UNIVERSIDAD AUTONOMA DE QUERETARO

ESCUELA DE INFORMATICA

Curso de Especialización :

" INGENIERIA DE SOFTWARE CON LA METODOLOGIA DE OBJETOS "

Tesina para obtener el titulo de :

LICENCIADO EN INFORMATICA

Presenta :

Ontiveros González Enrique

Mayo 1995

A mis padres

Por su gran apoyo, compresión y cariño atravez de tantos años de actividad estudiantil; hasta llegar a este tan importante momento.

Mi más grande amor y cariño.

A mis hermanos

J. Luis, Raul, Amparo y Ruth.

Mi admiración y respeto.

A mi Universidad

Por proporcionarme la oportunidad de adquirir una excelente educación integral.

Mi reconocimiento.

Al I.S.C. Armando González Basaldua

Por poner un gran empeño en iniciarnos en nuevas tecnologías que nos permiten ser mejores en nuestro actuar profesional.

Mil gracias.

A mis profesores

Por contribuir con sus valiosos conocimientos a mi formación personal y profesional.

Mis más sincero agradecimiento.

A mis amigos y compañeros

Por compartir buenos y malos momentos, por ser amigos, por ser compañeros.

Gracias.

ÍNDICE

Pág.

ÍNDICE

Pág.

ÍNDICE

INTRODUCCIÓN

El presente trabajo esta orientado para dar a conocer las técnicas de desarrollo y mantenimiento existentes en el campo de la producción de software orientadas a sistemas de computo digitales, haciendo énfasis en la metodología orientada a objetos (OOP). Esto debido a la imperiosa necesidad que se tiene (al menos por lo que he detectado en mi persona, en compañeros de la misma Universidad y en el campo laboral donde se convive con egresados de otras instituciones educativas), de usar una metodología específica para cada producto de software que se desee realizar. Considero que es necesario aclarar, antes de continuar, que el término "Producto de Software" es diferente al término "Software de uso Personal"; ya que el primero requiere de un enfoque más sistemático que el segundo, así como también tiene una mayor cantidad de usuarios y de programadores.

Siempre que se va a iniciar un desarrollo de software es necesario e imprescindible determinar y establecer explícitamente las necesidades, limitaciones y requerimientos específicos del usuario, ya que de esta manera tendremos una base sólida que sustentará nuestro desarrollo, en otras palabras, necesitamos saber lo que tenemos, lo que queremos y hacia donde vamos, para poder tomar medidas de acción y lograr nuestros objetivos de una manera fácil y contundente, y que además nos reditúe los mayores beneficios. En el desarrollo de productos de programación o de software se debe tomar en cuenta tanto al usuario como a los desarrolladores, así como a las personas que darán mantenimiento al producto y los equipos de cómputo. Además del código fuente, se deben elaborar todos los documentos de apoyo, tales como manual de operación, de instalación, del usuario, de entrenamiento y de documentación.

Lo anterior, con la finalidad de mejorar la calidad de los productos de software, aumentar la productividad, y la satisfacción personal.

CAPITULO 1
INTRODUCCIÓN A LA INGENIERÍA DE SOFTWARE

CONCEPTOS GENERALES

A lo largo del documento intentaremos explicar lo que es la Ingeniería de Software, por lo que iniciaremos definiéndola como: *una disciplina tecnológica y administrativa dedicada a la producción sistemática de productos de programación, que son desarrollados y modificados a tiempo y dentro de un presupuesto definido.*

La ingeniería de software, como disciplina, busca mejorar la calidad de los productos de software y aumentar la productividad y satisfacción profesional de las personas relacionadas con esta área.

Esta nueva disciplina utiliza técnicas de ingeniería para especificar, diseñar, instrumentar, validar y mantener los productos dentro del tiempo y el presupuesto establecido para el proyecto, además de contemplar aspectos administrativos. En contraste, la metodología tradicional (la que en su gran mayoría hacemos como estudiantes en las carreras de informática o computación, o la que llevan a cabo las personas ajenas al área para satisfacer sus necesidades) no requiere de gran nivel de especificación y detalle; ya que por el contrario, el desarrollo se realiza intuitivamente, esperando que la inspiración ilumine el camino correcto que debemos de seguir, para solo obtener el producto sin contemplar su eficiencia y muchas veces ni su efectividad.

Por lo anterior pudiéramos decir que los conceptos de ingeniería de software son aplicables a proyectos grandes y de larga duración, pero esto es completamente una falacia, pues aunque en los proyectos grandes es más notoria la aplicación de todos los pasos, etapas y técnicas de la ingeniería de software, éstos también son aplicables a los desarrollos pequeños, aunque algunas aplicaciones no sean factibles.

La ingeniería de software se preocupa por el desarrollo sistemático y el mantenimiento de la documentación y los manuales de apoyo, así como por el código mismo; es decir, cuida la documentación interna que describe las características del código fuente y la documentación externa que describe las características de los documentos asociados con el código.

Se puede considerar a la **CALIDAD** del producto como la preocupación primordial de la ingeniería de software, aunque las características de calidad especificas dependerán de cada producto en particular, ya que en ocasiones la portabilidad puede ser lo importante, y en otras el uso eficiente de la memoria. Pero existen características fundamentales de calidad para cada producto de software: la utilidad, la claridad, la confiabilidad, la eficiencia y la economía.

El factor más importante de la calidad es la Utilidad, es decir, que el producto satisfaga plenamente las necesidades del usuario.

La confiabilidad de un producto de software se refiere a su capacidad de desempeñar una tarea bajo ciertas condiciones durante un tiempo específico.

Los productos de software deben estar escritos con claridad para ser fáciles de entender y para probarlos de una manera exhaustiva que permita obtener la confiabilidad sobre él mismo.

Un producto de software debe ser eficiente, es decir, debe hacer lo que tenga que hacer; pero, lo debe hacer de la mejor manera posible.

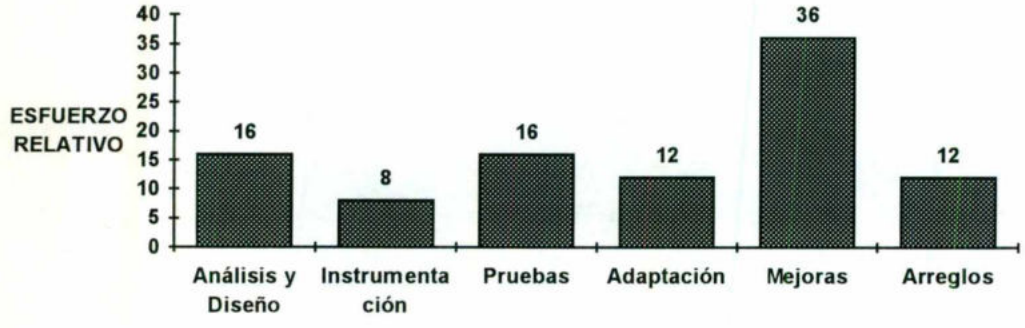
Por último, diremos que para que un producto de software sea económico, debe desempeñar una tarea específica usando menos tiempo o menos recursos humanos o industriales que los que se requerían antes de tenerlo.

FACTORES A CONSIDERAR

Para desarrollar un producto de software, es necesario considerar varios factores que nos permitan hacer una adecuada planeación de dicho desarrollo, lo cuál nos lleva a obtener los mejores resultados posibles sin invertir esfuerzo adicional al requerido.

a) Factores de tamaño

Dentro de este tipo de factores, debemos considerar el software o las herramientas, así como las técnicas que pueden mejorar la productividad de los programadores. Contemplar la distribución del esfuerzo durante el periodo de vida de un producto, el cual tiene una media de uno a tres años de desarrollo y de cinco a quince en su uso.



También hay que clasificar los proyectos de acuerdo a su tamaño, para poder determinar los niveles de control administrativo y el tipo de herramientas y técnicas que son necesarias para el proyecto. Podemos clasificar a los proyectos de acuerdo a su tamaño como:

- **Proyectos triviales:** no requieren del desarrollo de un análisis formal, documentación de diseño muy elaborada, pruebas pilotos extensivas, ni documentación de apoyo clara; sin embargo, el aplicarlas (aunque sea en grado mínimo) puede mejorar mucho el producto final.

- **Proyectos pequeños:** por lo regular no tienen interacción con otros programas, requieren de poca interacción entre programadores, e incluso entre programadores y clientes. Los estándares técnicos de documentación y notaciones, así como las revisiones sistemáticas de los proyectos deben usarse, aunque el grado de formalidad será menor al empleado en proyectos grandes.

- **Proyectos medianos:** se exige la interacción entre programadores y la comunicación con los usuarios; de ahí que se necesite cierta formalidad en la planeación, documentación y revisión del proyecto. Podemos considerar entre estos proyectos a los ensambladores, compiladores, sistemas pequeños de manejo de información, sistemas de inventario y aplicaciones de control de proceso.

- **Proyectos grandes:** comúnmente tienen interacción significativa con otros programas y sistemas de programación. Por su tamaño y complejidad es difícil, si no imposible, la prevención de eventualidades durante la fase de planeación y análisis. Son ejemplos típicos de este tipo de proyectos, los compiladores de gran tamaño, sistemas pequeños de tiempo compartido, paquetes de base de datos, sistemas gráficos para la adquisición y despliegue de información y proyectos para control de tiempo real. Por su magnitud, son esenciales los procedimientos sistematizados, la documentación estándar y las revisiones formales.

- **Proyectos muy grandes:** consta de varios subsistemas, cada uno de los cuales es un proyecto grande. Los subsistemas suelen tener interacciones complejas entre ellos y con otros sistemas desarrollados aparte. Comprenden sistemas de procesamiento en tiempo real, sistemas de telecomunicaciones y multiusuarios; entre ellos se pueden mencionar grandes sistemas operativos, grandes bases de datos, sistemas de control y dirección militar.

- **Proyectos extremadamente grandes:** constan de varios subsistemas de gran tamaño, que comprenden conceptos de tiempo real, telecomunicaciones, multitareas, y procesamiento distribuido; tienen a menudo requisitos de confiabilidad muy altos e incurren en procesos de vida o muerte. Por ejemplo, los controladores de tráfico aéreo, sistemas de proyectiles de defensa y sistemas de control y comando militar.

CATEGORIAS EN EL TAMAÑO DE UN PRODUCTO

Categorías	Número de Programadores	Duración	Tamaño del Producto
Trivial	1	1-4 Sem.	500 Línea de Código
Pequeño	1	1-6 Sem.	1k-2k
Mediano	2-5	1-2 Años	5k-50k
Grande	5-20	2-3 Años	50k-10
Muy grande	100-1000	4-5 Años	1M
Extremadamente grande	2000-5000	5-10 Años	1M-10

Es importante dentro de los factores de tamaño saber estimar como es que los programadores usan su tiempo en un desarrollo de programación, ¿porque?, simple y sencillamente porque nos ayudarán a realizar una planeación más eficiente de nuestro proyecto, y además, por que nos va a permitir cotizar de una manera más real nuestro proyecto en términos monetarios.

Para ejemplificar lo anterior, mencionaremos un estudio realizado por BELL Laboratories en 1964 (aunque hace ya tiempo, no existe razón por la cual se podían esperar cambios determinantes en la distribución del tiempo); los resultados de dicho estudio son los siguientes:

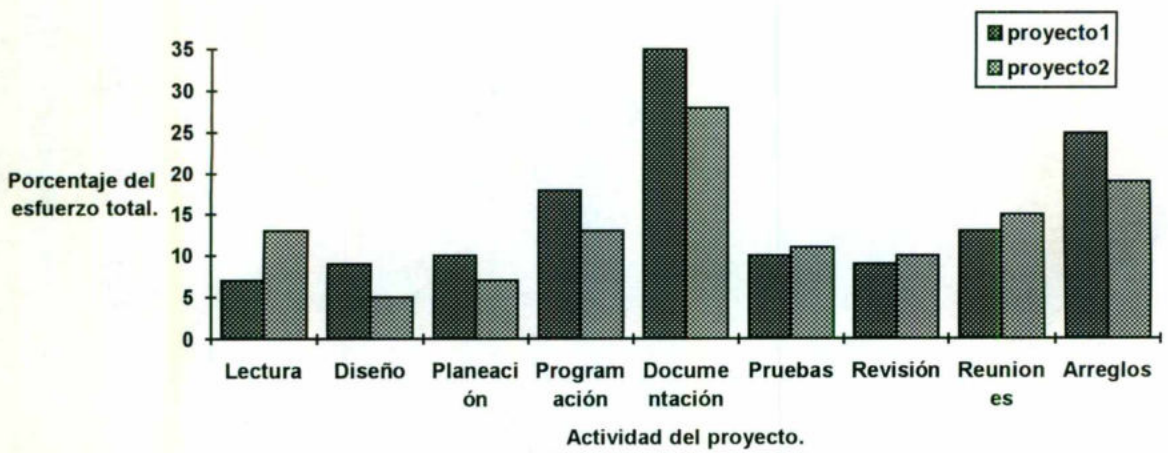
Estudio de los Laboratorios Bell (1964, 70 programadores)

Escritura de programas	13%
Lectura de manuales y programas	16%
Comunicaciones de trabajo	32%
Usos personales	13%
Varios	15%
Entrenamiento	6%
Correo	5%

39 % " Otros"

Como se aprecia en la gráfica, la escritura de programas ocupa un 13% del tiempo total del desarrollo, pero, lo que se intenta mostrar aquí son los problemas de la estimación del costo de un proyecto; puesto que muy pocas veces se consideran los demás porcentajes mostrados, lo que eleva considerablemente el costo total del desarrollo.

En la siguiente gráfica vemos como el proyecto 2 utiliza mayor tiempo en la actividad de lectura, reduciéndolo en actividades criticas como lo son diseño, planeación, programación y documentación.



b) Factores de Calidad y Productividad

El desarrollo y mantenimiento de programas son tareas muy complejas; a quien han escrito sólo pequeños programas para uso personal o como tareas escolares puede dificultársele entender la importancia de un desarrollo sistemático. La calidad del producto y la productividad del programador puede elevarse al mejorar los procesos necesarios para el desarrollo y mantenimiento de los productos. Algunos factores que influyen sobre la calidad y productividad se muestran a continuación:

- **Capacidad individual**, en este punto se deben considerar tanto la familiaridad del individuo con el área particular de aplicación, como su competencia global. Es decir, un buen programador en procesamiento de datos puede no serlo tanto en aplicaciones científicas o viceversa; por lo tanto siempre hay que contratar gente de lo mejor, sin embargo, no siempre es posible contratar a personas excepcionales.

- **Comunicación en el grupo**, por tradición se ha considerado a la actividad de programar como algo individual y privado, lo que provoca en muchas ocasiones, que el programador tome un enfoque subjetivo de la tarea asignada; malentendiendo en la mayoría de las veces el objetivo que se busca. La ingeniería de software tiene como propósito lograr que los programas sean más sociables para mejorar la comunicación entre los programadores, mediante el uso de técnicas como la revisión de diseño, recorridos estructurados y los ejercicios de lectura de código.

- **Complejidad del producto**, podemos mencionar tres niveles o tipos de complejidad generalmente aceptados para los productos de software: programas de aplicación, programas de apoyo y programas de sistema operativo. Los primeros incluyen rutinas científicas y de procesamiento de datos escritas en un superlenguaje; los segundos comprenden a compiladores, ensambladores, ligadores y cargadores, éstos pueden estar escritos en un superlenguaje o en lenguaje ensamblador; por último, los sistemas operativos se relacionan con rutinas de comunicación de datos, procesamiento en tiempo real para sistemas de control y las rutinas básicas del sistema operativo, los cuales suelen escribirse en lenguaje ensamblador o en un lenguaje de desarrollo de sistemas.

Los programas de aplicación tienen el más alto nivel de productividad, mientras que los programas de sistema operativo tienen el menor, medido en número de líneas de código por día de programador. La velocidad con que se puede terminar un proyecto de aplicación respecto a los de sistemas operativos es entre 25 y 100 veces mayor, mientras que los de apoyo pueden producirse a una velocidad 5 o 10 veces mayor que los de sistemas operativos.

- **Notaciones apropiadas**, los esquemas de representación son de vital importancia; una buena notación puede aclarar las relaciones e interacciones de importancia, mientras que las notaciones deficientes complican e interfieren con la buena práctica de la disciplina. Las notaciones apropiadas son vehículos de comunicación entre el personal asignado al proyecto.

- **Enfoques sistemáticos**, es necesario utilizar procedimientos y técnicas aceptadas de desarrollo de programas; con la finalidad de convertir en una disciplina profesional el desarrollo de software.

- **Control de cambios;** normalmente cuando desarrollamos un programa, es común toparnos con situaciones no contempladas que nos llevan a adaptar nuestros programas a éstas. Estos cambios ocurren por diversas causas, por gustos del cliente, características de equipos, políticas externas, etc.; pero sin importar su origen, el efecto es el mismo: incremento en los costos. Es necesario que al planear un proyecto se tome en cuenta la creación de un plan para el control de cambios.

- **Nivel tecnológico**, nos referimos en este punto al análisis y elección del lenguaje de programación que se va a utilizar, el equipo y las técnicas de programación, así como las herramientas de programación disponibles.

- **Nivel de confiabilidad**, todo producto de software debe de ofrecer un cierto nivel de confiabilidad. Altos niveles de confiabilidad nos conducen a una reducción en la productividad, estableciéndose un nivel de productividad de 2:1 entre un nivel de baja confiabilidad y uno de alta confiabilidad.

- **Captación del problema,** es necesario entender perfectamente el problema a solucionar ya que si no lo hacemos, nuestra solución no servirá de nada. La planeación cuidadosa, las entrevistas con el cliente, la observación de la tarea manual, el desarrollo de prototipo, una versión preliminar del manual del usuario y la especificación precisa del sistema puede incrementar el entendimiento del cliente y del experto sobre el problema en cuestión.

- **Habilidades gerenciales**, es necesario hacer notar que si queremos que un desarrollo sea llevado a buen término, se le debe encargar a una persona con aptitudes y conocimientos administrativos/gerenciales así como de programación. Evitando usar gerentes con poco o nulo conocimiento en programación, así como a programadores sin aptitudes/conocimientos gerenciales.

c) Factores de Administración

Las actividades técnicas y las gerenciales son igualmente importantes para el éxito de un proyecto de programación. Los gerentes controlan los recursos y el ambiente en que las actividades técnicas se suceden; así los gerentes tienen la última responsabilidad de asegurar que los productos se entreguen a tiempo y dentro del presupuesto estimado, además que los productos exhiban la funcionalidad y calidad que el cliente requiere. Las actividades gerenciales incluyen la elaboración de un plan de trabajo, contratación de proyectos, desarrollo de estrategias de mercado, así como la contratación y entrenamiento de personal.

Las actividades de la administración de un proyecto comprenden los métodos para organizar y seguir el curso a un proyecto; estimación de costos, políticas de asignación de recursos, control del presupuesto, definición de logros del proyecto, determinación del avance del proyecto, reasignación de recursos y ajustes al calendario de trabajo, establecimiento de procedimientos de control de calidad, mantenimiento de las diversas versiones, promoción de la comunicación entre miembros del proyecto, comunicación con los clientes, desarrollo de acuerdos contractuales con los clientes, y también asegurarse de la observancia de los términos legales y contractuales del proyecto.

- **Tiempo disponible,** la determinación del nivel optimo de personal y el tiempo requerido para desarrollar las diferentes actividades en un proyecto de programación es un aspecto importante y difícil en la estimación global de costos o recursos. No es lo mismo usar un programador durante seis meses que usar seis programadores durante un mes, para un proyecto que requiere un esfuerzo de seis meses - programador. La primera opción es mas eficiente que la segunda, pero aún más eficiente puede ser el usar dos programadores durante tres meses; esto, debido a la curva de aprendizaje esta directamente ligada al numero de participantes y esto afecta notablemente al proyecto.

- **Especialización requerida**, las actividades implicadas en la ingeniería de software reclaman una diversidad de habilidades y características personales entre las que podemos mencionar la habilidad de comunicación, tacto y diplomacia para extraer información y determinar necesidades; la habilidad para resolver problemas, la habilidad de deducción, etc. Estas habilidades son necesarias para el buen desarrollo del proyecto, y aunque es imposible que una persona las posea todas, es necesario reunirlas aunque sea con diferentes personas; y asignarles una tarea especifica de acuerdo a sus habilidades.

- **Facilidades y recursos,** hacemos referencia en este punto a las facilidades que se proporcionan al programador para desempeñar su tarea, así como a las herramientas con que podrá contar. pero se hace énfasis en el aspecto motivacional del programador, es decir, el tener un buen equipo y un lugar apropiado es más motivante que tener un estacionamiento privado o baños particulares; así mismo, siempre requieren de variedad en su trabajo y oportunidad de crecer profesionalmente.

- **Entrenamiento adecuado,** para contratar a un programador es necesario conocer su formación y experiencia, ya que no solo la implementación conforma un desarrollo completo, si no que se tienen que dominar las etapas de análisis, diseño, pruebas, mantenimiento y técnicas de administración.

CAPITULO 2
PLANEACIÓN DE UN PROYECTO DE SOFTWARE

DEFINICIÓN DE PROBLEMA

Todo elemento desarrollado por el hombre primero es una idea en su mente. Los sistemas computacionales se desarrollan en respuesta a requerimientos detectados. Las fuentes que producen las ideas de productos de programación incluyen las necesidades del cliente generadas externamente, las necesidades internas de la organización, planes de mercadotecnia, y los planes o misiones organizacionales.

El primer paso en la planeación de un proyecto de software es la definición del problema. Para hacer una buena definición del problema es necesario :

1.- Desarrollar un enunciado definitivo del problema por resolver utilizando terminología del cliente. Incluyendo una descripción de la situación actual, restricciones del problema , y de las metas que se lograrán.

2.- Justificar una estrategia de solución computarizada.

3.- Identificar las funciones por realizar, las restricciones, el subsistema de equipo electrónico, el subsistema del producto, y el del personal.

4.- Determinar los objetivos y requisitos en el nivel del sistema para el proceso de desarrollo y los productos finales.

5.- Establecer criterios de alto nivel para la aceptación del sistema.

ESTRATEGIA DE SOLUCIÓN

El segundo paso a realizar es desarrollar una estrategia de solución. Esta no es un plan detallado de solución, sino un enunciado general sobre la naturaleza de las posibles soluciones. Los factores estratégicos incluyen procesamiento por lote o tiempo compartido; base de datos o sistema de archivos; gráficas o texto y procesamiento en tiempo real o en línea. Una estrategia de solución debe considerar todos los factores externos que son visibles para el usuario del producto, y debe redactarse de tal manera que permita caminos alternos para el diseño de producto. Los pasos a seguir en el establecimiento de la estrategia de solución son:

1.- Esbozar varias estrategias de solución, sin considerar las restricciones.

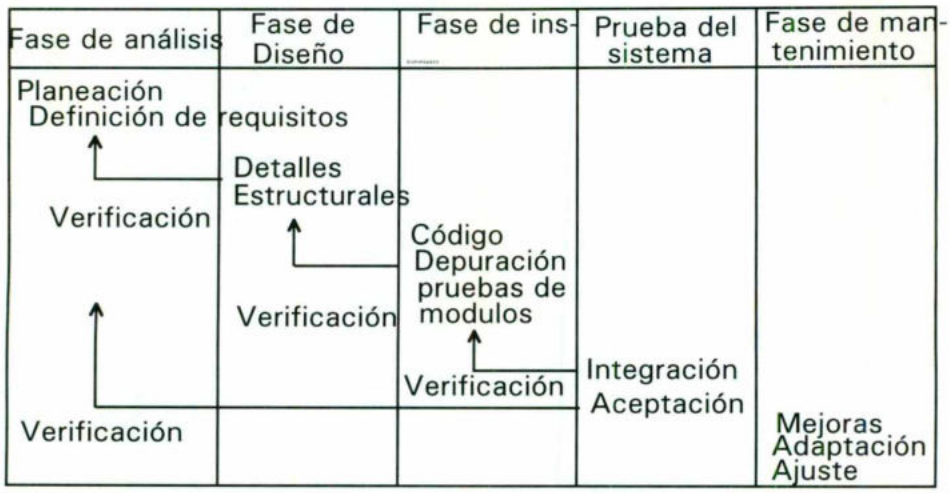
2.- Realizar un estudio de factibilidad para cada estrategia.

3.- Recomendar una estrategia de solución, indicando por qué se rechazan las otras.

4.- Desarrollar una lista de prioridades para las características del producto.

PLAN DE DESARROLLO

Un tercer paso es desarrollar un plan de desarrollo o plan del proyecto. Se comprenden varias consideraciones importantes; la primera es definir un modelo de ciclo de vida para el producto. Este ciclo incluye todas las actividades requeridas para definirlo, desarrollarlo, probarlo, entregarlo, operarlo y mantenerlo. Diferentes modelos de ciclo de vida hacen hincapié en distintos aspectos del ciclo, pero ninguno es apropiado para todos los productos. Es esencial definir un modelo de ciclo de vida para cada proyecto de programación, puesto que permite clasificar y controlar las diferentes actividades necesarias para el desarrollo y mantenimiento del producto. Los modelos de ciclo de vida que se analizan son los de : fases, costos, prototipos y versiones sucesivas.



Para realizar un adecuado plan de desarrollo se recomienda hacer:

1.- Definir un modelo de ciclo de vida y una estructura organizacional para el proyecto.

2.- Planear las actividades de administración para la configuración, control de calidad y validación.

3.- Determinar las herramientas por fase, técnicas, y notación por utilizar.

4.- Establecer estimados preliminares de costo para el desarrollo del sistema.

5.- Establecer un programa preliminar para el desarrollo.

6.- Establecer estimados preliminares de personal.

7.- Desarrollar estimados preliminares de recursos de cómputo necesarios para operar y mantener el sistema.

8.- Preparar un glosario de términos.

9.- Identificar fuentes de información, y referirse a ellas a lo largo del plan del proyecto.

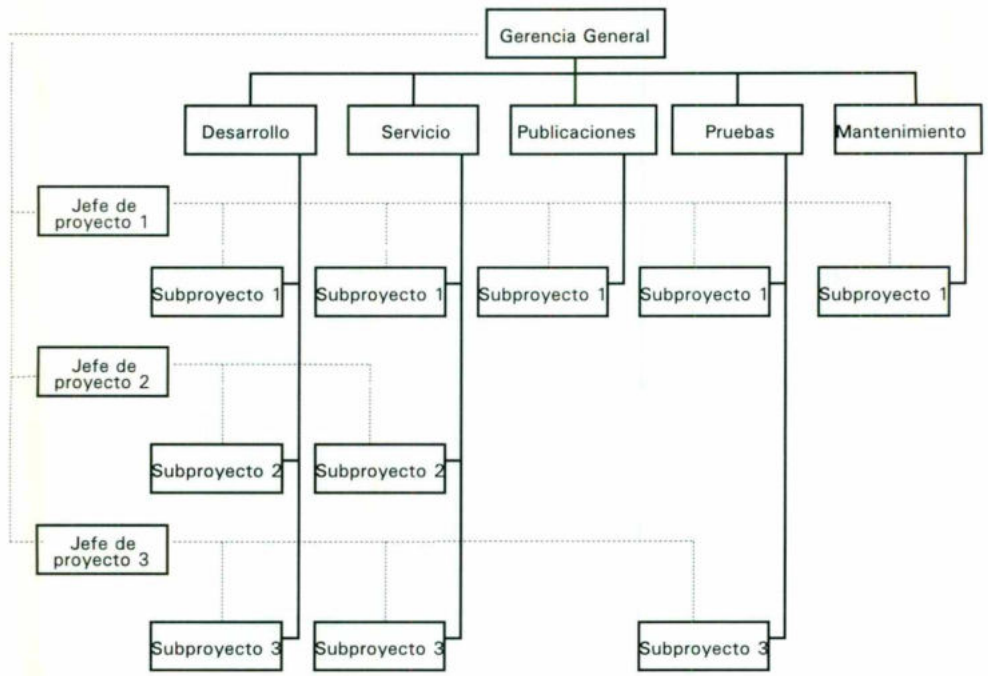
PLANEACIÓN DE UNA ESTRATEGIA ORGANIZACIONAL

Durante el tiempo de vida de un producto, se deben realizar varias actividades que comprenden planeación, desarrollo, servicios, publicaciones, control de calidad, apoyo y mantenimiento. Estas tareas deben ser organizadas, y algunas de las formas de hacerlo son :

Formato de proyecto.- El uso de este formato implica la integración de un equipo de programadores que lleven a cabo el proyecto de principio a fin y que realizan la definición, diseño, instrumentación, prueba y revisiones del producto, así como el desarrollo de los documentos de apoyo. Los miembros del equipo, por lo general, trabajan en él de uno a tres años y se asignan a nuevos proyectos cuando finalizan.

Formato funcional.- en este esquema un equipo diferente de programadores realiza cada fase del proyecto, y los productos, pasan de un equipo a otro conforme el producto va evolucionando. Una variación común del formato funcional comprende tres equipos: uno de análisis, otro de diseño e instrumentación, y un tercero de pruebas y mantenimiento. En este esquema un grupo de apoyo realiza las publicaciones, mantiene las instalaciones y proporciona el entrenamiento. Los miembros del equipo se rotan periódicamente para proporcionar desarrollo profesional y evitar el tedio de la superespecialización. Requiere más comunicación entre equipos, permite que el personal se especialice en ciertas áreas y que la documentación sea más clara a causa de las necesidades de comunicación.

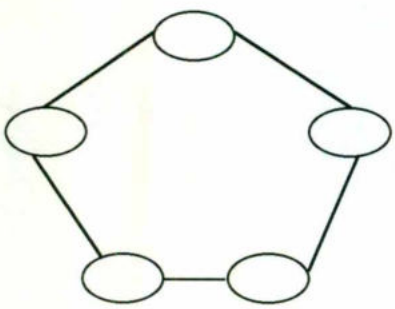
Formato matricial.- Cada función de las descritas tiene su propia administración y un equipo de gente dedicada exclusivamente a dicha función.



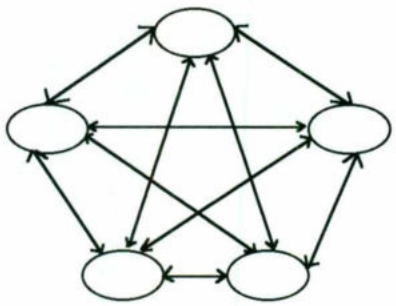
En las organizaciones matriciales cada quien tiene, por lo menos, dos jefes, y la necesidad de resolución de esta ambigüedad es el precio que hay que pagar por tener un proyecto más controlado. La organización matricial es cada vez más popular, puesto que la experiencia especializada, puede concentrarse en funciones específicas, lo que produce una utilización eficaz y eficiente del personal.

Es importante establecer que todo equipo de programación debe tener una estructura interna, la óptima estructura para un proyecto depende de su naturaleza y la del producto. Las estructuras básicas son :

Grupos democráticos.- Las metas y decisiones se definen por consenso, el liderazgo rota de un miembro a otro dependiendo de las tareas que se realicen y de las capacidades particulares de cada miembro.



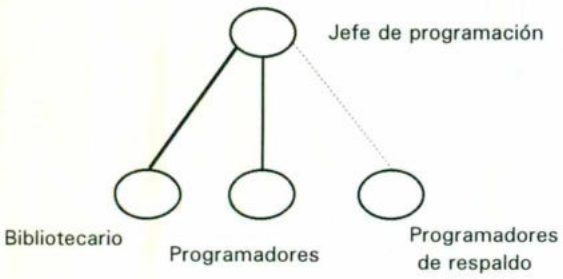
a) Estructura



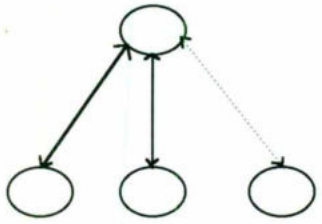
b) Trayectoria de comunicaciones

Grupos con jefe de programación.- Estos grupos son muy estructurados. El jefe diseña el producto, instrumenta partes críticas de él, y toma las decisiones técnicas importantes; también asigna el trabajo de los programadores, y éstos de número de dos a cinco, escriben el código, lo depuran, documentan y prueban.

Un bibliotecario de programas mantiene en un lugar accesible los documentos inherente al proyecto.

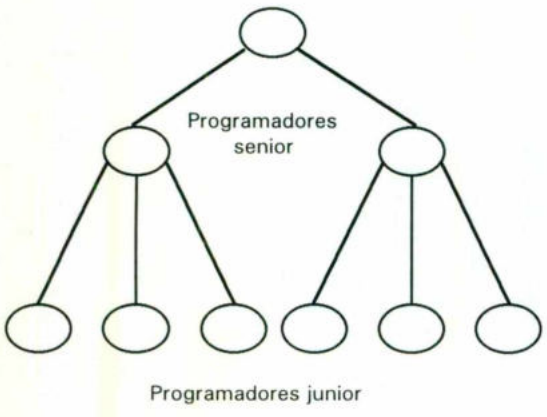


a) Estructura

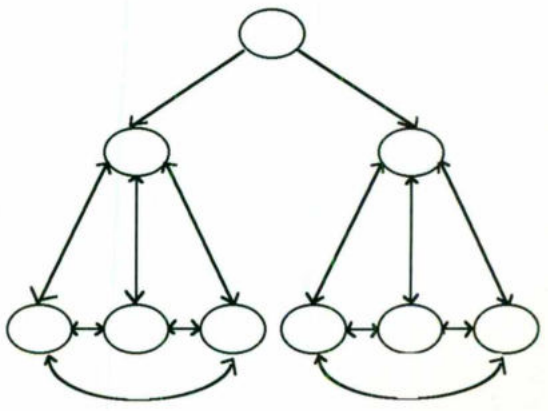


b) Trayectorias de comunicación

Grupos bajo jerarquía administrativa.- Ocupa un punto intermedio entre los grupos democrático y de jefe. El líder de proyecto asigna tareas, asiste a revisiones y recorridos, detecta áreas de problemas, balancea las cargas de trabajo y participa en actividades técnicas; es particularmente útil para el desarrollo de productos de programación con jerarquía, puesto que cada subsistema se puede asignar a un equipo distinto.



a) Estructura



b) Trayectorias de comunicación

CAPITULO 3
ESTIMACIÓN DE COSTOS DEL SOFTWARE

La estimación de costos de un producto de programación es una de las más difíciles y erráticas tareas de la ingeniería de software; es difícil hacer estimaciones exactas durante la fase de planeación de un desarrollo debido a la gran cantidad de factores desconocidos en ese momento. Reconociendo este problema, algunas organizaciones utilizan una serie de estimadores de costos; se prepara un estudio preliminar durante la fase de planeación y se presenta en la revisión de la factibilidad del proyecto. La estimación mejorada se muestra en las revisiones de los requisitos de programación y la estimación final se presenta durante la revisión preliminar del diseño.

FACTORES EN EL COSTO DEL SOFTWARE

Existen muchos factores que influyen en el costo de un producto de programación. El efecto de estos factores es difícil de estimar y, por ende también lo es el costo de esfuerzo en el desarrollo o en el mantenimiento. Entre los principales factores que afectan, se encuentran :

- **Capacidad del programador**, hablamos de su habilidad para solucionar problemas de programación y de su familiaridad con el área de aplicación.

- **Complejidad del producto**, existen tres categorías para los productos de programación : programas de aplicación, programas de apoyo, y programas de sistemas. Cada uno con su nivel de complejidad; los programas de apoyo son tres veces más difíciles de construir que los de aplicación, los de sistemas, a su vez, son tres veces más difíciles que los de apoyo. Para predecir el esfuerzo total en meses de programador que se requiere en un desarrollo, consideremos las siguientes ecuaciones:

Programa de aplicación : PM=2.4*(KDSI)**1.05
Programas de apoyo : PM=3.0*(KDSI)**1.12
Programas de sistema : PM=3.6*(KDSI)**1.20

donde KDSI son los millones de instrucciones de código entregadas con el producto.

El tiempo de desarrollo de un programa es de:

Programas de aplicación : TDEV=2.5*(PM)**0.38
Programas de apoyo : TDEV=2.5*(PM)**0.35
Programas de sistemas : TDEV=2.5*(PM)**.032

- **Tamaño del producto.** un proyecto grande de programación es obviamente más caro en su desarrollo que uno pequeño.

- **Tiempo disponible,** el esfuerzo total del desarrollo se relaciona con el calendario de trabajo asignado para la terminación del proyecto, varios investigadores han estudiado la cuestión del tiempo óptimo de desarrollo, y la mayoría concuerda con que los proyecto de programación requieren más esfuerzo si el tiempo de desarrollo se reduce o incrementa más de su valor óptimo. El esfuerzo de un proyecto es inversamente proporcional al tiempo de desarrollo elevado a la cuarta, E=K/(Td**4).

- **Nivel de confiabilidad requerido,** puede definirse como la probabilidad de que un programa desempeñe una función requerida bajo ciertas condiciones especificadas y durante cierto tiempo.

- **Nivel tecnológico,** se refleja en lenguaje utilizado, la máquina abstracta, las practicas y las herramientas de programación utilizadas.

TÉCNICAS DE ESTIMACIÓN DE COSTOS DEL SOFTWARE

Dentro de la mayor parte de las organizaciones, la estimación de costos de la programación se basa en las experiencias pasadas. Los datos históricos se usan para identificar los factores de costo y determinar la importancia relativa de los diversos factores dentro de la organización. Lo anterior, por supuesto, significa que los datos de costos y productividad de los proyectos actuales deben ser centralizados y almacenados para un empleo posterior.

La estimación de costos puede llevarse a cabo en dos formas:

a) Forma jerárquica hacia abajo.- se enfoca primero a los costos del nivel del sistema, así como a los costos de manejo de configuración, del control de calidad, de la integración del sistema, del entrenamiento y de las publicaciones de documentación, Los costos del personal relacionado se estiman mediante el examen del costo de proyectos anteriores que sean similares.

b) Forma jerárquica hacia arriba.- primero se estima el costo del desarrollo de cada módulo o subsistema; tales costos se integran para obtener un costo total.

En la práctica, ambas formas deben desarrollarse y compararse para que iterativamente se eliminen las diferencias obtenidas. Existen varias técnicas de estimación de costos, que utilizan una u otra de las formas, entre las que se menciona:

- **Juicio experto,** es la técnica más utilizada. Se basa en la experiencia, en el conocimiento anterior y en el sentido comercial de uno o más individuos dentro de la organización.

- **Técnica DELFI,** puede adaptarse a la estimación de costos de la siguiente manera :

1.- Un coordinador proporciona a cada experto la documentación con la definición del sistema y una papeleta para que escriba su estimación.

2.- Cada experto estudia la definición y determina su estimación en forma anónima; los expertos pueden consultar con el coordinador, pero no entre ellos.

3.- El coordinador prepara y distribuye un resumen de las estimaciones efectuadas, incluyendo cualquier razonamiento efectuado por alguno de los expertos.

4.- Los expertos realizan una segunda ronda de estimaciones, otra vez anónimamente, utilizando los resultados de la estimación anterior. En los casos que una estimación difiera mucho de las demás, se podrá solicitar que también en forma anónima el experto justifique su estimación.

5.- El proceso se repite tantas veces como se juzgue necesario, impidiendo una discusión grupal durante el proceso.

- **Estructura de división del trabajo,** es un organigrama jerárquico en donde se establecen las diferentes partes de un sistema (WBS). Un organigrama WBS de procesos identifica las actividades de trabajo y sus interrelaciones; algunos identificadores ocupan el WBS para realizar sus estimaciones de costos. Las ventajas de esta técnica son la identificación y contabilización de los diversos procesos y factores de productos de un sistema, así como la aclaración con exactitud de que costos se incluyen en la estimación.

- **Modelos de costo por algoritmos o módulos,** los costos se estiman mediante la adición de los costos de cada uno de los módulos o subsistemas que conforman al sistema.

El módulo constructivo de costos o COCOMO, es un modelo de costos por algoritmos, que a continuación se resume:

1.- Identificar todos los subsistemas y los módulos del producto.

2.- Estimar el tamaño de cada módulo y calcular el tamaño de cada subsistema y del sistema en total.

3.- Especificar los factores multiplicadores de módulos para cada uno; estos son: la complejidad del producto, la capacidad de programación, la experiencia de máquinas virtuales, y la experiencia de lenguajes modernos de programación.

4.- Calcular el esfuerzo para cada módulo, así como el tiempo de desarrollo; para lo anterior usar las ecuaciones de estimación nominal junto con los factores relevantes a cada módulo.

5.- Especificar los once multiplicadores restantes de cada subsistema.

6.- De los pasos 4 y 5, calcular el esfuerzo y el tiempo de desarrollo estimado para cada subsistema.

7.- Del paso 6, calcular el esfuerzo y tiempo de desarrollo totales para el sistema.

8.- Efectuar un análisis de sensibilidad sobre la estimación, estableciendo comparaciones para diversos factores.

9.- Sumar los otros ingredientes en el costo del desarrollo, como la planeación y el análisis, que no se hayan incluido antes.

10.- Comparar la estimación con otra obtenida a partir de la técnica DELFI, identificando y corrigiendo las diferencias en la estimación.

ESTIMACIÓN DEL NIVEL DE CONTRATACIÓN

La cantidad de personal requerido a través de un proyecto de desarrollo no es constante; por lo regular, la planeación y el análisis lo efectúa un grupo pequeño de individuos; el diseño arquitectónico, un grupo mayor, aunque todavía pequeño, y el diseño detallado lo realiza un grupo grande de personas. La fase inicial de mantenimiento puede requerir un número considerable de personas, pero este número deberá disminuir en poco tiempo. Si no existen mejoras o adaptaciones importantes, el número de personas para el mantenimiento permanecerá pequeño.

El número total de meses de programador y el tiempo de desarrollo total puede usarse para obtener el número de meses de programador y el tiempo necesario para cada actividad. Un estimado del número de empleados de tiempo completo requeridos para cada fase del desarrollo del proyecto se puede lograr con la división del número de meses de programador requeridos entre el tiempo disponible.

ESTIMACIÓN DE LOS COSTOS DE MANTENIMIENTO DE SOFTWARE

El mantenimiento de software suele necesitar de 40% a 60% y en algunos hasta 90% del esfuerzo total durante el ciclo de vida del proyecto; estas actividades comprenden agregar mejoras al producto, adaptar el producto para nuevos ambientes de proceso y corregir los problemas de los programas.

Una regla útil usada para la distribución del esfuerzo de las actividades de mantenimiento es asignar 60% del tiempo a mejoras, 20% a adaptación y 20% a la depuración o corrección de problemas.

CAPITULO 4
ETAPAS DEL DESARROLLO DE PRODUCTOS DE SOFTWARE

Las etapas del desarrollo de productos de software son también conocidas como las etapas del ciclo de vida de un producto de software. Diversos autores hacen la clasificación de esta etapas según su punto de vista, por lo cual se pueden encontrar una gran variedad de clasificaciones. Pero hay que hacer notar que todas surgen a partir de las siguientes fases base: Análisis, Diseño, Implementación y Mantenimiento. También de acuerdo a el tipo de metodología usada en el desarrollo del producto de software dependerán las etapas empleadas.

Debido a que cada etapa depende de varios factores para su desempeño, y cada etapa puede valerse de diferentes técnicas para lograr su objetivo, es complicado mencionar aquí todas las técnicas. Pero si mencionaremos el objetivo primordial de cada una de las cuatro etapas básicas.

ETAPA DE ANÁLISIS

En esta etapa, la meta es establecer en forma concreta y precisa que es lo que se busca. Es decir, que se va a hacer y como se va a hacer. Para lograr lo anterior se tienen que identificar todas las posible situaciones que se pueden ver involucradas en nuestra área de interés. Es en sí, una tarea de investigación y recopilación de información para llegar a conocer el más mínimo detalle que pueda influir en nuestro entorno.

Para realizar esta tarea existen diferentes técnicas, entre las que podemos mencionar a la entrevista, el cuestionario, la observación, etc. Pero no todas son aplicables en todos los casos, por lo tanto, su utilización esta condicionada a las características particulares de la situación a la cual se enfrenta el desarrollador.

La misma etapa de análisis presenta variaciones en su alcance, según algunos autores, todo lo mencionado en el capitulo 2 forma parte de la etapa de análisis; pero, otros mencionan que son actividades separadas. Pero al final de cuentas, ya sea conjuntamente o por separado, al llegar al final de la etapa de análisis, el desarrollador tiene que haber establecido que es lo que se va a solucionar (el problema), así como una alternativa de solución.

ETAPA DE DISEÑO

Dentro de la etapa de diseño el desarrollador tiene que especificar en forma detallada la manera en que se va a llevar a cabo la alternativa de solución establecida al término de la etapa de análisis. Es decir, establecer las herramientas, algoritmos, tiempos, estructuras y criterios para poder desarrollar la alterativa de solución elegida para satisfacer los requerimientos que dieron origen al desarrollo. También para esta etapa existen infinidad de técnicas que ayudarán para cumplir con el objetivo, pero es imposible comentarlas todas.

ETAPA DE IMPLEMENTACIÓN

La etapa de implementación consiste principalmente en traducir todos los lineamientos establecido en al etapa anterior al lenguaje de programación establecido. es decir, sentarse frente a la computadora y llevar a cabo la programación o generación de lo que será el producto final de un desarrollo de software.

En esta fase también se contemplan la realización de pruebas que midan el nivel de aceptación de nuestro producto de acuerdo a los criterios definidos, así como la liberación del mismo, hasta haber sido autorizado por salvar todas las pruebas aplicadas

ETAPA DE MANTENIMIENTO

Esta es una de las etapas más importantes del desarrollo de software, ya que es la fase donde se retroalimenta al desarrollador. Sus actividades consisten en monitorear el desempeño del producto liberado, para solucionar los problemas que se presenten y volver a reanudar el proceso de desarrollo hasta que sea factible.

CAPITULO 5
INGENIERÍA DE SOFTWARE CON LA METODOLOGÍA DE OBJETOS.

La programación orientada a objetos no puede por si sola proporcionar todos los beneficios de una buena ingeniería de software. La orientación de Objetos puede ser empleada durante todas las fases del proceso de desarrollo de software.

Cuando elaboramos un producto de Software, creamos modelos del área de aplicación que nos interesa. Un modelo puede incorporar un sistema, en un área de una empresa o contemplar a la empresa en su totalidad.

El modelo representa un aspecto de la realidad y se construye del modo que nos ayude a comprender a ésta. La sencillez de un modelo, comparado con la realidad, nos permite un alto grado de manejo sobre éste, lo que nos da una gran ayuda a idear sistemas o rediseñar áreas de la empresa.

CONCEPTOS BÁSICOS DE LA TEORÍA OO

Para entender la metodología orientación a objetos, debemos definir primeramente sus elementos. Una vez comprendidos éstos, nos será más fácil poder comprender este paradigma.

El primer concepto por explicar el del *OBJETO* mismo; y basándonos en lo que se menciono lineas atrás, en que los modelos que desarrollamos surgen a partir de la concepción de nuestro entorno. Diremos que un objeto es cualquier cosa que nos rodea en nuestra vida real, tal como una pelota, un ave, una nube, una persona, etc.

Pero en el desarrollo de software orientado a objetos, nos interesa conocer el comportamiento de dicho objeto. Para realizar dicho comportamiento (operaciones), el objeto debe almacenar datos (información) que le permitan conocer su patrón de comportamiento. Por lo tanto diremos que : Un *objeto* es una entidad (cualquier cosa real o abstracta) capaz de almacenar un estado (información) y que ofrece varias operaciones (comportamiento) que nos permiten modificar su objeto.

Tal vez suene contradictorio, pero, es necesario aclarar que la información y el comportamiento almacenado en un objeto, no propiamente esta almacenado en el objeto. Es decir, que no todos los objetos van a tener almacenada su propia información y su propio comportamiento. ¿Cómo es esto?, bien, lo intentaremos explicar mediante dos concepto: *Tipo de objetos e instancias*.

Un tipo de objeto es una categoría de objeto, y un objeto esta definido como una instancia de un tipo de objeto.
En otras palabras, Cuando hablamos de un empleado, nos referimos a un tipo de objeto llamado EMPLEADO, ya que este tipo de objeto es el que tiene almacenada la información y el comportamiento propio de una persona a la cual podemos clasificar como empleado. Esta persona se puede llamar "Pedro", "Juan" o "Maribel", los cuales no contienen cada uno su propia información y comportamiento sino que comparten lo almacenado en EMPLEADO; pero estos serán lo que se llama una INSTANCIA de un tipo de objeto llamado EMPLEADO.

Para que un objeto pueda reflejar su comportamiento es necesario que cuente con lo que llamaremos *MÉTODOS*, los cuales especifican la forma en que se controlan los datos de un objeto. Los métodos en un tipo de objeto SOLO hacen referencia a las estructuras de datos de este tipo de objeto. Y no pueden tener acceso a las estructuras de datos de otro tipo de objeto, sino es mediante el uso de mensajes.

Un *MENSAJE* es la manera de estimular a un objeto a realizar un método soportado por su comportamiento, estos mensajes únicamente son mandados al objeto pero en realidad no se sabe a ciencia cierta que proceso realizará el objeto estimulado, pero se cuenta con la seguridad de que realizará eficientemente su tarea. A ésta característica se le conoce como "encapsulamiento".

El *Encapsulamiento* es sencillamente el ocultamiento de la información de los tipos de objetos, por lo cual decíamos que nadie conoce que información guarda y que ó como realiza sus operaciones.

Ahora describiremos los conceptos de *CLASE e INSTANCIAS* , una clase es la implantación de un tipo de objeto, es decir, especifica una estructura de datos y los métodos operativos permisibles que se aplican a cada uno de sus objetos. Tal vez se considere que esta definición hace referencia a un tipo de objeto, pero, una clase y un tipo de objeto son dos cosas diferente. La clase es algo más que la simple definición de estructuras de datos y manipulaciones que pueden hacerse sobre estas. Un clase puede ser la definición de varios tipos de objetos juntos, que mediante la abstracción describe todas las características comunes de los datos que forman parte de la clase. Por ejemplo, digamos, una clase puede ser el ser humano, y los tipos de objetos para esta clase son seres humanos asiáticos, seres humanos europeos, seres humanos americanos, seres humanos africanos, etc., y las instancias para esta clase pueden ser John, Chang, Jean, etc.

La instancias creadas a partir de clases nos generan un comportamiento dinámico apropiado para poder modelar la realidad en términos de objetos. Pero, bajo este enfoque, el dinamismo de las instancias provocaría ciertas incompatibilidades si éstas no pudieran amoldarse a las características de las demás instancias con las cuales se relacionarán. Surge de esta manera el concepto de *POLIMORFISMO*, el cual se refiere a la capacidad que debe tener una instancia para poder estimular y ser estimulada sin importar las características propias de la instancia a la cual estimula o viceversa. Es decir, el emisor de un estímulo no necesita conocer la clase de la instancia receptora. La instancia receptora puede pertenecer a una clase arbitraria.

Por ultimo definiremos el concepto de *HERENCIA*. La herencia se refiere a la capacidad que tiene una instancia de hacer uso de la información y comportamiento almacenados en la clase a la cual pertenece y desciende. Es decir, Los hijos heredan las características de sus padres, y a la vez los hijos de los hijos heredarán estas mismas características más algunas nuevas. Dicho de otra manera, las instancias pueden ser modificadas e incluírseles nuevos datos y métodos, de esta manera se pueden convertir en clase y heredar a sus instancias descendientes todos sus atributos. La combinación de dos o más clases para heredar su información a una instancia, se le conoce como "*HERENCIA MÚLTIPLE*".

PARADIGMA ORIENTADO A OBJETOS VERSUS SA/SD

El paradigma orientado a objetos sugiere una metodología que difiere del tradicional proceso de Desarrollo, Análisis Estructurado / Diseño Estructurado (SA/SD) propuesto por Yourdon en muchas maneras de forma significativa. El SA/SD ayudan a modelar los "Procesos" que manipulan los datos de entrada obtenidos para producir la salida deseada.

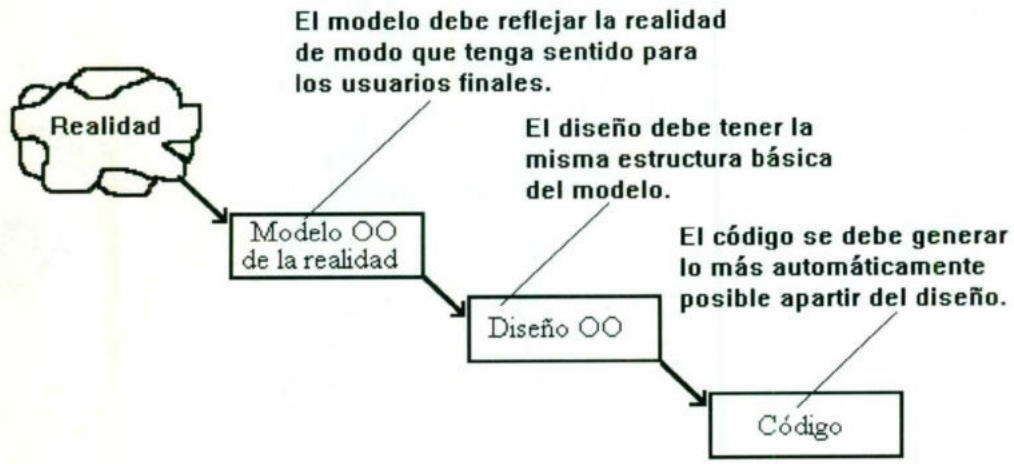
El primero de estos pasos en SA/SD involucran la creación preliminar de diagramas de contexto y diagramas de flujo de datos. La siguiente fase consta de la creación de un modelo "Esencial", el modelo esencial es considerado para ser la combinación del modelo "Ambiental" y el modelo "Conductual". El modelo conductual esta hecho sobre esencias de los modelos de los "Procesos esenciales" y la interconexión entre estos procesos. Esto es, una parte esencial de la metodología de Yourdon es el modelado de los procesos o métodos que son usados para transformar un dato de entrada en la salida deseada.

En las últimas dos décadas las metodologías de desarrollo de software, la orientada a procedimientos y la de análisis estructurado, han tenido un extenso uso en varias áreas. Reutilizar el código existente no ha sido una consideración primordial en las fases de diseño o de desarrollo para la mayoría de los esfuerzos de desarrollo de software. Las herramientas usadas en el diseño y desarrollo de software no están orientadas para crear o incorporar módulos de código reutilizable.

Varios investigadores tienen ahora un resurgimiento con los modelos de desarrollo de software orientado a objetos. La literatura indica que los componentes necesarios de las metodologías orientadas a objetos son fuertemente debatidos. No ha emergido todavía como la metodología estandar. De cualquier forma los investigadores tienen varias cuestiones sobresalientes. Una cuestión es la de las especificaciones de software, ésta percibe ampliamente que la especificación propia de un sistema puede resultar sobresaliente en los objetos y las relaciones entre objetos. Otra cuestión es la habilidad o capacidad para identificar y diferenciar un buen diseño orientado a objetos de uno malo. Para ser capaz de hacer esto, se requiere de alguna medida que pueda facilitar la clasificación de la legibilidad, reusabilidad y extendibilidad. La reusabilidad del código es otra cuestión que es considerada por el paradigma de programación orientada a objetos.

TEORÍA ORIENTADA A OBJETOS

Con la orientación a objetos, la forma de modelar la realidad difiere del análisis convencional. Modelamos el mundo en términos de tipos de objetos y lo que le ocurre a estos, lo cual nos lleva también a diseñar y programar sistemas de forma orientada a objetos. En la siguiente figura ilustraremos la manera en que normalmente construimos sistemas:



Primeramente creamos un modelo de la realidad o área de interés; después convertimos el modelo en un diseño, y por último lo transportamos a un código. Todo modelo debe ser capaz de ilustrar la manera en que los usuarios finales perciben el área o lo que los usuarios desean que realice el sistema.

La construcción de modelos mediante un enfoque OO refleja la realidad de una manera más natural que el punto de vista de los enfoques tradicionales. Ya que, si lo vemos de esta manera, la realidad consta de objetos y eventos que cambian el estado de dicho objeto.

BENEFICIOS DE LA TECNOLOGÍA OO

La siguiente lista nos presenta la mayoría de los beneficios que se pueden obtener al desarrollar productos de software con la Orientación a Objetos:

- **Reutilización.** Nos referimos a que las clases son diseñadas e implementadas de tal manera que se pueden adaptar y ser reutilizadas en muchos otros sistemas diferentes al cual originalmente fueron orientadas.

- **Estabilidad.** Las clases diseñadas para la reutilización repetida se vuelven estables, por decirlo de otra manera, se convierten en constantes que podemos utilizar de una manera abierta.

- **El diseñador piensa en términos del comportamiento de objetos y no en detalles de bajo nivel .** No se sabe como funciona determinado objeto, pero se esta consciente de que realiza lo que debe realizar. Por lo tanto, lo que se debe de conocer es la función que realiza cada objeto y la manera en que podemos comunicarnos con el.

- **Se construyen clases cada vez más complejas.** A partir de una clase plenamente probada, se pueden generar nuevas clases mediante otras clases. Generando clases mas complejas que dan solución a problemas mas complejos.

- **Confiabilidad.** Si se construyen sistemas con clases estables altamente probadas, es muy probable que el nivel de fallas sea considerablemente menor que el presente en los sistemas desarrollados completamente nuevos.

- **Un diseño más rápido.** Debido a que se cuenta con clases o componentes ya existentes, los cuales son confiables y además poseen una capacidad de adaptabilidad, la tarea de diseño se facilita y reduce el tiempo consumido en la misma aplicando técnicas tradicionales.

- **Diseño de mayor calidad.** Debido a que las clases o componentes son constantemente revisados y depurados, un diseño que utilice estas herramientas tendrá mayor calidad.

- **Integridad.** Las estructuras de datos sólo se pueden utilizar con métodos específicos.

- **Programación más sencilla.** Los programas se conjuntan a partir de piezas pequeñas, las cuales son más fáciles de realizar. A estas piezas se les llama métodos, y los métodos se van incluyendo al comportamiento del objeto de una manera gradual, según se van realizando.

- **Mantenimiento más sencillo.** Dado que cada clase realiza sus funciones de manera independiente con respecto a las demás clases, el mantenimiento a los sistemas se facilita; ya que sólo se le da mantenimiento a una sola clase y no a todo el conjunto.

- **Inversión.** Los implantadores eficientes encuentran que se pueden generar ideas con rapidez mediante las herramientas OO más poderosas del CASE ejecutándose en una estación de trabajo. Las herramientas los animan a inventar e implantar con rapidez sus ideas. Las personas brillantes pueden ser mucho más creativas.

-Ciclo vital dinámico. El objetivo del desarrollo de un sistema cambia con frecuencia durante la implantación. Las herramientas OO del CASE facilitan los cambios a la mitad del ciclo vital. Esto permite a los implantadores de software conocer mejor a los usuarios finales, adaptase a los cambios de las empresas, afinar los objetivos conforme el sistema se consolida, mejorar de manera constante el diseño durante la implementación.

-Modelado más realista. Dado que la OO considera el mundo real como objetos y su comportamiento, y la realidad esta conformada por objetos y sus comportamientos. El análisis Orientado a Objetos nos facilita modelar el mundo de una manera más real.

-Mejor comunicación entre los profesionales de los sistemas de información y los empresarios. La comprensión del sistema por parte de los usuarios se da de una manera más natural, ya que las personas informáticas le trasmiten sus expectativas en un lenguaje más comprensible para ellos.

-Modelos empresariales inteligentes. Debido a que los empresarios deben expresar las reglas bajo las cuales quieren que se comporte su empresa, así como la forma en que dicho comportamiento debe ser modificado. El diseño de la aplicación es concebido de una manera que se logre la mayor automatización posible.

-Especificaciones declarativas y diseño. Se debe de establecer explícitamente todo lo que se desea y necesita para permitir que el diseñador piense como un usuario final en vez de pensar como una computadora.

-Un interfaz de pantalla sugestiva para el usuario. El usar iconos o elementos de menú desplegados en la pantalla y relacionados con los objetos facilita al usuario el manejo de la aplicación ya que es más fácil ver y apuntar que escribir y recordar.

-Interacción. Debido a que las clases son diseñadas para generar un estandar, el software de varios proveedores puede funcionar como un conjunto.

-Independencia del diseño. Las clases son diseñadas para funcionar independientemente del ambiente de plataformas, hardware y software bajo las cuales vayan a ser utilizadas.

CARACTERÍSTICAS DE LAS TÉCNICAS ORIENTADAS A OBJETOS

1.- Cambian nuestra forma de pensar sobre los sistemas. Debido a que nuestro entorno esta formado por objetos, es más natural pensar en términos de objetos, sus eventos y sus mecanismos de activación.

2.- Los sistemas suelen construirse a partir de objetos ya existentes plenamente probados, lo cual nos lleva a una reducción considerable tanto en costo, tiempo de desarrollo y un mayor grado de confiabilidad en el sistema.

3.- Podemos construir nuevos y más complejos objetos a partir de otros objetos, los cuales a su vez están construidos de objetos, etc. Esto no genera la posibilidad de crear aplicaciones muy complejas.

4.- Podemos crear sistemas con un funcionamiento correcto de una manera más fácil, debido a que las clases OO están diseñadas para poder ser utilizadas por cualquier otra aplicación; además, éstas están diseñadas para que puedan ser construidas, depuradas, y modificadas con relativa facilidad.

5.- Se cuenta con un depósito de herramientas CASE, el cual, debe contener una vasta y creciente biblioteca de tipos de objetos, ya sean comprados o producidos propiamente.

6.- Las técnicas orientadas a objetos se ajustan de manera natural a la tecnología CASE.

HACIA UNA METODOLOGÍA DE DESARROLLO DE SOFTWARE ORIENTADA A OBJETOS

Hace unos cuantos años, antes de que una metodología de desarrollo de software Orientada a Objetos (OOSD) emergiera como un producto probado. Varios investigadores habían hecho varias propuestas para desarrollar software en un ambiente orientado a objetos. Wirfs-Brock y Wilkerson compararon dos técnicas de diseño orientadas a objetos: la propuesta data-driven y la propuesta responsibility-driven. En el diseño data-driven, los objetos son identificados contestando las siguientes preguntas:

* ¿ Que estructura representa este objeto ?
* ¿ Que operaciones pueden ser realizadas por este objeto ?

El diseño responsibility-driven, por otro lado, se enfoca a contestar las siguientes preguntas:

* ¿ Que acciones responden para este objeto ?
* ¿ Que información comparte este objeto ?

Wirfs-Brock y Wilkerson llegaron a las siguientes conclusiones:

La propuesta data-driven para el diseño orientado a objetos se enfoca en la estructura de los datos en un sistema. Esto resulta de la incorporación de información estructurada en la definición de las clases. Hecho de esta manera se viola el encapsulamiento. La propuesta responsibility-driven enfatiza el encapsulamiento de la estructura de objetos y del comportamiento de los objetos. Para enfocarse a las responsabilidades contractuales de una clase, el diseñador es capaz de posponer las consideraciones de la implementación hasta la misma fase de implementación.

Shlaer y Mellor propusieron una metodología de análisis orientada a objetos que involucraba la construcción de tres tipos de modelos formales: un modelo de información, un conjunto de modelos de estado indicando el ciclo de vida, y un conjunto de modelos de proceso. Ellos consideraban al modelo de información "la piedra angular del análisis orientado a objetos, el cual es usado para identificar las entidades conceptuales del mundo que se va a modelar." El modelo de información ayuda a describir el mundo real en términos de objetos, y de las relaciones o asociación entre dos o más conjuntos de entidades del mundo real abstraídas como objetos. El ciclo de vida de los objetos y de las relaciones pueden ser modelados en términos de estados que indique la condición de un objeto, eventos que inicialmente cambian a la siguiente fase del ciclo de vida, y acciones (procesos) que tienen lugar cuando un objeto de algún tipo cambia de estado y arriba a un nuevo estado. El modelo de estado representa el comportamiento de un objeto o relación. Shlaer y Mellor sugirieron construir un conjunto de modelos de proceso en términos de un diagrama de flujo de datos separado para cada estado, en cada modelo de estado.

Ed Yourdon presento una propuesta de cinco pasos para el desarrollo de software orientado a objetos:

Paso 1 Identificación de objetos.
Paso 2 Identificación de estructura.
Paso 3 Identificación de sujetos.
Paso 4 Definición de atributos.
Paso 5 Definición de servicios.

ANÁLISIS ORIENTADO A OBJETOS

Como ya mencionamos anteriormente, la tarea de la actividad de un análisis, es realizar una exhaustiva investigación para determinar que es lo que deseamos realizar y como lo vamos a realizar.

En un análisis orientado a objetos, se deben construir dos tipos de modelos de la realidad estrechamente relacionados: el primero es un tipo de modelo que representa a los tipos de objetos y sus estructuras, clases, relaciones entre los objetos y herencias; se le conoce como Análisis de la Estructura de Objetos. El segundo es un modelo que refleja lo que le ocurre a los objetos así como su comportamiento a través del tiempo; se le conoce como Análisis del Comportamiento de Objetos.

a) Análisis de la estructura de objetos

El análisis de la estructura de objetos define las categorías de los objetos que percibimos de nuestro medio ambiente y las formas en que los asociamos.

La finalidad de éste análisis es identificar lo siguiente:

¿ Qué son los tipos de objetos y cómo se asocian ?,

¿Cómo se organizan los tipos de objetos en supertipos y subtipos? y,

¿Cuál es la composición de los objetos complejos ?.

Durante este tipo de análisis, se debe preocupar más por identificar los tipos de objetos que por identificar los objetos individuales en un sistema.

Los tipos de objetos son importantes, puesto que crean los bloques conceptuales de construcción para el diseño del sistema, además, que los tipos de objetos son un índice para los procesos del sistema, ya que en un ambiente orientado a objetos todos los procesos son realizados mediante operaciones asignadas a los objetos que intervienen en dichos procesos. Es decir, si un proceso consta de correr cierta distancia y al final realizar un brinco, y tenemos un objeto llamado "persona" que dentro de sus eventos o atributos le fue declarado la función de saltar y correr; entonces dicho proceso puede ser realizado mediante dos funciones por una instancia del tipo de objeto "persona".

Una actitud normal como persona es darle ciertas categorías a los objetos. Es decir, para una persona, un objeto representa un rol, un papel, una parte de su realidad; mientras que para otra persona el mismo objeto representa un papel diferente, un rol diferente, una parte de la realidad diferente.

Aunque esta forma de conceptualizar las cosas pudiera verse como una actitud muy arbitraria, así funciona nuestra mente. De hecho, esta es la tarea a desarrollar en el primer paso de un análisis OO: "La determinación de los objetos involucrados en el sistema". Este paso consiste en definir los tipos de objetos que conforman el fragmento de mundo o realidad que vamos a solucionar. Los tipos de objetos que definimos o usamos son variados ya que los elegimos en base a la compresión de nuestro alrededor. Nosotros elegimos la forma de dar categorías a nuestro mundo, podemos caracterizar al mismo objeto de varias formas.

Los tipos de objetos clasifican a todos los objetos que conforman parte de nuestro entorno, es decir, si en el entorno que vamos a analizar nos encontramos que algunos de los elementos u objetos que participan son "Pedro", "Juan" y "María" (los cuales son personas), tendremos que crear un tipo de objeto que defina y clasifique a estos objetos; a este tipo de objeto lo podemos definir como "PERSONA" y de esta manera definimos y clasificamos a la persona llamada "Pedro", a la persona llamada "Juan" y a la persona llamada "María".

ASOCIACIÓN DE OBJETOS

Después de determinar los tipos de objetos que conforman nuestro entorno, el siguiente paso es establecer las relaciones que se presentan entre los diversos tipos de datos declarados. Estas relaciones deberán de especificar su significado y el número de objetos de un tipo que se relacionan con los de otro tipo. Esta manera de conceptualizar las relaciones da un mayor sentido y nivel de comprensión a las asociaciones.

JERARQUÍAS DE GENERALIZACIÓN

La siguiente acción para adquirir mayor conocimiento sobre nuestra área de interés, es jerarquizar de lo más general a lo más particular. En otras palabras, debemos generalizar los tipos de objetos . Entendamos por generalizar, el acto de distinguir un tipo de objeto como más general. De este concepto se desprenden los conceptos de SUPERTIPO y SUBTIPO, los cuales podemos definir en pocas palabras como padre e hijo respectivamente; es decir, todas las características que posea el supertipo le serán transmitidas a todos sus subtipos. Un subtipo puede derivarse de uno o más supertipos.

JERARQUÍAS COMPUESTAS

Algunos tipos de objetos son considerados como complejos cuando dichos objetos están conformados por otros objetos. Por ejemplo, un automóvil esta formado por chasis, llantas y motor. A su vez el motor está formado por el monoblock, válvulas, pistones, etc. Además, un pistón es un objeto complejo formado por anillos, una biela y una cabeza; así sucesivamente hasta llegar a la partícula más mínima.

b) Análisis del Comportamiento de Objetos

Ya clasificados e identificados los objetos mediante tipos de objetos, es necesario realizar la segunda parte del análisis orientado a objetos: "Análisis del Comportamiento de los Objetos".

En esta segunda fase de análisis se debe identificar la siguiente información:

-¿ Cuales son los estados que puede presentar un objeto ?
-¿ Qué transiciones de estado se pueden dar ?
-¿ Qué eventos ocurren ?
-¿ Qué operaciones se llevan a cabo ?
-¿ Qué interacciones ocurren entre objetos ?
-¿ Cuales son las reglas de activación que se utilizan para reaccionar ante el evento ?
-¿ Como se representan las operaciones en los métodos ?

ESTADOS DE UN OBJETO

Los tipos de objetos pueden ser definidos como los posibles ESTADOS que se presentan en el ciclo vital de un objeto, es decir, supongamos que una instancia del tipo de objeto "persona" puede ser considerada como un tipo de objeto:

-Persona hombre, ó
-Persona mujer

De esta manera podemos decir que la instancia a la cual nos referimos posee un estado, ya sea el de hombre o bien el de mujer. Sin embargo, un objeto puede tener una gran variedad de perspectivas de ciclos vitales. En otras palabras, diremos que los objetos pueden poseer al mismo tiempo estados de diferentes tipos de objetos (como se menciono en secciones anteriores, puede ser una instancia de varios tipos de objetos).

"El estado de un objeto es la colección de los tipos de objetos que se aplican a él." también podemos decir que "el estado de un objeto es la colección de asociaciones que tiene un objeto".

EVENTOS

El medio ambiente en el cual nos desarrollamos esta lleno y controlado por eventos (nada sucede si no ocurre un evento), por ejemplo: Pedro compra un libro, ceci sale de vacaciones, la pelota bota, el perro corre, etc,.

Dentro del análisis Orientado a Objetos, no se busca conocer cada evento que ocurra a un objeto; sólo se desea conocer los tipos de eventos. Hablamos de tipos de eventos de manera similar a como hablamos de los tipos de objetos.

Los tipos de eventos indican los cambios sencillos en el estado de un objeto, básicamente, los tipos de eventos describen las siguientes formas de cambios de estado:

-Un objeto se crea.

-Un objeto se termina.

-Un objeto se clasifica como una instancia de un tipo de objeto.

-Un objeto se desclasifica como una instancia de un tipo de objeto.

-Un objeto cambia de clasificación.

-El atributo de un objeto cambia.

EL CICLO VITAL DE UN OBJETO

La mayoría de los objetos tiene un ciclo vital en el que una sucesión de eventos pueden ocurrirle y cada uno de estos modifica su estado.

Para representar un ciclo vital de un objeto, podemos desarrollar un diagrama de reja como el que se muestra a continuación:

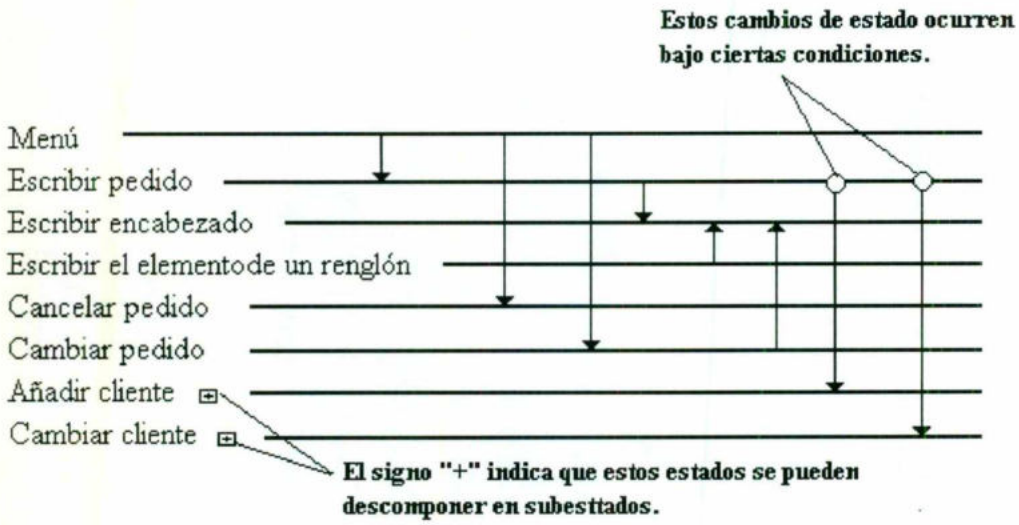


Diagrama de reja que muestra cómo un objeto diálogo en pantalla puede tener varios estados, cada uno de los cuales muestra una pantalla diferente. Las líneas horizontales representan los estados; las líneas verticales muestran la transición entre los estados.

En el diagrama anterior, el estado menú del objeto diálogo en pantalla puede tener cambio al estado escribir pedido, al estado cancelar pedido, o bien al estado cambiar pedido; el estado escribir pedido puede cambiar al estado escribir encabezado y bajo ciertas condiciones lo puede hacer al estado añadir cliente ó al estado cambiar cliente; etc.

INTERACCIONES ENTRE TIPO DE OBJETOS

Los diagramas de reja sirven únicamente para expresar el ciclo vital de un objeto particular. Sin embargo, la mayoría de los procesos requieren la interacción de varios objetos; por lo cual es necesario desarrollar un diagrama que nos muestre la forma en que diversos objetos se relacionan.

A continuación se muestra un diagrama de red del proceso de solicitud de pago de salario a un objeto EMPLEADO. Para efectuarlo hay que enviar una solicitud para calcular y regresar una cantidad de IMPUESTO. Al darle este cambio de estado, se solicita crear un objeto cheque. El resultado es que el estado del objeto empleado cambia al de empleado pagado.

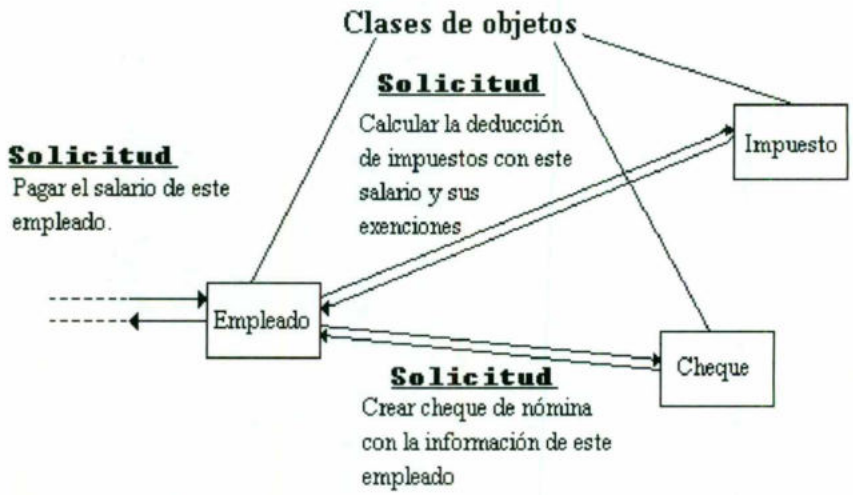
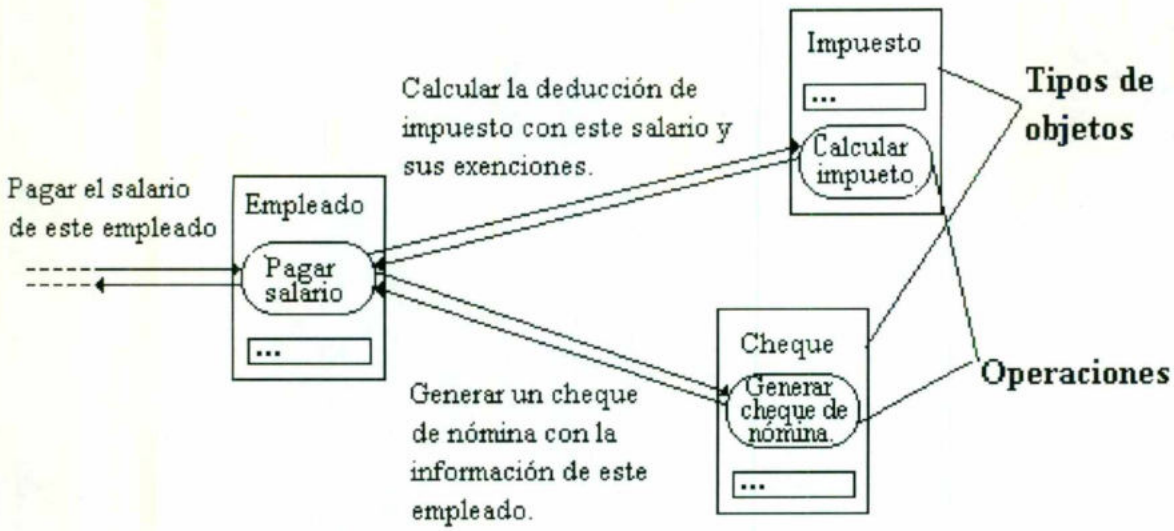


Diagrama de mensaje que se transfieren entre clases.

El diagrama de red muestra como los distintos tipos de objetos cambian de estado y pueden solicitar a otros objetos que cambien su estado en el proceso. Pero, es necesario desarrollar un diagrama que muestre las operaciones necesarias para el proceso de la nómina.



Extensión de la figura anterior que muestra las operaciones invocadas.

Para realizar un diagrama de red en esta manera es necesario implantar los tipos de objetos como clases y las operaciones se convierten en operaciones del programa OO. El expresar los requisitos de procesamiento en esta forma, facilita el trabajo del diseñador. Sin embargo, no muchos usuarios conceptualizan los procedimientos de su aplicación en términos de tipos de objetos saturados de operaciones y que hacen distintos tipos de solicitudes. Prefieren usar un formato de guión cuando se dan eventos que activan operaciones, las cuales, a su vez, producen eventos que activan otras operaciones , etc. A este formato se le llama esquema de eventos, y representa los cambios en el ciclo vital de otro objeto.

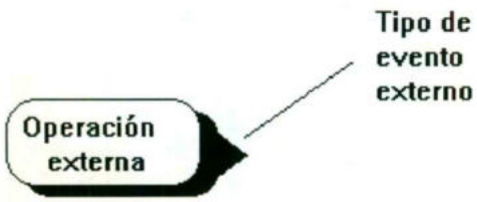
OPERACIONES

Cuando hablamos de una operación, nos referimos a una unidad de procesamiento que puede ser solicitado. Cuando esto sucede implementamos un método para realizar el procedimiento. Dicho método especifica la forma en cómo se debe realizar la operación.

Cuando una operación es invocada, se convierte en una instancia de una operación. Una operación puede o no cambiar el estado de un objeto, si lo cambiara, ocurriría un evento. La representación gráfica de una operación es mediante cuadros con esquinas redondas. Los tipos de eventos se representan mediante triángulos sólidos negros conectados a la caja.

FUENTE EXTERNA DE EVENTOS

Los eventos son cambios de estado que un sistema puede conocer y reaccionar ante ellos de alguna manera. Muchas de las veces, estos eventos son producidos por operaciones externas al sistema. Para gráficar estos casos utilizamos la siguiente figura:

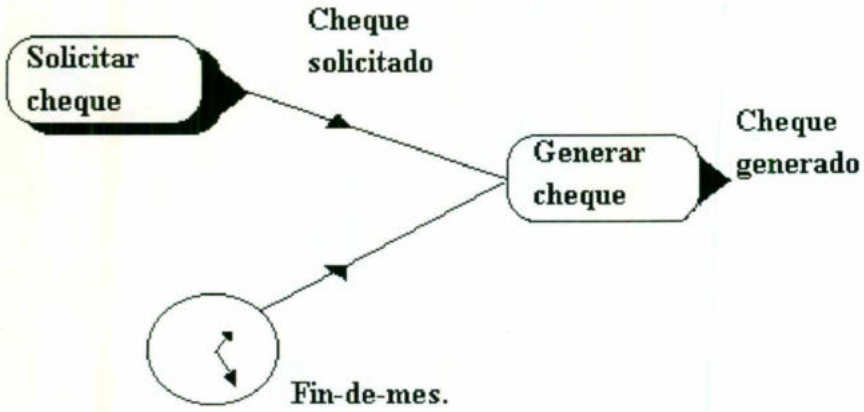


Una forma particular de identificar una fuente externa es mediante un reloj externo, el cual nos indicará que un proceso externo emitirá señales de reloj con cierta frecuencia predeterminada (por ejemplo, cada minuto, al final del día, cada semana, mes, o año, etc.). Así, el evento se producirá cada que el reloj externo emita su señal. El reloj externo se representa con la siguiente figura:

REGLAS DE ACTIVACIÓN

Como ya menciona con anterioridad, cuando ocurre un evento, es regla general que se realice una o más operaciones. Llamaremos reglas de operación a aquellas que definen la relación entre la causa y el efecto.

Siempre que ocurra un evento, la regla de activación invoca a una o más operaciones ya definidas, de la misma manera que un evento activa una ó más operaciones, una operación puede ser activada por dos o más eventos (o reglas de activación). La siguiente figura nos muestra como la operación generar cheque es activada por el evento solicitar cheque o por el evento fin-de-mes.



La operación producir cheque se activa cuando ocurre cualquiera de los eventos cheque solicitado o fin-de-mes.

CONDICIONES DE CONTROL

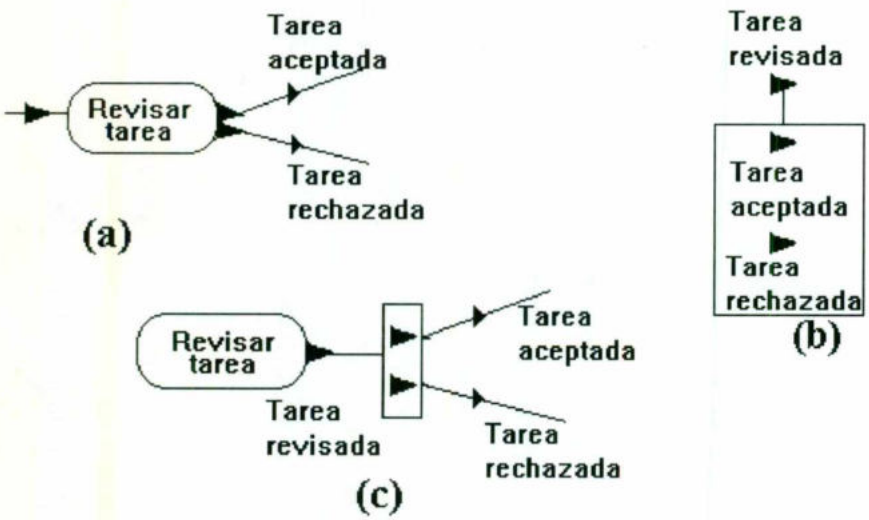
Sabemos que una operación puede ser invocada por varias reglas de activación, pero en ocasiones es necesario verificar ciertas condiciones antes de activar el proceso o la operación. Surge aquí el concepto de condiciones de control.

Una condición de control garantiza que un conjunto evento está completo antes de proceder con una operación. Las condiciones de control se grafican con un rombo antes de la operación, como se muestra a continuación.

SUBTIPOS Y SUPERTIPO DE EVENTOS

Sabemos que un evento cambia el estado de un objeto, pero también este evento puede tener subeventos. Analógicamente podríamos decir que es como una pelota roja y una pelota azul; los dos son pelotas, pero diferentes entre si. Es decir, tenemos un supertipo llamado pelotas y dos subtipos: pelota roja y pelota azul.

Esto mismo sucede con los eventos, para ejemplificarlo se muestra la siguiente gráfica.



Los eventos pueden dividirse en subtipos mediante diagramas independientes (a) y (b); o bien, es posible expresar la misma información en un diagrama ampliado (c).

(a) El proceso revisar tarea invocar 2 tipos de eventos: tarea aceptada y tarea rechazada. (b) Pero, estos 2 eventos pueden clasificarse en un evento más general que llamaremos tarea revisada (ya que una tarea revisada solo puede ser aceptada o rechazada). (c) Por lo tanto, la operación revisar tarea ahora solo invoca al supertipo de evento tarea revisada, el cual se subdivide en dos subeventos llamados: tarea aceptada y tarea rechazada.

ESQUEMAS JERÁRQUICOS

Dentro de la metodología de Objetos, como se ha mencionado varias veces, debemos pensar en términos de objetos, eventos, procesos y cambios de estados; pero, existen procesos que son demasiado complejos. Por lo cual surge la necesidad de jerarquizar la representación de estos procesos en pequeños subprocesos, es decir, un proceso muy complejo debe ser desglosado en procesos más pequeños y entendibles, los cuales a su vez pueden sufrir otro proceso de subdivisión; así sucesivamente hasta llegar a un punto en el cual el proceso que posea el foco de atención sea tan simple que no posea más divisiones.

Después de lograr simplificar a su mínima expresión el proceso que atraé nuestra atención, habremos logrado a otro concepto: "Aislamiento de la Causa y el Efecto". Este concepto consiste en lograr que los procesos representados en nuestro análisis no dependan de otras cuestiones externas que no sean sus métodos para poder realizar su función, es decir, si deseamos que un proceso sea efectuado únicamente debemos estimular mediante un mensaje al proceso indicado, el cual deberá responder sin importarle quien mande el estimulo o si su resultado generará algún otro efecto.

Lo anterior nos lleva a obtener una *modularidad más clara* la cuál logramos pensando en los términos de una orientación de objetos: objetos, eventos, procesos y cambios de estado. Esto lo decimos basándonos en que si se realiza lo anteriormente descrito, llegaremos a obtener uno de los beneficios más buscados por la tecnología orientada a objetos: "La Reutilización". Indudablemente, nuestros procesos sólo alcanzarán un grado de reutilización cuando puedan ser invocados por cualquier aplicación que desarrollemos y respondan de igual manera de como lo hacen con la operación que los concibió originalmente, y no necesiten de la más mínima modificación.

DISEÑO ORIENTADO A OBJETOS

El análisis orientado a objetos se divide en dos categorías: Análisis de la Estructura de los Objetos y Análisis del Comportamiento de los Objetos. De igual manera, podemos hacer ésta división dentro del diseño orientado a objetos: Diseño de la Estructura del Objeto y Diseño del Comportamiento del Objeto.

Como se menciono anteriormente en este trabajo, el objetivo de la tarea de diseño es, definir las políticas y estandares sobre las cuales fincaremos la implementación de nuestro desarrollo de Software.

En el diseño orientado a objetos, tanto en el de la estructura como en el del comportamiento del objeto, los objetivos que se buscan son los de identificar las siguientes cuestiones:

- ¿ Qué clases se implantarán ?, para lo cuál usaremos como guía los resultados arrojados por el análisis de la estructura de los objetos.

- ¿ Qué estructura de datos utilizará cada clase ?, podemos hacer un diagrama que represente la estructura de datos.

- ¿ Qué operaciones ofrecerá cada clase y cuáles serán sus métodos?, debemos identificar todas y cada una de las operaciones, así como los métodos.

- ¿ Como se implantará la herencia de clases y cómo afectará ésta las especificaciones de los datos y las operaciones ?

- ¿ Cuáles son las variantes ?, debemos identificar los posibles casos especiales bajo los cuales trabajará el objeto y evitar que rompa con la reutilización.

a) Clase

El primer paso en el diseño orientado a objetos es definir las clases de objetos que intervendrán en nuestro desarrollo. Si en el análisis de estructura de objetos ya identificamos los tipos de objetos, en el diseño nos corresponde realizar la implementación de estos tipos de objetos. Definiremos entonces a la clase como la implementación de un tipo de objeto.

La definición de una clase debe especificar la estructura de datos del objeto y las operaciones que se utilizarán para poder accesarlo. La especificación de las operaciones debe ser la manera en como se llevan a cabo, y a esto lo llamaremos *método*. Para que un objeto realice una operación o puedan ser utilizado, se hará exclusivamente por medio de métodos específicos previamente definidos.

Un objeto es la *instancia* de una clase. Los métodos y datos quedan encapsulados en la clase (nos referimos a encapsulamiento cuando hablamos de la capacidad de ocultar éstos al exterior), lo que restringe que sólo las instancias de esta clase son las únicas que podrán hacer uso de los procedimientos en ella encapsulados. Por lo tanto, al ser un objeto la instancia de una clase, éste objeto podrá hacer referencia a sus métodos; con lo cuál se logra que los usuarios observen el "comportamiento" de los objetos en términos de las operaciones que se pueden aplicar a estos. Estas operaciones forman lo que llamaremos la *interfaz* del objeto con los usuarios, es decir la forma en que se comunica el uno con el otro.

b) Operación vs. Método

Puede ser que se haya logrado provocar una especie de confusión entre lo que hemos definido como proceso y lo que hemos definido como método. Para intentar aclarar esto consideremos las siguientes definiciones:

Los *procesos* u operaciones se pueden solicitar como unidades. Los *métodos* son especificaciones de la manera en que se lleva a cabo un proceso dentro de una clase. Es decir, el proceso es el tipo de servicio solicitado (dibuja, crea, etc.), y el método es su código de programación.

c) Herencia de clase

Mencionamos durante el estudio del análisis, que los objetos deben ser conceptualizados con una capacidad de generalización. La generalización se refiere a que el objeto (mejor dicho la instancia de una clase) podrá ser utilizado por múltiples aplicaciones. Y establece que las propiedades de un tipo se aplican a sus subtipos.

La generalización es la base fundamental del concepto de Herencia. Si consideramos que lo que llamamos herencia hace que las estructuras de datos y operaciones sean disponibles para su reutilización por parte de sus subclases.

CAPITULO 6
REPRESENTACIÓN GRÁFICA DEL PROCESO DE ANÁLISIS Y DISEÑO

FIGURAS MAS USUALES

Se ha establecido que la tendencia de la industria del software debe enfocarse hacia el establecimiento de estandares que permitan construir bases para conformar a ésta industria como tal.

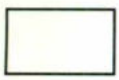
Tal vez uno de los puntos más importantes en la estandarización de algo, es el lenguaje utilizado para describir y entender este algo. Dentro del Desarrollo de software, es regla general expresar el desarrollo del plan de solución a nuestro problema mediante representaciones gráficas llamadas diagramas.

Un diagrama permite plasmar tanto al usuario, al analista y al diseñador la forma de conceptualizar un problema mediante figuras que representan una parte del problema.

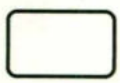
En la actualidad existen diversos tipos de diagramas, todo ocupados para representar distintas actividades de distintas áreas. Lo cual puede provocar que llegue a generar ciertas incompatibilidades en el entendimiento de una situación. Por lo tanto, también es necesario establecer un estandar de diagrama que permita la unificación y fácil transmisión del pensamiento acerca de una misma cosa entre los usuarios, analistas y diseñadores.

Para que un diagrama pueda considerarse estandar debe cumplir con ciertos requisitos como: poseer un mínimo de figuras representativas, el significado de las figuras deben ser lo bastante obvio, proporcionar diagramas de fácil entendimiento y no muy complejos, además de que puedan ser utilizados por diferentes tipos de diagramas.

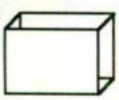
A continuación presentaremos las figuras o símbolos de una de las técnicas de diagrama más usadas en los últimos tiempos dentro de las técnicas involucradas en el desarrollo de software.



Representa datos, tipos de entes, registros, campos, tipos de objetos, etc., por ejemplo alumnos, maestros.



Representa una actividad un proceso ó una operación, por ejemplo aplicar examen, calificar examen.

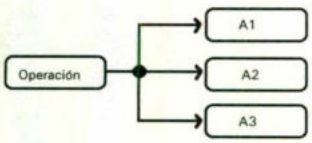


Representa un objeto físico que se mueve de un proceso a otro, por ejemplo boleta de calificaciones.

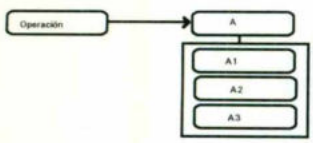
Representa un objeto o tipo de objeto externo.

Representa una actividad, proceso u operación externa.

La linea puede representar conceptos como asociaciones, descomposición, flujo dependencia del tiempo y reglas de activación. (puede contener punta de flecha u otros símbolos dependiendo donde es utilizada)

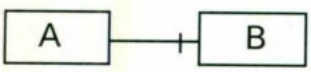


El símbolo ● significa "Exclusividad Mutua", conocida también como un "OR" lógico. Indica que después de realizar operación, se debe realizar una y sólo una de las tres operaciones.

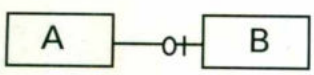


Una caja con divisiones también representa exclusividad mutua, pero a diferencia del círculo relleno, facilita la generación (herencia de clases) y reduce la definición redundante de tipos y operaciones.

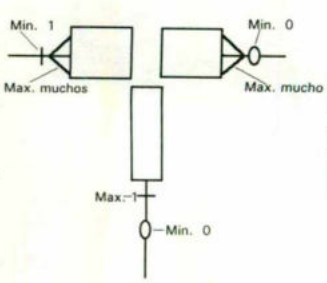
Representa la cardinalidad uno a muchos entre 2 elementos, también se le conoce asociación uno a muchos.



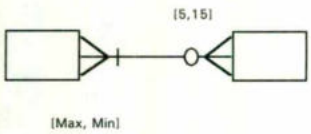
Cardinalidad uno a uno.



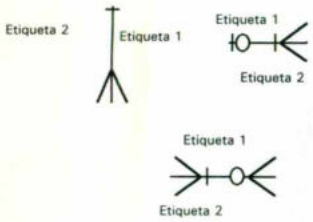
Cardinalidad nula, A se relaciona cero ó una vez con B



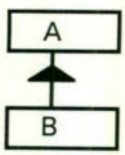
Indica Restricciones de cardinalidad en máximos y mínimos, el máximo se coloca siempre junto a la caja a la que se refiere.



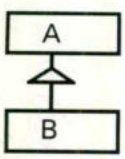
Expresa que el mínimo es 5 y el máximo 15, se utiliza cuando la cardinalidad no puede expresarse en términos de cero, uno o muchos.



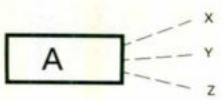
Las lineas etiquetadas expresen la orientación de la lectura de cardinalidad. La etiqueta sobre la linea horizontal da el nombre a la asociación leída de izquierda a derecha. Una etiqueta abajo de una línea horizontal da nombre a la asociación leída de derecha a izquierda. La etiqueta a la derecha de una linea vertical da nombre a la relación leída hacia abajo de la linea, una etiqueta a la izquierda de una linea vertical da nombre a la relación leída hacia arriba de la linea.



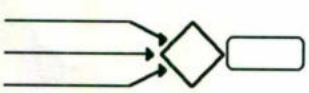
Indica la generación (herencia) de una relación, B es un subtipo de A.



Indica la composición (in tipo particular de relación) B es un componente de A.



Representa la instancias; X, Y y Z son objetos de tipo A

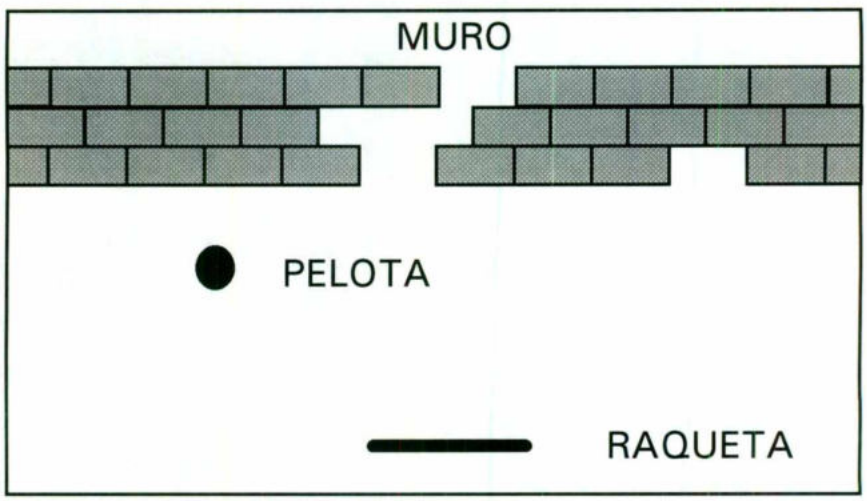


Condiciones de control determina si se da el evento o no.

CAPITULO 7
CASO PRACTICO

En está parte del trabajo intentamos plasmar, mediante un ejercicio, paso a paso el desarrollo de un producto de software mediante la tecnología Orientada a Objetos.

El problema que intentamos solucionar es la construcción de un juego de video que presente la siguiente pantalla.



El objetivo de este juego es eliminar los tabiques que conforman el muro. Para desaparecer un tabique es necesario que la pelota lo golpee. La pelota es impulsada mediante un golpe de la raqueta. El jugador utilizará hasta 3 pelotas (una a la vez) para eliminar los tabiques, se pierde una pelota cuando pasa por debajo de la raqueta, si el jugador logra demoler el muro entonces gana, de lo contrario, pierde.

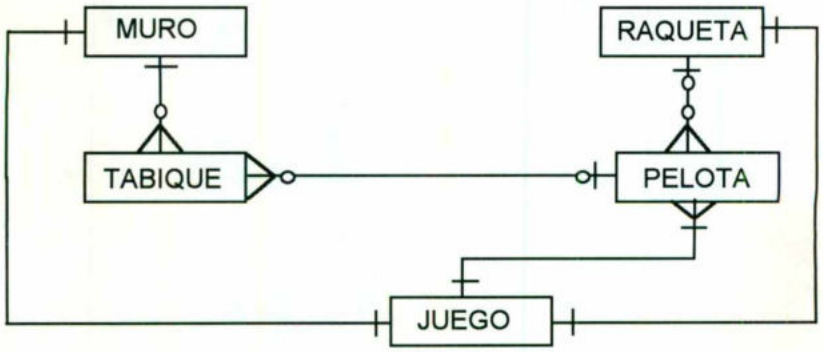
Una vez ya establecido el problema a solucionar con todos sus componentes, el siguiente paso es realizar un análisis, debemos recordar que en la técnica de OO existen dos tipos de análisis; de la estructura de Objetos y del comportamiento de objetos. El primer análisis en realizar es el de la estructura de objetos.

El AEO debe descubrir los tipos de objetos y sus asociaciones, como se organizan en supertipos y tipos, así como la composición de los tipos complejos.

Para nuestro ejemplo contamos con los siguientes tipos de objetos:

- MURO
- TABIQUE
- PELOTA
- RAQUETA
- JUEGO

Después de determinar los tipos de objetos que conforman nuestro entorno, los cuales no nos dicen mucho por si solos, es necesario establecer cuales y como se relacionan.



La gráfica anterior no solamente presenta la relación entre los tipos de objeto, además muestra la cantidad de objetos (tanto de un tipo como de otro) que pueden y deben asociarse. Para una mayor comprensión, consulte la simbología en el capitulo 6. Como una adición extra que ayude a ser más explicativa la gráfica anterior es recomendable dar nombre a las asociaciones, de acuerdo a la simbología del capitulo 6, entre los objetos de los diversos tipos.

MISMA GRAFICA QUE LA ANTERIOR, PERO CON LOS NOMBRES DE LAS ASOCIACIONES

El siguiente paso sería establecer diagramas jerárquicos de los tipos de objetos, pero para nuestro ejemplo esto no es posible.

Es momento de iniciar con la segunda fase del análisis Orientado a Objetos; el análisis del comportamiento de objetos, en este análisis, buscamos definir cuales son los estados que pueden presentar un objeto y sus transiciones, que eventos ocurren entre los objetos y que reglas de activación existen.

Los estados que presentan nuestros objetos son:

JUEGO
* Juego Activo
* Juego Inactivo

MURO
* Muro Construido
* Muro Derruido

TABIQUE
* Tabique Existente
* Tabique Inexistente

PELOTA
* Pelota en Juego
* Pelota en Espera
* Pelota hacia Frente
* Pelota hacia Atrás
* Pelota hacia Derecha
* Pelota hacia izquierda
* Pelota perdida

RAQUETA
* Raqueta hacia Derecha
* Raqueta hacia Izquierda
* Raqueta en Movimiento
* Raqueta Pasiva

En seguida describiremos los tipos de eventos

JUEGO
* Crear Juego
* Iniciar Juego
* Terminar Juego

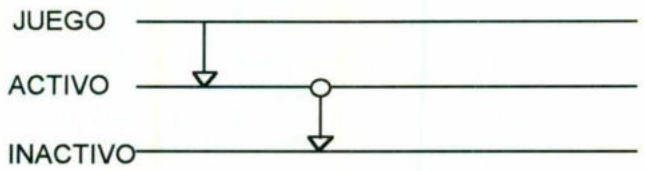
MURO
* Construir Muro
* Derrumbar Muro

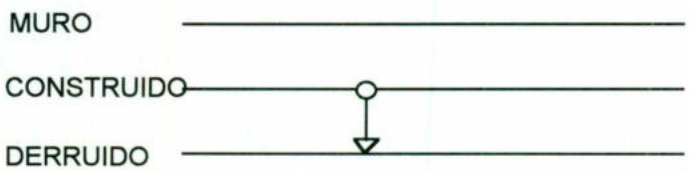
TABIQUE
* Construir Tabique
* Eliminar Tabique

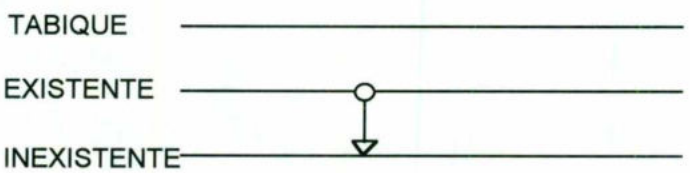
PELOTA
* Crea Pelota
* Mueve Pelota
* Elimina Pelota

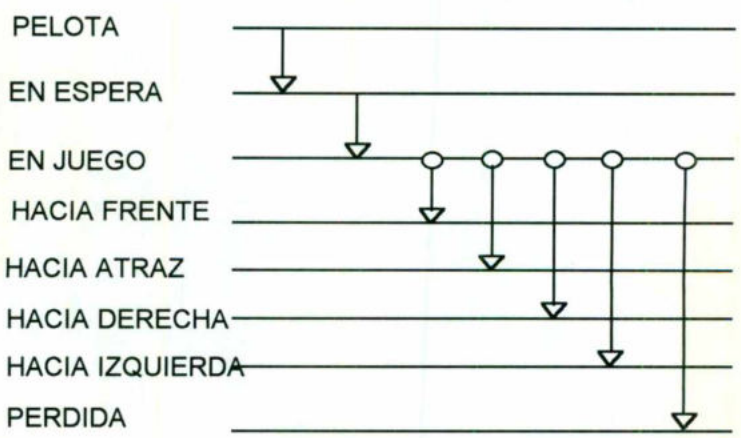
RAQUETA
* Crea Raqueta
* Mueve Raqueta
* Elimina Raqueta

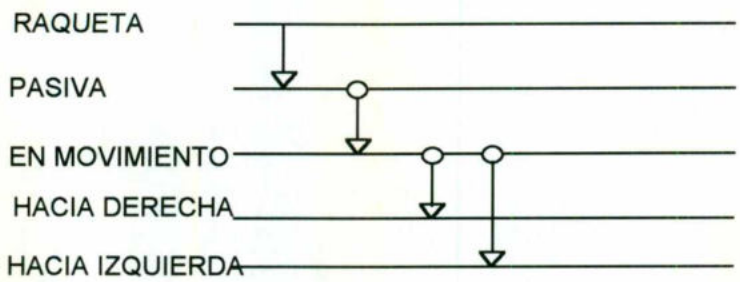
Las siguientes figuras representan los ciclos vitales de los objetos de nuestro ejemplo:



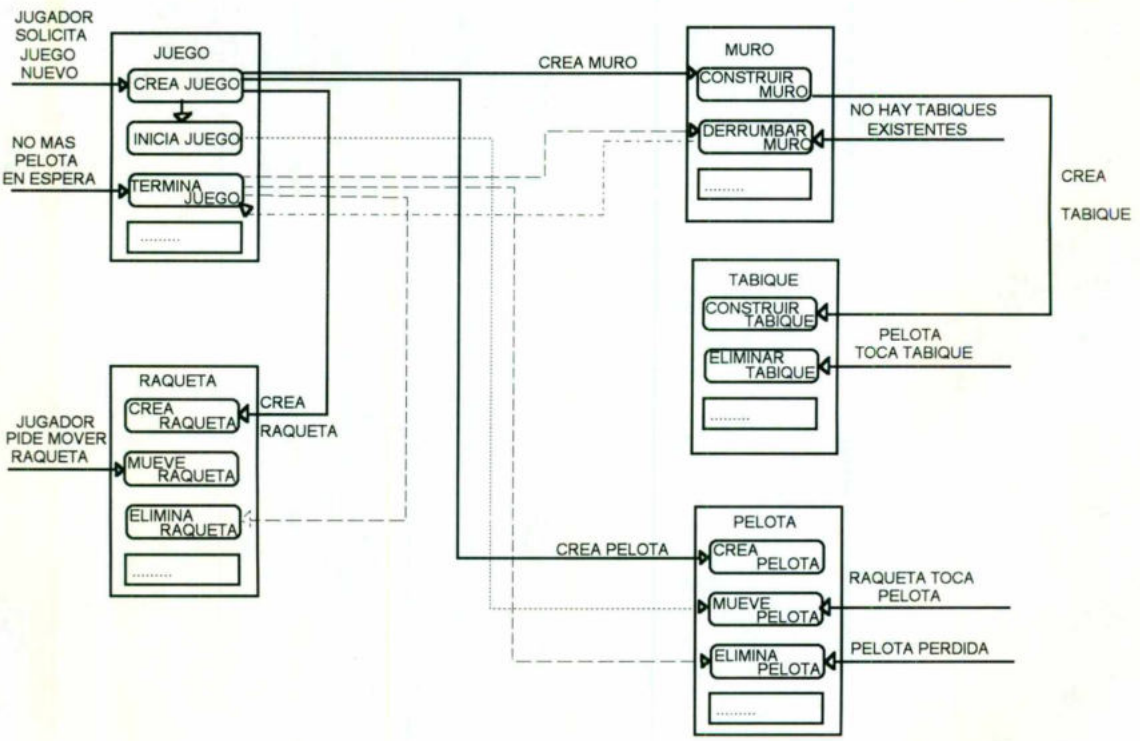








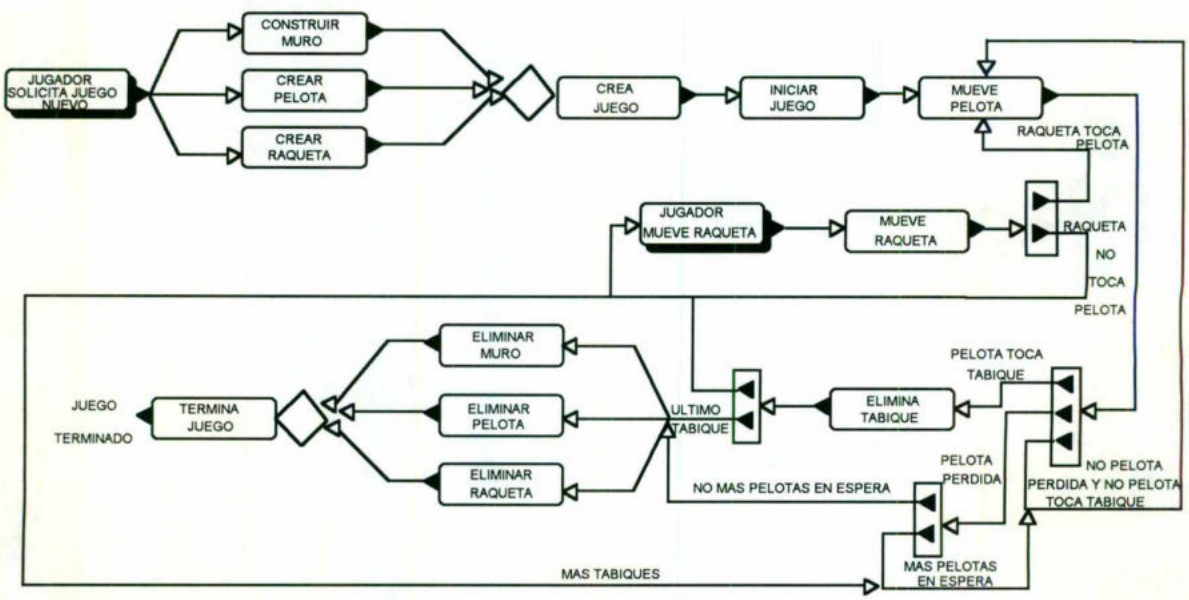
La siguiente figura muestra la relación que existe entre los objetos a través de sus eventos.



ESQUEMAS DE EVENTOS

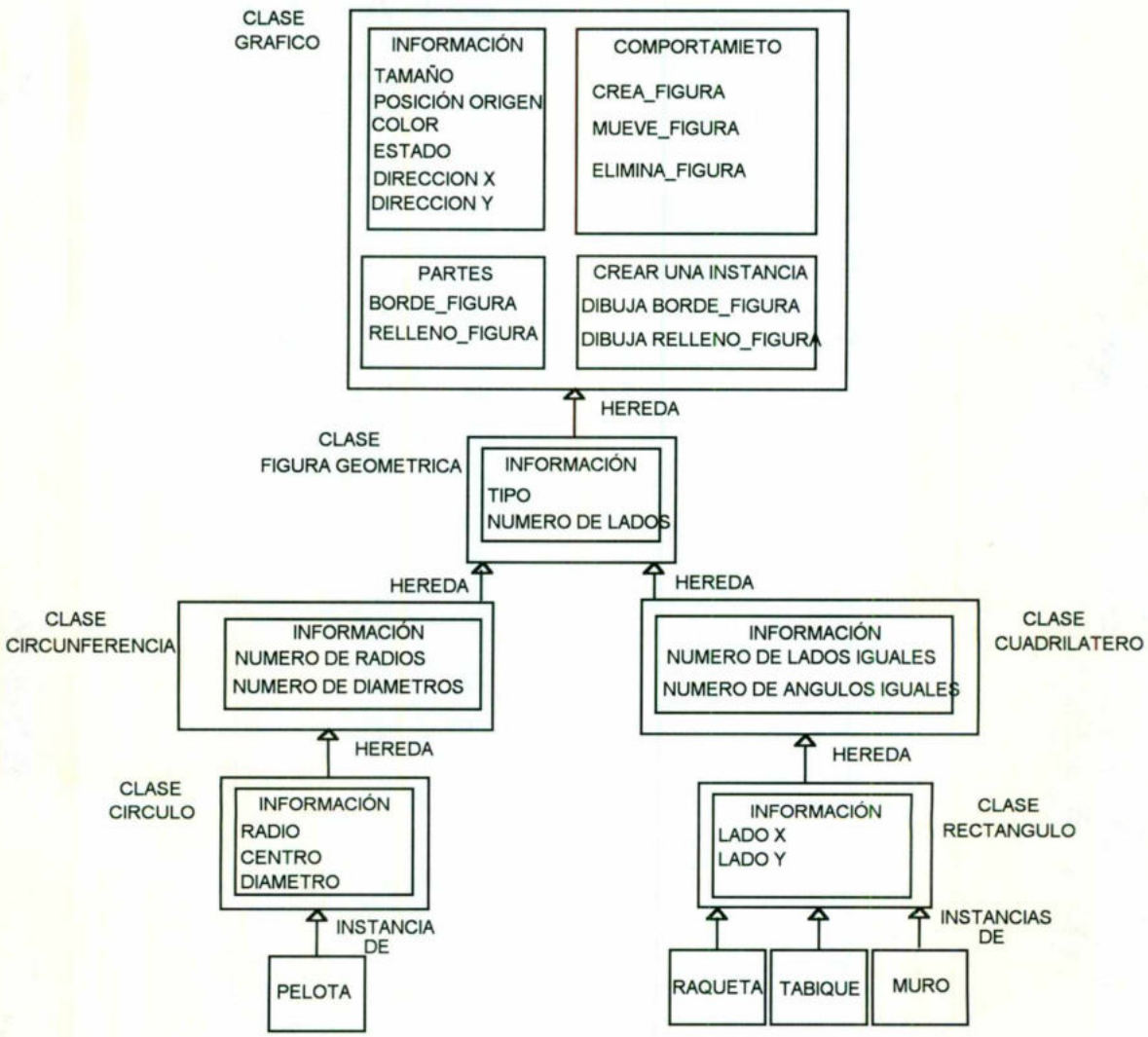
El anterior esquema de eventos es el modelo inicial. Existen otros métodos de diagramación de eventos que aunados a algunas otras actividades generan un esquema mas comprensible que permite a todo el personal que interviene en el proyecto, desde el analista hasta, sus relaciones, condiciones, reglas de activación, estructuras, herencia. eventos y todo lo relacionado a su comportamiento.

Una vez definido e identificado nuestro problema podemos desarrollar otra modalidad de esquema de eventos en el cual se muestren, apoyándonos en los estandares de diagramación, los tipos de eventos (internos y externos), reglas de actuación, condiciones de control, así como los subtipos y supertipos de eventos. Continuando con nuestro ejercicio, en seguida se muestra el siguiente diagrama.

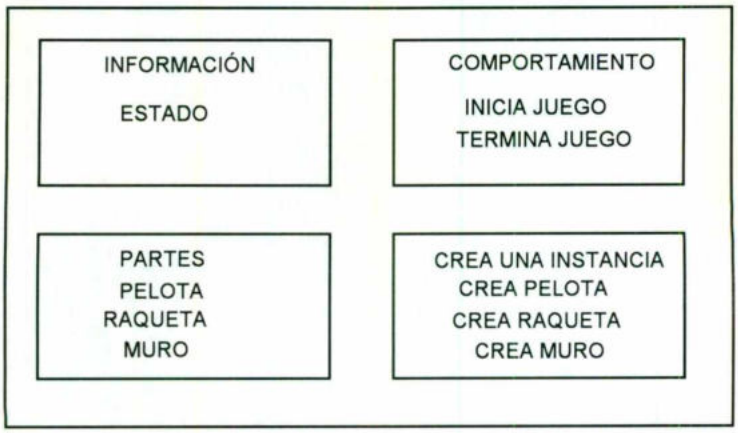


Ahora es momento de iniciar el desarrollo del diseño orientado a objetos, como ya se menciono con anterioridad, el diseño se divide en "Diseño de la Estructura del Objeto" y "Diseño de comportamiento del Objeto" pero por estar íntimamente relacionados con el análisis y con la elección del Lenguaje de Programación Orientado a Objetos (LPOO), se llevan a cabo como si fueran uno mismo.

El primer paso en la etapa de diseño es establecer las clases que se implantarán, para lo cual nos sirve de parámetro la lista de tipos de objetos detectados en el análisis de estructura del objeto, para nuestro ejemplo, las clases a establecer son: Gráfico, Figura geometrica, Circunferencia, Circulo, Cuadrilatero, Rectangulo y Juego.



CLASE JUEGO



La siguiente fase dentro del diseño, es la implementación de os eventos (comportamiento) que controlan cada una de las causas, se debe indicar que la implantación de los eventos en una forma especifica es considerar todos los pormenores del LPOO en el cual se hará la implementación, para salvar esta situación, es recomendable realizar la descripción pseudocodigo y/o un diagrama de métodos de evento.

Como en nuestro ejemplo no hemos tomado un LPOO especifico, se ha decidido realizar únicamente pseudocodigo para identificar los eventos detectados.

```
{ crear_figura}
inicio
   colocase_en_ punto_origen
   dibujar_figura
fin
```

```
{mueve_figura}
inicio
   borrar_figura
   colocarse_punto_origen+incremento_punto_origen
   dibujar_figura
fin


{ eliminar_figura}
inicio
   borrar_figura
fin


{ inicia_juego}
inicio
   crea_figura_pelota
   crea_figura_raqueta
   crea_figura_muro
   crea_figura_pelota
fin


{ termina_juego}
inicio
   elimina_figura_muro
   elimina_figura_pelota
   elimina_figura_raqueta
fin
```

El pseudocodigo y/o diagrama de métodos de eventos puede variar de profundidad y complejidad de acuerdo a la capacidad del analista, diseñador y usuario del desarrollo; así, como del LPOO en el cual se decidió realizar la implementación.

Como de este ejercicio no se contemplo la implementación, hemos llegado al final de nuestro trabajo.

Pero antes se debe mencionar, que lo aquí presentado, significa un gran avance en la solución de nuestro problema, restando únicamente por hacer la programación, la cual puede resultar muy especifica de acuerdo al LPOO elegido. También falta por definir el plan de trabajo para la etapa de mantenimiento, el cual depende del propio LPOO y de las políticas de la organización en la cual se realiza el desarrollo.

CONCLUSIÓN

El uso de una metodología detalada y específica para el desarrollo de software nos ofrece muchos beneficios en el quehacer de nuestro desarrollo.

Al usar una de la muchas metodologías de desarrollo de software que hasta el momento han aparecido, podemos obtener lo siguiente :

- Mejor aprovechamiento de los recursos.
- Menores costos de producción.
- Productos más efectivos y eficientes.
- Productos con menor requerimiento de mantenimiento.

los anteriores son de los principales beneficios que se obtienen del uso de una metodología formal de desarrollo; esto visto de manera general, ya que dentro de cada una de las etapas de desarrollo se obtienen beneficios muy particulares.

Dentro de la gran variedad de metodologías se presenta la problemática de que no se define un estandar de producción, debido a lo cual surgen problemas de incompatibilidad entre las diversas plataformas de desarrollo. Entonces se hace necesario generar un ambiente de desarrollo que nos lleve a lograr ese estandar de desarrollo.

La metodología orientada a objetos se perfila como una tendencia que propicie el ambiente necesario para lograr la estandarización requerida. De manera individual, esta metodología nos brinda una serie de beneficios muy particulares tales como:

- Tener una visión más natural de la realidad.
- Propiciar en mayor grado la reutilización de código.
- Permitir la compatibilidad entre diversas plataformas.

BIBLIOGRAFIA

INGENIERÍA DE SOFTWARE
RICHARD FAIRLEY
MC-GRAW HILL

ANALISIS Y DISEÑO DE SISTEMAS DE INFORMACIÓN
JAMES A. SENN
MC-GRAW HILL

PRINCIPIOS DE SISTEMAS DE INFORMACIÓN
GEORGE M. SCOTT
MC-GRAW HILL

SISTEMAS DE INFORMACIÓN GERENCIAL
2ª EDICIÓN - 1ª EN ESPAÑOL
GORDON B. DAVIS
MARGRETHE H. OLSON
MC-GRAW HILL

ANALISIS Y DISEÑO ORIENTADO A OBJETOS
JAMES MARTIN
JAMES J. ODELL
PRENTICE HALL

C++ AND THE OOP PARADIGM
BINDU R. RAO
MC-GRAW HILL