de datos, poco a poco se están integrando a este frente, funciones de promoción **y o** facilidades para proporcionar información a los clientes y no clientes. Generalmente atiende transacciones de los productos y servicios tradicionales del banco, como son las cuentas corrientes, ahorros, cartera, cambios.

**Asistencia Operativa (Back-office)**

Referida más a un proceso interno de procesamiento de transacciones e información, normalmente en ella encontramos funciones como captura de datos en el origen y/o complementarios a las operaciones de ventanilla, autorizaciones, cuadre y

contabilización de operaciones, emisión distribuída de informes y documentos, intercambio de información con clientes que involucre cierta gestión con otras entidades internas de la institución, etc. Se caracteriza por atender transacciones sin clientes.

**Plataforma de Clientes (Retail Platform)**

Tiene como objetivo principal la promoción y venta de productos y servicios financieros, teniendo en cuenta como mercado objetivo a las personas naturales. El ejecutivo comercial se convierte en el soporte financiero para los clientes personales. Como funciones en este frente podemos mencionar entregar información completa de la situación del cliente y sus informes comerciales, generación de informes para la aprobación de créditos, impresión de documentos que respaldan los productos (pagarés, contratos, etc.), simulaciones de créditos, acceso a una completa información del portafolio de productos que ofrece la institución, la venta cruzada de productos y servicios, etc.

**Plataforma Corporativa (Corporate Platform)**

Este frente tiene por objeto brindar una gama de herramientas para facilitar al Ejecutivo de Cuentas que atiende al cliente empresa información completa y oportuna. Entra las principales funciones que podemos mencionar tenemos: ingreso y actualización financiera de clientes, emisión de informes para niveles superiores de decisión, manejo de información de línea de crédito, traspaso de fondos, manejo relativo a las garantías, soporte al proceso de otorgamiento de créditos, comité electrónico en línea, etc.

**Mesón de Atención al Cliente (Front Desk)**

Representa un frente de apoyo en la cadena de producción de la sucursal con el propósito de optimizar la calidad de atención junto a la manutención y atención de productos contratados por los clientes. Está orientada a entregar servicios de información en las sucursales. Como funciones principales podemos mencionar el intercambio de documentación con el cliente (recepción de correspondencia y solicitudes, entrega de

estados de cuenta), consulta de saldos y estado de operaciones, reposición de chequeras y tarjetas de crédito, órdenes de no pago de documentos, etc.

**Gestión Administrativa (Branch Administration)**

Orientada a apoyar en el proceso de toma de decisiones en cuanto al ámbito de operación de la sucursal. Como funciones podemos mencionar la consulta de información estadística de operaciones de la agencia, consulta de la posición de caja, consulta de nivel de atención (control de colas), control de pago a proveedores, administración de circulares y procedimientos de operación, etc.

**Autoservicio (Self service)**

Este frente de atención se caracteriza principalmente por la ausencia de empleados del banco en el punto de atención y aprovecha la participación directa del cliente para obtener un determinado servicio. Los sistemas de autoservicio mejoran la imagen de la institución y mejoran la capacidad de servicio al cliente. Entre las funciones principales tenemos la consulta de información de promoción, operaciones que no involucren manejo de efectivo tales como: consultas de saldos, transferencia de fondos, pago de servicios con cargo en cuenta, etc.

## EL AMBIENTE TRANSACCIONAL DE SARA BANK

Una transacción SARA BANK es un conjunto de actividades elementales que se realizan para completar una operación bancaria.

Veamos el ejemplo de una pago de cheque (simplificado). Como se puede apreciar en la figura A.1, podemos identificar cuatro instancias: el cliente, la ventanilla, el computador central y el supervisor o jefe de agencia.

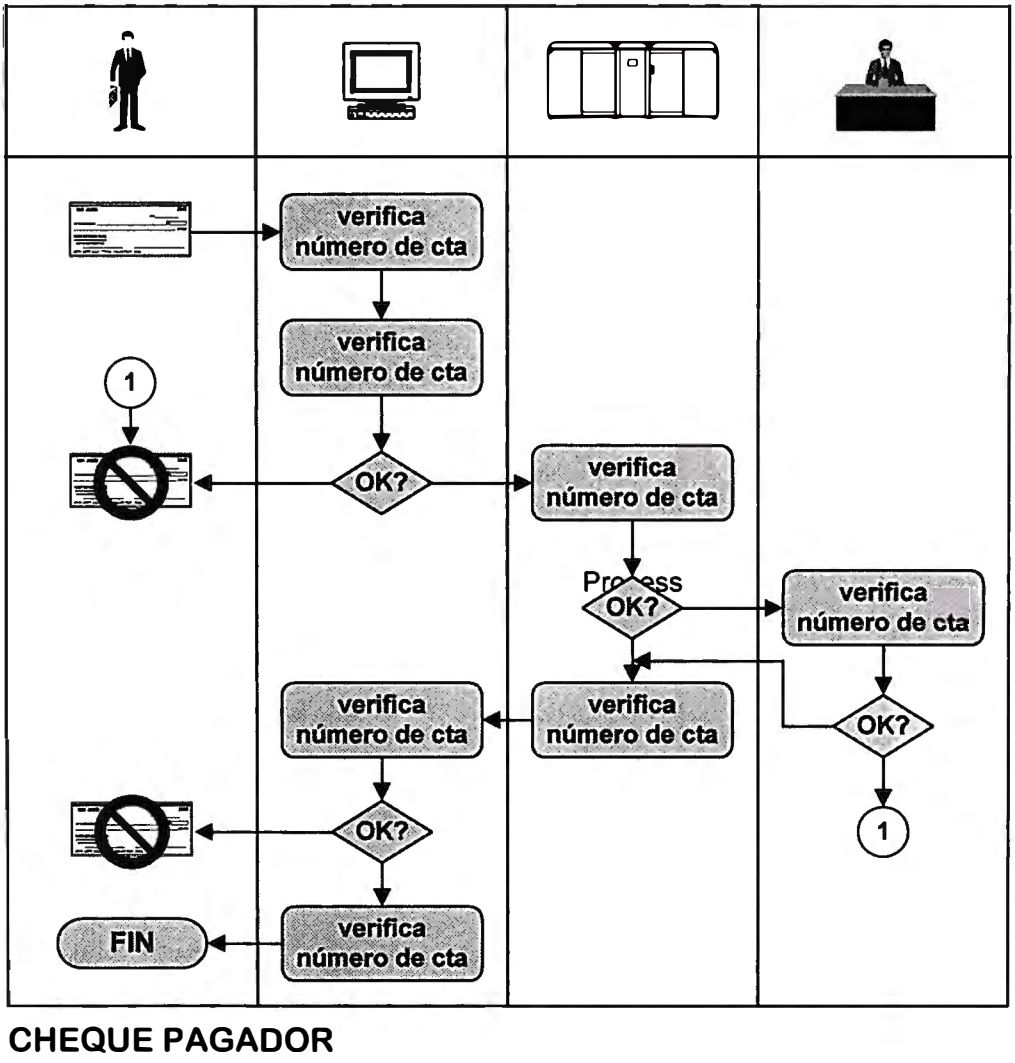



**Figura A1**. Transacción de Pago de Cheque.

El flujo de la operación se describe a continuación:

1. El cliente presenta el cheque en la ventanilla
2. El cajero verifica el número de cuenta
3. El cajero verifica si el cheque tiene algún impedimento de pago. Si es el caso devuelve el cheque al cliente, sino continúa con el paso siguiente.
4. Se verifica que el saldo de la cuenta sea superior o igual al monto del retiro. Si no hay suficiente dinero en la cuenta, la transacción para la autorización del sobregiro por el supervisor. Si hay suficiente dinero, pasa al paso 6.

5. El supervisor toma la decisión de aprobar o rechazar el sobregiro. Si lo rechaza, se devuelve el cheque al cliente. En caso contrario (aprueba el sobregiro), se continúa con el paso siguiente.
6. Se procede a rebajarle el saldo a la cuenta giradora del cheque.
7. Se verifica las firmas, si están correctas se continúa, sino se devuelve el cheque al cliente.
8. Se procede al pago.
9. Se contabiliza la operación.

SARA BANK facilita las labores de los tres últimos actores en el proceso de ejecución de la transacción.

## ¿COMO SE CONSTRUYE UNA TRANSACCION?

Para SARA Bank, una transacción está compuesta por fases y una vez cada fase está compuesta de funciones. Una fase es un conjunto de actividades elementales (implementadas a través de pequeños programas llamados funciones SARA). El criterio de agrupamiento puede ser contable (una fase para el crédito y otra para el haber), funcional (fases opcionales y/o alternativas), manejo multimoneda (una fase para cada moneda), etc. Una función es un programa que tiene por objetivo realizar una actividad elemental específica. Se trata de un programa pequeño que actualiza la caja, verifica el saldo de una cuenta, graba en el diario electrónico, etc.

A continuación, veamos un ejemplo de estructura de transacción SARA BANK para una Nota de Débito.

Esta operación consiste en rebajar el saldo de la cuenta y cargar el mismo importe a algún concepto contable. El criterio contable será el que defina en este caso la división de la transacción en dos fases: una para el debe y otra para el haber. De esta manera, en la primera fase se capturarán todos los datos de la operación excepto la cuenta contable a abonar, que será

capturada en la segunda fase. La estructura de la transacción es como se muestra en la figura A.2.

Si en lugar de una nota de débito, se tratara de un pago de cheque, la estructura sería bastante similar, Las diferencias serían: verificar las firmas que aparecen en el cheque, eliminar la captura de la cuenta contable (pues se trata de una operación con caja), añadir la actualización de caja. Ver la figura A.3.

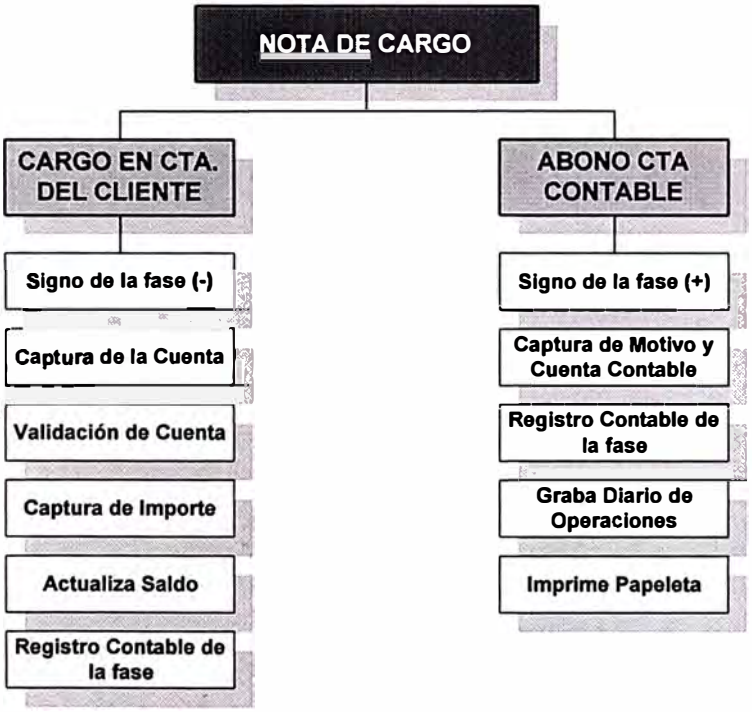



**Figura A2.**

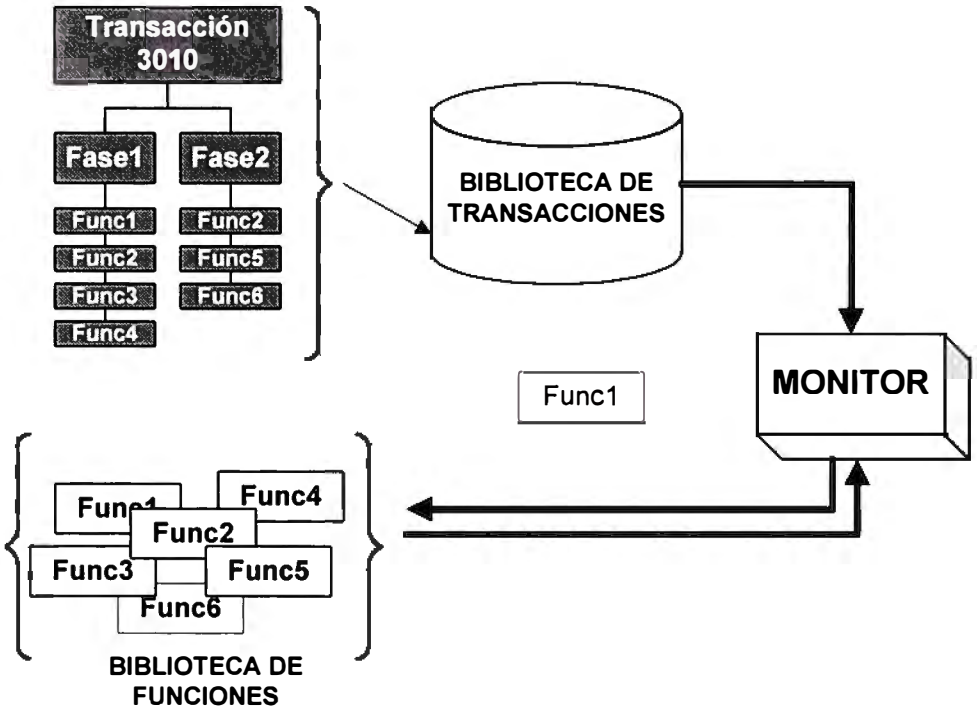

**Figura A3.**

## ¿COMO EJECUTAMOS UNA TRANSACCION?

La ejecución de una transacción SARA BANK sigue los siguientes pasos (Ver figura A.4):

1. El programa Monitor lee el esquema de la transacción seleccionada de la biblioteca de transacciones.
2. El programa Monitor actualiza los parámetros y variables correspondientes a la transacción y verifica la autorización de acceso.
3. Cada una de las funciones del esquema es ejecutada en la secuencia preestablecida por el esquema de la transacción. Cada función envía al programa Monitor un código de retorno que permite determinar si hubo error o no en su ejecución.
4. Al finalizar con el ejecución de las funciones, el programa Monitor culmina con el commit para grabar todas las operaciones realizadas sobre los archivos. Si hubiera un

error en cualquier parte del esquema, el programa Monitor devuelve el estado de los archivos al estado anterior a la ejecución de la transacción.




## CARACTERÍSTICAS FUNCIONALES

### Descentralizar el servicio bancario

SARA BANK posee dos esquemas de trabajo a nivel de comunicación con el computador central del banco: autónomo y en línea.

El esquema de trabajo autónomo permite a la agencia atender las operaciones bancarias sin una conexión con el computador central del banco en tiempo real de ejecución de operaciones. Al final del día o semana, bajo este esquema, se produce un diskette para su envío a la oficina principal. Este esquema se utiliza mucho en agencias con dificultades de comunicación.

El esquema en línea permite el tratamiento de las operaciones en tiempo real con el computador central.- Dentro de este esquema se puede trabajar bajo tres modalidades:

On-line: Es decir nos encontramos comunicados con el host, de modo que una actualización en los archivos locales, afectará inmediatamente a los archivos del host.

Off-line: Cuando se produce una falla en la comunicación (pérdida de línea), el SARA BANK pasa automáticamente a este modo y almacena las transacciones para su posterior envío al host. Al activarse este modo es factible inhabilitar algunas transacciones que por sus características propias o por seguridad sólo puedan ser ejecutadas en modo on-line.

Off-load: Permite realizar una desconexión voluntaria de comunicación. Esta puede efectuarse a tres niveles: toda la agencia, un cajero o para una sola transacción. Esto se utiliza generalmente para agilizar la atención al cliente en horas de mayor congestión de comunicación o de gran afluencia de público.

**Permite la generación y el mantenimiento dinámico de transacciones**

La manera de definir la estructura de las transacciones, basada en la identificación de operaciones elementales y claramente definidas, permite la generación de nuevas transacciones por personal ajeno a las labores de programación. Lo único que se requiere es identificar dichas operaciones elementales y asociarlas a una función de la biblioteca de funciones de SARA BANK.

**Manejo multimoneda**

SARA BANK permite trabajar con diferentes monedas, lo que permite una mayor flexibilidad en la construcción de transacciones. Podemos construir transacciones multimoneda, es decir, donde la misma transacción pueda trabajar para diferentes monedas, o transacciones de moneda mixta, donde dentro de la transacción se trabaja con cantidades en diferentes monedas (por ejemplo, operaciones de compra-venta de moneda extranjera).

**Diseño funcional flexible de transacciones**

Las transacciones pueden ser construídas de la manera que la institución financiera esté acostumbrada a trabajar. Podemos concebir las transacciones de manera clásica, es decir, una transacción por cada tipo de producto o servicio (por ejemplo, depósito efectivo en cta cte., depósito efectivo en cta. ahorros, retiro efectivo en cta. ahorros, etc). Por el contrario, si deseamos reducir la cantidad de transacciones podríamos construirlas de manera cruzada (por ejemplo, depósito en efectivo, retiro en efectivo, depósito con cheque, etc).

**Flexibilidad para el control de acceso**

Para el control de accesos se manejan tres conceptos: grupo, perfil y usuario (nos referimos a la persona que maneja el sistema, vale decir, cajero, supervisor, etc.). Un grupo de transacciones permite clasificar a las transacciones según diferentes criterios: funcionales (transacciones de cuentas corrientes), operativos (transacciones de depósito), de restricción de acceso (transacciones del cajero de ME). El perfil es un concepto asociado al puesto de trabajo o función de una persona dentro de la agencia, por ejemplo: administrador, cajero, cajero de bóveda, etc. Un perfil está compuesto por grupos de transacciones. El usuario está asociado con una persona física y está definida por los perfiles que le han sido asignados. Ver figura A.7.

**Control de montos límites**

SARA BANK permite el control de límites a nivel de transacción, usuario y perfil de manera paramétrica e incluso de acuerdo a la condición de las comunicaciones. Por ejemplo, podemos definir que cuando la transacción trabaja en modo on-line, el monto límite a considerar sea a nivel perfil, y en el caso de modo off-line el monto sea validado respecto a límite a nivel transacción.

**Procesamiento de transacciones en dos puestos**

Las transacciones SARA BANK pueden ser construídas de manera que sean tratadas en dos puestos de trabajo. Por ejemplo, algunos bancos cuentas con un cajero que se encarga de

recibir el documento, ingresar los datos, liquidarlo si es necesario (por ejemplo, una letra) y a continuación lo pasa a otro empleado que se encarga exclusivamente de pagar o recibir el dinero por dicha operación. Otro caso que se produce con frecuencia es el sobregiro de cuentas que requiere de una autorización del supervisor (segundo puesto de trabajo). La persona que finaliza la transacción, la recupera cuando el concepto SARA BANK denominado recaptura de la transacción, esta modalidad de trabajo evita el reingreso de todos los datos. Ver figura A.8.

**Registro contable en Línea**

En algunos casos es muy importante realizar la contabilización de las operaciones en agencia en tiempo real, y ésta es una facilidad que brinda SARA BANK.

**Manejo flexible de la estructura de cuenta**

Cada banco posee una estructura de cuenta diferente, la definición de dicha estructura de cuenta es bastante flexible en SARA BANK. La única restricción consiste en que el tamaño no puede exceder 18 posiciones numéricas y que la cuenta no contenga caracteres alfanuméricos.

**Gestión de Cajas**

SARA BANK permite la gestión de las cajas a tres niveles: agencia, bóveda de la agencia y cajero. Adicionalmente permite el control de cada caja a nivel de denominaciones de billetes y monedas. La definición de las diferentes para cada moneda es dinámica y realizada por el usuario final. Debido a que no todos los bancos desean trabajar con el billetaje discriminado existen tres modalidades de manejo de billetaje: discriminado, global y mixto. El primero significa que cada vez que se haga un ingreso o salida de caja, el usuario deberá especificar la cantidad de cada denominación que recibe o entrega. En el caso de billetaje global, no existe solicitud adicional luego de ingresar la cantidad. En el manejo mixto de billetaje, el cajero maneja la caja de manera discriminada y la bóveda de la agencia de manera global.

**Guía de Procedimientos en línea**

SARA BANK provee la facilidad de contar con la guía operativa electrónica en cada puesto de trabajo. Esta guía es ingresada en SARA BANK.

## CARACTERISTICAS TECNICAS

El sistema está desarrollado íntegramente en MicroFocus COBOL. El software de base que permite el trabajo cliente-servidor para el manejo de archivos, servicios de impresión compartida, servicios de dispositivos bancarios compartidos, servicios de comunicaciones SNA (LUO, LU1, LU2, APPC y HLLAPI) se denomina LAN Distribution Paltform (LANDP). Se encuentra en diferentes sistemas operativos( DOS; OS/2, AIX, OS/400).

## COMPONENTES DEL SISTEMA

El propósito de la aplicación consiste en ejecutar transacciones bancarias, y al mismo tiempo proporcionar un ambiente de definición y construcción de dichas transacciones SARA BANK. Basado en la arquitectura actual de ejecución de las transacciones, el producto se compone de los siguientes productos: SARA BRANCH, SARA APPLIMATIC y SARA SIGN. Adicionalmente, se proveen los servicios de desarrollo de la interfaz con el computador central denominado SARA LINK, el servicio de garantía (SARA SUPPORT) y de mantenimiento (SARA MAINT).

**Sara Branch**

Corresponde al ambiente transaccional, es el entorno donde se ejecutan las transacciones. Es la parte del sistema que estará instalada en los puestos de trabajo de los cajeros de la agencia bancaria. Sus componentes más importantes son el programa MONITOR, la biblioteca de transacciones y la biblioteca de funciones.

**Sara Applimatic**

Es el ambiente donde se construyen las transacciones. Está formado por un generador de transacciones, un diccionario de datos y un generador de refrendos. Esta es la parte del

producto que permite la definición de los esquemas de transacciones, que no son otra cosa que la secuencia lógica de ejecución de funciones elementales para completar una transacción. Este componente del producto se instala en la oficina principal del banco en el área de organización y métodos u operaciones para el desarrollo y mantenimiento de los esquemas de transacciones y su posterior distribución a las agencias.

**Sara Sign**

Es el ambiente de captura y administración de firmas y restricciones de las cuentas. Permite la captura, creación, modificación, eliminación y distribución de firmas y restricciones (poderes, límites autorizados, etc.) de las cuentas. Trabaja en comunicación con el host o autónomamente. Permite almacenar la información desde un modo totalmente descentralizado (todas las firmas en el computador central) hasta descentralizado (las firmas sólo se encuentran en la agencia dueña de la cuenta).

## ANEXO 2 : LA PLATAFORMA DE NEGOCIOS

## INTRODUCCION

Uno de los negocios del sistema financiero que tiene mejores perspectivas de crecimiento y en el que las entidades del sector vienen concentrando sus esfuerzos, es el de la banca personal.

La plataforma de clientes es el frente de atención a través del cual se canalizan estos esfuerzos apoyados en el uso de información actualizada de clientes, prospectos y productos, y de herramientas que facilitan a los ejecutivos comerciales y de cuentas convertirse en los propulsores de estas oportunidades de negocio.

En este frente encontramos diversos componentes bajo los cuales se agrupan una serie de funciones específicas, mas o menos complejas dependiendo de la información con la que cuenta la institución financiera. Los componentes que consideramos son:

- Gestión de Productos
- Gestión de Clientes
- Gestión de Ventas
- Gestión de Cuentas
- Gestión Transaccional

## GESTIÓN DE VENTAS

Incluye un conjunto de funciones que permiten al ejecutivo comercial y de negocios realizar una efectiva labor de venta de los productos y servicios de la institución.

Entre las características más importantes que podemos mencionar tenemos:

- Permite a los funcionarios encargados de las Ventas de Productos y Servicios Financieros gestionar los mismos desde el manejo de prospectos hasta el seguimiento postventa.
- Proporciona información relevante y actualizada que ayude al progreso de toma de decisiones en cuanto a ventas se refiere.
- La Gestión de Ventas se soporte en sistemas que proporcionan la mejor interfaz usuario tomando en cuenta que debe usarse en forma interactiva con el cliente.

Las funciones de la Gestión de Ventas son:

- Venta Cruzada
- Atender Solicitudes, Aperturas y Afiliaciones
- Consulta de Información Institucional
- Promoción de Productos
- Simulación de Productos
- Captura de Requerimientos
- Seguimiento de Prospectos
- Calificación de Clientes
- Captura y Consulta de Avales
- Información de Tarifas, Intereses y Comisiones
- Impresión de Contratos y Formularios

## ¿QUÉ ES LA VENTA CRUZADA?

Definiremos la Gestión de Venta Cruzada como el proceso de venta basado en las características y calificación del cliente o prospecto, y las características y requerimientos de los productos de la institución financiera. Este proceso es automatizado para facilitar la labor de venta de productos y servicios de los ejecutivos comerciales a los clientes o prospectos.

Entre las características y requerimientos de los productos a considerar en una sesión de Venta Cruzada tenemos:

**Características**

- Aplicativos
- Tipos de operación
- Tasas, comisiones
- Tipo de productos
- Productos Complementarios
- Productos Suplementarios
- Ciclo de Vida del Producto

**Requerimientos**

- Productos Pre-requisitos
- Monto Mínimo
- Monto Máximo
- Ingreso mínimo promedio
- Mercado Objetivo
- Nivel de Riesgo
- Datos Complementarios

Dentro de la información Cliente a considerar tenemos:

Potencial del Cliente

- Categoría
- Calificación
- Rentabilidad
- Saldo Promedio
- Nivel de Riesgo

Información Histórica

- Datos personales
- Datos financieros
- Datos Comerciales
- Movimientos
- Productos que posee
- Productos en otros bancos

Las características principales de la venta cruzada automatizada son las siguientes:

- La interfaz con el usuario es gráfica, la cual incluye el manejo de íconos y menúes.
- Muestra la lista de Productos y Servicios que ofrece el banco con la finalidad de seleccionar uno o más productos, y poder discutir con el cliente sobre las ventajas y características de cada uno de ellos.
- Muestra la lista de Productos y Servicios que son *comparables* con aquellos que son seleccionados, ofreciéndole alternativas al cliente o prospecto con respecto al producto seleccionado.
- Muestra la lista de Productos y Servicios relacionados con aquellos que son escogidos.
- El sistema selecciona los productos y servicios que son aplicables al perfil del cliente o prospecto que se está atendiendo, para esto el sistema:
  * Accede al perfil del cliente (parámetros) que le permita seleccionar los productos y servicios que aplican a dicho perfil (Ejemplo: saldo promedio, actividad económica, productos y servicios ya adquiridos, etc.). Considerando los niveles de riesgo correspondientes.
  * Adicionalmente, y si es necesario, el sistema solicita el ingreso de información adicional que le permita realizar la selección de las alternativas. Esta información adicional puede ser general y/o relacionada a los productos que se vayan seleccionando.

* Para el caso de prospectos, es decir, para aquellas personas o empresas de las cuales el banco no tiene perfil, el sistema debe solicitar el ingreso de información básica que le permita hacer la selección correspondiente.

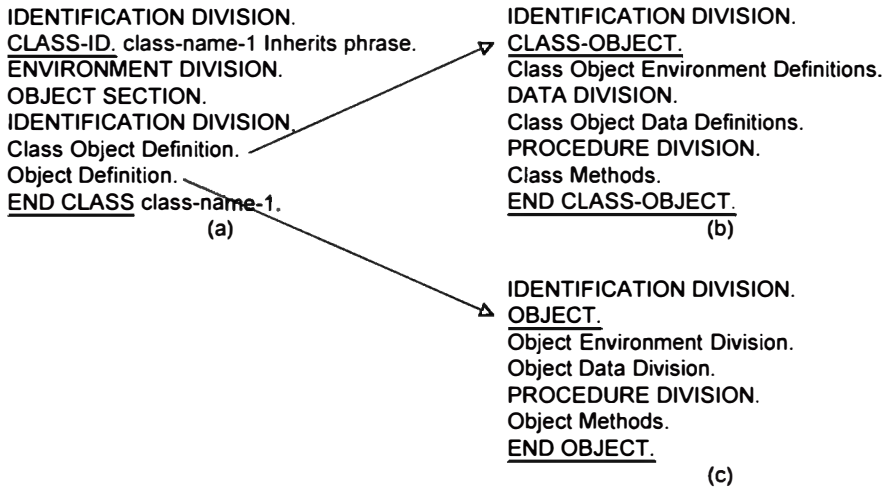
- Comparación de productos o servicios comparables. El sistema presenta adicionalmente a la información general, aquella información comparable de las variables de ambos Productos de una forma resumida.
- Muestra información de promoción (gráficos) de los productos seleccionados, la cual es invocada en cualquier momento durante la sesión de atención al público.
- Muestra información resumida de los productos seleccionados, la cual es invocada en cualquier momento durante la sesión de atención al público.
- Cada producto o servicio tiene asociado funciones de "What-if".
- Permite ir seleccionando aquellos productos y/o servicios que el cliente va adquiriendo con el objeto de ejecutar las aperturas correspondientes al final de la sesión.
- Dependiendo de los productos vendidos, el sistema solicita el ingreso de información complementaria, caso contrario si esta información ya ha sido ingresada con anterioridad, el sistema recupera esta información y la muestra en pantalla. O que al final de la sesión dichos datos complementarios sean ingresados.
- Imprime los formatos correspondientes a las aperturas.
- El sistema debe permitir a mas de un cliente/prospecto a la vez.

## ANEXO 3 : ANSI OBJECT COBOL

A continuación se presentan las características de la técnica de análisis y diseño orientado a objetos que son soportadas por este nuevo estándar ANSI del COBOL.

## DEFINICIÓN DE CLASE

La definición de clases en el Object Cobol está centrada alrededor del parágrafo Class-Id, y proporciona la estructura para la definición de variables de clase, variables de instancia, métodos para la clase e instancia, y la posición de un objeto dentro de la estructura jerárquica. La clase está definida dentro del Object Cobol como "la entidad abstracta a la cual pertenece un objeto clase y cero o más objetos que comparten la misma definición". Opcionalmente incluída dentro de la definición de una clase están las definiciones de la *class environment, class object y object*. A pesar que el Object Cobol no soporta explícitamente metaclases, el objeto clase provee algunas funcionalidades en este sentido. Desde el punto de vista de la sintaxis, una clase está definida por un nombre de clase y una serie de *identificatio divisions* intercaladas por definiciones de objetos clases e instancias de objetos, como se muestra a continuación en la Figura 1(a).




## DEFINICIÓN DEL OBJETO CLASE

La clase objeto debe capsular datos y procedimientos, incluyendo su propia *data* y *procedure division*. Los datos declarados dentro de una clase son globales a todas las instancias de esta

clase. La responsabilidad principal de un objeto clase es proporcionar los constructores y destructores para la creación y supresión de instancias de objetos. Sin embargo, los constructores y destructores no son obligatorios, pues el Cobol provee por defecto dichas operaciones. La estructura de una definición de objeto clase se puede apreciar en la figura 1(b).

## DEFINICIÓN DE UN OBJETO

Cada objeto pertenece a una clase, los datos y procedimientos que comprenden un objeto son declarados dentro del parágrafo Object, y puede incluir una evironment division, una data division, y una procedure division del objeto. Los métodos del objeto están contenidos dentro de la procedure division del objeto mientras que los datos del objeto están declarados dentro de la file o working storage de la data division del objeto. La figura 1(c) muestra un esquema de la definición de un objeto.

## CREACIÓN DE UN OBJETO

La definición de un objeto solo provee uno de caracter potencial, no se genera automáticamente la instancia, es necesario crearlo mediante un método. El mecanismo que permite la creación de objetos es provista por una clase base denominada CBL-Base, heredada por todos los objetos clase. La clase CBL-Base no tiene datos, solo métodos, uno de los cuales, CBL-New, es usado por los objetos clase para crear instancias. A continuación se ilustra la creación de una instancia. La palabra clave USAGE ha sido extendida para incluir datos del tipo objeto. SELF y SUPER son indentificadores de objetos predefinidos usados para referenciar el objeto de un método en ejecución. La forma general del INVOKE solicita el servicio de un objeto, en este caso de CBL-New, y retorna el handle del objeto a object.item-1. La data division o working storage section de un método puede contener datos temporales, creados cada vez que el método es invocado. La invocación del método crea una copia de la data, asignada únicamente al método.

```
IDENTIFICATION DIVISION.
CLASS-OBJECT.
PROCEDURE DIVISION.
METHOD-ID. method-name-1.
```

```
LINKAGE SECTION.
01 data-name-1 USAGE OBJECT REFERENCE SELF.
PROCEDURE DIVISION RETURNING object-item-1
    INVOKE SUPER 'CBL.NEW' RETURNING object-item-1
    EXIT-METHOD.
END CLASS-OBJECT.
```

## CICLO DE VIDA DE UN OBJETO

Los atributos de los objetos clase y de los objetos están definidos en el parágrafo Class-Id como persistent o transient, y si son definidos como transient, pueden ser collectable o non-collectable. Los objetos persistentes son almacenamientos de datos de larga duración y existen hasta que no se especifique explícitamente su destrucción. La destrucción de los objetos es realizada al invocar el método CBL-Discard de la clase CBL-Base. Los objetos transient existen por el tiempo que dure una unidad de ejecución y son destruídos automáticamente cuando la visibilidad del objeto termina o cuando la unidad de ejecución culmina. El listado 2 especifica la sintaxis para especificaciones de atributos de clase.

```
IDENTIFICATION DIVISION.
CLASS-ID. class-name-1 IS { TRANSIENT [COLLECTABLE] | PERSISTENT }.
Class Environment Division.
IDENTIFICATION DIVISION.
CLASS-OBJECT.
```

## LOS MÉTODOS

Los métodos son declarados dentro del Object-Cobol mediante la introducción de una nueva estructura de programa, el parágrafo Method-Id. Los múltiples pueden ser definidos dentro de la procedure division de los objetos clase u objetos que los contienen. La asignación de un atributo de método como público, privado o restringido define la interfaz y el grado de visibilidad de un objeto. Una interfaz de objeto está definida por su colección de métodos y

especificación de parámetros. De igual modo que con el Cobol estándar, cada parámetro debe ser definido de acuerdo con la definición establecida dentro de la linkahe section. Los items de datos definidos en la linkage section y recibidos en la sentencia using, deben ser de un tipo de dato Cobol válido, incluyendo los objetos. La palabra clave returning permite responder a los clientes, con más de un parámetro de cualquier tipo. Una manera general para definir prototipos de un método de clase, un método de objeto, y un método son mostrados en el listado 3, que a continuación se presenta.

```
IDENTIFICATION DIVISION.
METHOD-ID. method-name-1
IS { PUBLIC
   { RESTRICTED } PROTOTYPE OF CLASS-OBJECT OF class-name-1.
   { PRIVATE
Method Environment Division.
DATA DIVISION.
Method Data Definitions.
Linkage Section.
PROCEDURE DIVISION USING phrase RETURNING phrase.
Method Procedure Statements.
END METHOD method-name-1.
```

El Object Cobol soporta el concepto de class utility, donde métodos de común interés entre clases son compartidos. Los Class Utilities son métodos de tipo prototype, agrupados bajo una declaración de interfaz explícita. Unicamente los nombres de métodos y parámetros son incluídos en la interfaz, permitiendo que la funcionalidad sea compartida por todas las clases. Un ejemplo de prototipo de método es ilustrado a continuación.

```
IDENTIFICATION DIVISION.
METHOD-ID. method-name-1 IS PROTOTYPE.
DATA DIVISION.
LINKAGE SECTION.
PROCEDURE DIVISION USING phrase RETURNING phrase.
END METHOD method-name-1.
```

## INVOCACIÓN DE MÉTODOS

Para invocar o llamar a un método se utiliza la palabra clave INVOKE seguida del identificador de objeto y un nombre de método. INVOKE pasa un mensaje o solicita un servicio para un cliente a un objeto proveedor. El listado 5 muestra la sentencia INVOKE.

```
IDENTIFICATION DIVISION.
METHOD-ID. method-name-1.
PROCEDURE DIVISION.
    INVOKE object-identifier-1 method-name-1 USING phrase
            RETURNING phrase
    END-INVOKE
EXIT METHOD.
```

## ENCAPSULAMIENTO

El Object Cobol controla el acceso de datos y métodos mediante tres atributos claves: público, privado y restringido. Privado y restringido son mutuamente excluyentes, y limitan o relajan el acceso a la data encapsulada. Por defecto, la data de los objetos clase y objetos es privada, accesible sólo a los métodos definidos dentro del objeto. La declaración como restringidas para entradas de una file description permite el acceso a los data items mediante los métodos de clases descendientes. Los métodos pueden ser declarados públicos, privados o restringidos. Una definición abreviada de data privada y restringida de objetos clase es ofrecida a continuación.

```
IDENTIFICATION DIVISION.
CLASS-OBJECT.
Class Object Environment Definitions.
```

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01  record-1 IS RESTRICTED.
01  record-2 IS PRIVATE.
...
END CLASS-OBJECT.
```

## SELECCIONANDO OBJETOS

La facilidad para consultar sobre un objeto es soportado por el Object Cobol mediante la sentencia SELECT. Esta sentencia es utilizada junto con la invocación de un método del objeto de interés. Una invocación de SELECT se presenta a continuación en el listado 7.

```
IDENTIFICATION DIVISION.
METHOD-ID. method-name-1.
PROCEDURE DIVISION.
    SELECT class-name-1 into object-identifier-1
        WHEN object-identifier-1  :: method-name-1  condition-1
            imperative-staments
        WHEN OTHER
    END-SELECT
```

## HERENCIA

El Object Cobol permite la herencia simple y múltiple y la noción de unión de clases. La sintaxis del uso de herencia de clases se muestra en el listado 8.

```
IDENTIFICATION DIVISION.
CLASS-ID. class-name-1 Class attributes INHERITS class-name-2 ...
...
END CLASS class-name-1.
```

Las interfaces al igual que las clases pueden ser heredadas dentro del Object Cobol. La herencia de interfaces extiende el concepto de jerarquía de clases, de manera separada de la definición e implementación de clase. La sintaxis de la herencia de interfaz se muestra en el listado 9.

```
IDENTIFICATION DIVISION.
INTERFACE-ID. interface-name-1INHERITS interface-name-2 ...
...
END INTERFACE interface-name-1.
```

## POLIMORFISMO

El Polimorfismo es implementado mediante la extensión de las palabras claves USAGE y SET, y por la introducción de dos nuevas palabras: AS y UNIVERSAL. Un ejemplo de la frase extendida USAGE se muestra en el listado 10 para la declaración de items de datos del tipo objeto.

```
01 object-item-1 USAGE OBJECT REFERENCE class-name-1 ONLY.
01 object-item-2 USAGE OBJECT REFERENCE class-name-1.
```

En el listado 10, object-item-1 solo pueden referenciar a los objetos de la clase class-name-1. mientras que object-item-2 puede referenciar a cualquier objeto de la clase class-name-1 o cualquier descendiente de la clase class-name-1. Adicionalmente, el Object Cobol permite a un objeto pertenecer a una clase desconocida como se muestra en el listado 11.

```
01 object-item-1 USAGE OBJECT REFERENCE.
```

El objeto referido como object-item en el listado 11 es desconocido en tiempo de compilación y puede variar dependiendo del proceso de ejecución.

```
SET object-item-1 to object-item-2 AS UNIVERSAL.
```