UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

Sistema para Control de Gimnasios (SCG)

TESIS

QUE PARA OBTENER EL TITULO DE

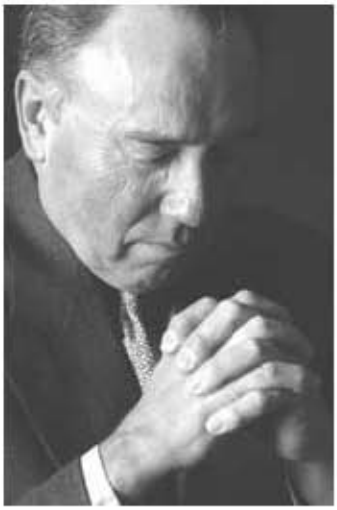
Licenciado en Matemáticas Aplicadas y Computación

PRESENTA

Carlos Alberto Murcia Vargas

Asesor: Lic. Teresa Carrillo Ramírez

Fecha: Julio 2006



Agradecimientos

Cuando bebas agua, recuerda la fuente.

PROVERBIO CHINO

Creo que esta es una de las cosas que mas trabajo me costo escribir, por que no quiero excluir a nadie, aunque sea por error.

Antes que nada quiero agradecer a mis padres, Alberto Murcia Ureña y Georgina Vargas Yañez por darme la vida, por el ejemplo de honradez y tenacidad que siempre me han dado y por no dejarme solo nunca, ya que sin ese apoyo y amor incondicional, no estaría hoy aquí.

A todas y cada una de las personas que intervinieron en la realización de este trabajo, en especial a los instructores y al dueño del gimnasio "Power Body Gym", al Ing. Evaristo Daniel Guillen Silva y demás personas, que contribuyeron para llevar a buen término este proyecto.

A mis hermanos, Edwin y Elizabeth Murcia Vargas que me brindaron su apoyo y me animaron en los momentos en que más lo necesitaba.

A mis abuelos, que aunque algunos ya no están físicamente conmigo los llevo siempre en mi corazón y en mi pensamiento, gracias por sus enseñanzas y consejos.

A mis compañeros y amigos que me dejaron tantos recuerdos y experiencias, algunas alegres y otras no tanto, a lo largo de toda mi vida académica.

A la Universidad Nacional Autónoma de México en especial a la ahora Facultad de Estudios Superiores Acatlán, por darme la oportunidad de obtener una formación profesional en esta gran institución.

A los profesores que a lo largo de todos mis estudios me brindaron los conocimientos y herramientas necesarias para llegar a este punto en el cual me encuentro hoy.

A mi asesora la profesora Teresa Carrillo Ramírez por su paciencia, disponibilidad, apoyo y consejos, a lo largo de todo este proceso de titulación.

Y por último, aunque no por eso menos importante, agradecer a esa entidad superior, que sea cual sea el nombre que se le de (Dios, Cristo, Jehová, Ala, Buda, etcétera). Es nuestro refugio en los momentos más difíciles y sin la cual no existiría el mundo como lo conocemos.

A todos y cada uno de ellos les digo sinceramente ***GRACIAS***.

Carlos Alberto Murcia Vargas

Introducción

Considero más valiente al que conquista sus deseos que al que conquista a sus enemigos, ya que la victoria más dura es la victoria sobre uno mismo.

ARISTÓTELES

Actualmente gozamos de los beneficios de años y años de perfeccionamiento en lo que a programación se refiere. En sus inicios, la programación era con base en tarjetas perforadas las cuales eran incorporadas directamente a la memoria principal de la computadora. Estos programas eran escritos en lenguajes binarios por esta razón eran muy susceptibles a errores, y la falta de una estructura hacia mucho muy difícil el mantenimiento del mismo, puesto que, el código era muy poco comprensible.

Después vinieron los lenguajes de tipo procedural a los cuales también se les conoce como imperativos o de procedimientos. Estos permiten reducir un programa a un mínimo de procedimientos detallados para el procesamiento de los datos. Estos procedimientos definen la estructura general del programa. Siendo las llamadas en secuencia a estos procedimientos las que dan lugar a la ejecución del programa, la cual termina al ejecutarse el último procedimiento en la lista.

A continuación llegaron los lenguajes de tipo modular que como su nombre lo indica trabajan por módulos. Esta programación divide un programa en una gran cantidad de

componentes o módulos. Cada uno de los cuales se compone de datos y procedimientos que manipulan a estos datos. Cuando alguna parte del programa necesita aprovechar la funcionalidad de un módulo, solamente utilizan la interfaz de éste.

El siguiente paso se dio con la programación orientada a objetos. La cual estructura un programa dividiéndolo en objetos. Cada objeto le da forma a algún aspecto del problema que se intenta resolver, estos objetos interactúan entre sí para dar lugar al flujo del programa.

Todos estos tipos o paradigmas de programación tienen un objetivo común que es crear software. Pero ¿que es un software? si se consulta en un diccionario, se puede ver que un software es "*(voz angloamericana). INFORMÁT. Conjunto de programas y rutinas que permite a la computadora la realización de ciertas tareas.*"[DEL03]. Ahora bien si se ve el software como un producto debe tener un proceso de construcción de manera tal, que la ingeniería de software (SE del inglés Software Engineering) es "conjunto de disciplinas usadas para la especificación, diseño e implantación de software para computadoras, siendo su enfoque primario la lógica utilizada en los procesos computarizados".

Con el paso de los años hemos visto que el uso de software se ha diversificado y especializando, por lo que ahora ya podemos ver una computadora en casi cualquier lugar como es el caso de hospitales, oficinas, bancos, centros comerciales, casas, etc. Esto en gran medida causado por las necesidades de información, ya que los lideres, dueños y demás dirigentes de todo tipo de empresas, se han dado cuenta de la importancia que tiene la información en la actualidad, principalmente en la toma de decisiones (estudios de marketing para sacar un nuevo producto al mercado, revisar las ventas realizadas durante un cierto periodo para determinar si se implanta o no una promoción, entre muchas otras cosas).

Dada la gran diversidad de tareas que se pueden realizar, con ayuda del software, como el cálculo de nóminas, el almacenamiento de la información, la generación de reportes y estadísticas. Por esta razón se ha convertido en una herramienta muy importante en los negocios.

De tal manera que cuando me encontraba buscando un tema de titulación, surgió la oportunidad de desarrollar un trabajo, y esto ocurrió en el lugar menos esperado, cuando menos para mí, que fue el gimnasio donde entrenaba. Ya que durante mi estancia en el gimnasio me percate de que el instructor y dueño del gimnasio había adquirido una computadora. Aquí fue donde surgió la idea pues, si ya contaba con una computadora, ¿por qué no desarrollar un software para llevar el control del gimnasio (evaluaciones, pagos, etc.)? Le planteé la idea a mi instructor y no puso objeción, al contrario se mostró muy interesado.

Este trabajo tiene como fin, aplicar la metodología para el desarrollo de sistemas, bajo el paradigma orientado a objetos, específicamente aplicado a los procesos de un gimnasio.

El objetivo de este trabajo es: Aplicar la metodología orientada a objetos para el desarrollo de un sistema que automatice los procesos de un gimnasio. De donde se desprende la siguiente hipótesis: "Al elaborar un sistema empleando la metodología orientada a objetos se reduce el tiempo empleado en la construcción del mismo y se produce un sistema eficiente, en este caso, para el control de los procesos de un gimnasio (evaluaciones a los atletas, determinación del somatotipo y, elaboración y asignación de dietas), mejorando con esto la calidad de los servicios que brinda el gimnasio Power Body Gym".

En el capítulo I, se presentan de forma concreta los conceptos de Ingeniería de Software y de la Metodología Orientada a Objetos (MOO). Para posteriormente presentar el planteamiento del problema para el caso especifico del Sistema para Control de Gimnasios (SCG).

En el capítulo II, se da una breve descripción de las distintas metodologías orientadas a objetos para el desarrollo de sistemas, para posteriormente enfocarse en la explicación de la metodología para análisis de sistemas orientados a objetos propuesta por Coad y Yourdon. La cual se utilizó para el desarrollo del Sistema de Control de Gimnasios (SCG).

En el capítulo III, se expone la metodología, en está parte, para el diseño de sistemas orientados a objetos, propuesta por Coad y Yourdon sus conceptos y generalidades para concluir con el diseño específico para el SCG.

Finalmente, en el capítulo IV, se describe la manera en que se construyeron todos los objetos descritos en los capítulos anteriores. También se explica como se construyeron los listados, con ayuda de la herramienta denominada Crystal Reports. Además se presentan fragmentos de código fuente de algunas de las clases desarrolladas, así como las pantallas que muestran los distintos pasos para la creación de un listado. También se incluyen algunas de las pantallas del sistema ya terminado.

Cabe mencionar que durante el desarrollo de este software se utilizaron las siguientes herramientas:

Para crear la base de datos del sistema se utilizo ACCESS®, para realizar los diagramas entidad relación de esta base de datos se utilizo ERWIN 2000®, para los diagramas de Capas se utilizo el programa denominado VISIO 2000®, para la programación se utilizo Visual C++ ® y como se menciono anteriormente se utilizo Crystal Reports® para la creación de algunos listados. Más adelante se explica el por qué de la elección de estas herramientas.

Por último vienen las conclusiones a las cuales se llego con respecto al objetivo e hipótesis planteados y se incluyen algunas observaciones en cuanto al uso del paradigma orientado a objetos. Así como, lo que representaron las experiencias adquiridas en la realización de este sistema.

Además se incluye un Glosario y las Referencias Bibliográficas de las obras y publicaciones consultadas durante la realización del presente trabajo.

Capítulo I El software como una herramienta para la administración de un gimnasio

Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber.

Albert Einstein

I.1 Principios de Ingeniería de Software

Ingeniería de Software

Antes de presentar íntegramente el análisis del problema, se darán los conceptos básicos de ingeniería de software, así como aquellos conceptos más específicos que se emplearán en el desarrollo del sistema.

El término Ingeniería es usado en la descripción de metodologías modernas para explicar que se usan disciplinas formadas con técnicas precisas y bien pensadas, antes que inventadas sobre la marcha.

El concepto de Ingeniería de Software se refiere al conjunto de disciplinas usadas para la especificación, diseño e implantación de software para computadoras, siendo su enfoque primario la lógica utilizada en los procesos computarizados. La metodología de Ingeniería de Software se llega a formalizar a finales de los 70's e involucra técnicas para desarrollo de

software tales como la programación estructurada, análisis y diseño, tanto estructurado, como orientado a objetos, además de herramientas para dar soporte a las mismas.

Definición de sistema

Tomando la definición de Roger S. Pressman [1] se tiene que un sistema es: *Un conjunto o arreglo de elementos que están organizados para realizar un objetivo predefinido procesando información.* Donde este objetivo puede ser muy variado desde cosas simples como entretener y enseñar a un niño (sistemas educativos), hasta cosas tan complejas como controlar un robot explorador (sistemas expertos). En general, un sistema esta compuesto de una entrada, un proceso y una salida:

Las entradas pueden ser las peticiones del usuario y la información con que se cuenta.

El proceso son las operaciones internas que debe realizar la computadora para satisfacer la petición realizada.

Y la salida son los resultados obtenidos después de ese proceso.

Los objetivos generales al desarrollar un sistema son:

- Funcionalidad: Que el software haga el trabajo para el que fue creado.
- Confiabilidad: Que lo haga bien.
- Disponibilidad: Que todos los que quieran utilizar el sistema lo puedan hacer sin problemas.
- Seguridad: La persona que no esté autorizada no debe tener acceso al sistema o a parte de él.
- Integridad: Que la información sea verídica e igual en todos lados.
- Estandarización: Las características de la interfaz de usuario deben ser comunes entre múltiples aplicaciones.
- Integración: Que todos lo módulos sean fáciles de acceder.
- Consistencia: Que el apoyo visual sea igual en todas las pantallas. También se refiere a la terminología y los comandos usados en la interfaz.
- Portabilidad: Que el paquete sea reconocido por la mayoría de las computadoras.

---

[1] Roger S. Pressman Ingeniería del Software Un Enfoque Practico, Mc Graw Hill, 4ª edición, México 2000.

- Calendarización: Respetar fechas límites para la culminación del proyecto.
- Presupuesto: No rebasar el presupuesto acordado, considerando que el 70% del costo del software se gasta en el mantenimiento.

Otros elementos a tomar en cuenta para el desarrollo de un sistema son los que tienen que ver con los

Factores Humanos:

- Velocidad de Aprendizaje.- Se pretende que la persona aprenda a usar el sistema lo más pronto posible.
- Velocidad de Respuesta.- El tiempo necesario para realizar una operación en el sistema.
- Tasa de errores.- Porcentaje de errores que comete el usuario.
- Retención.- Cuánto recuerda el usuario sobre el uso del sistema en un período. de tiempo.
- Satisfacción.- Se refiere a que el usuario esté conforme con el sistema.

Factores de Adecuación

- Características Físicas.- Cada persona tiene diferentes características físicas. Hay algunas personas que no les gustan los teclados mientras que a otras sí. Es por eso que hay teclados ergonómicos. Lo mismo sucede con el Mouse.
- Ambiente.- Cada interfaz tiene que adecuarse al lugar donde va a ser usado el sistema.
- Visibilidad.- Tomar en cuenta la cantidad de iluminación del lugar. ¿Se refleja el brillo en la pantalla?
- Cultura.- Este factor es importante si el mercado para el sistema es a nivel internacional. La cultura de la sociedad en la que se empleara el sistema determina factores importantes para el desarrollo del mismo. Algunos de estos factores pueden ser tipos de ventanas, cubículos, oficinas, sillas, etc. Que comúnmente se utilizan en esa sociedad.

Antes de iniciar con el sistema, es necesario determinar qué paradigma de programación se va a emplear. Por esta razón a continuación se mencionan los principales paradigmas de programación con algunas de sus principales características:

Estructurado: es una técnica en la cual la estructura de un programa, esto es, la interpelación de sus partes se realiza tan claramente como es posible mediante el uso de tres estructuras lógicas de control que son: la secuencia, la selección y la interacción. Estas tres estructuras lógicas de control pueden ser combinadas para producir programas que manejen cualquier tarea de procesamiento de información.

- Un programa estructurado esta compuesto de segmentos, los cuales puedan estar constituidos por unas pocas instrucciones o por una página o más de codificación.

- Cada segmento tiene solamente una entrada y una salida, estos segmentos, asumiendo que no poseen lazos infinitos y no tienen instrucciones que jamás se ejecuten, se denominan programas propios. Cuando varios programas propios se combinan utilizando las tres estructuras básicas de control mencionadas anteriormente, el resultado es también un programa propio.
- Una característica importante en un programa estructurado es que puede ser leído en secuencia, desde el comienzo hasta el final sin perder la continuidad de la tarea que cumple el programa, lo contrario de lo que ocurre con otros estilos de programación.

Lógico: se basa en un subconjunto del cálculo de predicados, en instrucciones escritas en formas conocidas como cláusulas de Horn.

- Estos programas están constituidos por expresiones lógicas, es decir que son Falsas o Verdaderas, con base a una pregunta o a órdenes.
- El orden de ejecución de las instrucciones no es en forma secuencial.
- No hay instrucciones de control propiamente dichas, ya que siguen las "reglas o cláusulas de Horn". Es decir una instrucción se ejecuta en el momento de cumplirse ciertas condiciones. Además de estas reglas también se definen "factors" en los cuales se aplicaran las reglas.

Funcional: Como su nombre lo indica trabajan a través de funciones. Entendiendo estas no como subprogramas, sino como, funciones matemáticas puras.

- Cada función devuelve un solo valor, dada una lista de parámetros.
- No existe la asignación de variables.
- La falta de construcciones estructuradas como la secuencia o la iteración lo que obliga al uso de la recursividad cuando es necesario repetir una instrucción.
- El orden de ejecución de las instrucciones es en forma secuencial.

Orientado a Objetos: este paradigma consiste en descomponer el problema a resolver en objetos, los cuales tienen atributos. Entre sus principales características se encuentran:

- Los objetos con características similares pueden agruparse en categorías llamadas *clases.*
- Estos objetos interactúan a través de *mensajes* que representan la comunicación o interacción de elementos de la realidad.
- La *herencia*, que es el hecho de crear clases a partir de otras ya creadas, donde la clase original a la que se le llama *clase base* le hereda sus atributos y comportamientos a una clase derivada denominada *clase hija.*
- Separación entre lo *que* debe hacer el objeto (responsabilidad asignada a este objeto) y *cómo* lo va a realizar (implementación) a esto se le llama *encapsulación.*

- Por ultimo esta el polimorfismo que no es otra cosa que permitir al objeto despreocuparse de quien va a realizar la tarea sólo le importa que sea el responsable de llevar a cabo el trabajo encomendado.

Para elaborar el sistema que me emplea en este trabajo se utilizará el paradigma Orientado a Objetos OO, por los siguientes motivos:

- Al permitir la reutilización de código se hace más rápido el proceso de análisis diseño e implementación.
- Genera sistemas estandarizados en su presentación.
- Al contar con librerías de funciones que ya se han probado, se ahorra tiempo, esfuerzo y se disminuyen los errores.

A estas se pueden agregar las siguientes:

- Es una metodología que ha adquirido amplia popularidad.
- Actualmente el que suscribe utiliza este tipo de programación en la empresa en que labora.
- Puede servir como una aportación a los estudiantes que actualmente estudian Matemáticas Aplicadas y Computación.

A continuación se dan algunos conceptos que son esenciales para comprender lo que es la visión OO.

I.2 Conceptos y principios orientados a objetos

El enfoque orientado a objetos para el desarrollo de software se propuso por primera vez a finales de los años 60. Sin embargo, necesito casi veinte años para llegar a ser ampliamente usado. Este enfoque lleva a la reutilización de componentes de software lo cual redunda en un desarrollo de software más rápido y programas de mejor calidad.

El software orientado a objetos es más fácil de mantener debido a que su estructura es inherentemente descompuesta. La notación de diseño OO combina aspectos tanto de los diagramas de entidad – relación como de flujo de datos. Además, las especificaciones de objetos pueden tomar como punto de partida diagramas de flujo, lenguaje estructurado, tablas de decisión o cualquiera de las notaciones de especificación de proceso que se hayan aprendido anteriormente.

I.2.1 El marco de proceso común OO

Un marco de proceso común (MPC) define un enfoque organizativo para el desarrollo y mantenimiento de software, por consiguiente identifica el paradigma de ingeniería de software aplicado para construir y mantener el software, así como las tareas, hitos y entregas requeridos. También establece el grado de rigor con el cual se enfocarán los diferentes tipos de proyectos. Siempre es adaptable de tal manera que siempre cumplirá con las necesidades individuales, del equipo (hardware) o del proyecto, siendo esta su característica más importante. Como el software OO evoluciona a través de un número de ciclos, el MPC debe ser evolutivo.

Ed Berard [BER93] y Grady Booch [BOO00], entre otros autores, sugieren el uso de un "modelo recursivo/paralelo" para el desarrollo de software orientado a objetos. El cual funciona de la siguiente manera:

- Realizar los análisis suficientes para aislar las clases de problemas y las conexiones más importantes.
- Realizar un pequeño diseño para determinar si las clases y conexiones pueden ser implementadas de manera practica.
- Extraer objetos reutilizables de una biblioteca para construir un prototipo previo.
- Conducir algunas pruebas para descubrir errores en el prototipo.
- Obtener retroalimentación del cliente sobre el prototipo.
- Modificar el modelo de análisis basándose en lo que se ha aprendido del prototipo, de la realización del diseño y de la retroalimentación obtenida del cliente.
- Refinar el diseño para acomodar sus cambios.
- Construir objetos especiales (no disponibles en la biblioteca).
- Ensamblar un nuevo prototipo usando objetos de la biblioteca y los objetos que se crearon nuevos.
- Realizar pruebas para descubrir errores en el prototipo.
- Obtener retroalimentación del cliente sobre el prototipo.

Este enfoque continúa hasta que el prototipo evoluciona hacia una aplicación productiva. Berard [BER93] describe el modelo de la siguiente manera:

- Descomponer sistemáticamente el problema en componentes altamente independientes.
- Aplicar de nuevo el proceso de descomposición a cada una de las componentes independientes para, a su vez, descomponerlas (la parte recursiva.).
- Conducir este proceso de reaplicar la descomposición de forma concurrente sobre todas las componentes (la parte paralela).
- Continuar este proceso hasta cumplir los criterios de compleción.

Para controlar el marco de proceso recursivo/paralelo, el director del proyecto tiene que reconocer que el progreso esta planificado y medido de manera incremental. Esto es, las tareas y la planificación del proyecto están unidas a cada una de las "componentes altamente independientes", y el progreso se mide para cada una de estas componentes individualmente.

Cada iteración del proceso recursivo/paralelo requiere planificación, ingeniería (análisis, diseño, extracción de clases, creación de prototipos y pruebas) y actividades de evaluación. Al continuar el trabajo de ingeniería, se producen versiones incrementales del software. Durante la evaluación se realizan, para cada incremento, revisiones evaluaciones del cliente y pruebas, las cuales producen una retroalimentación que afecta a la siguiente actividad de planificación y el subsiguiente incremento. En la figura I.1 se muestra el proceso antes descrito.

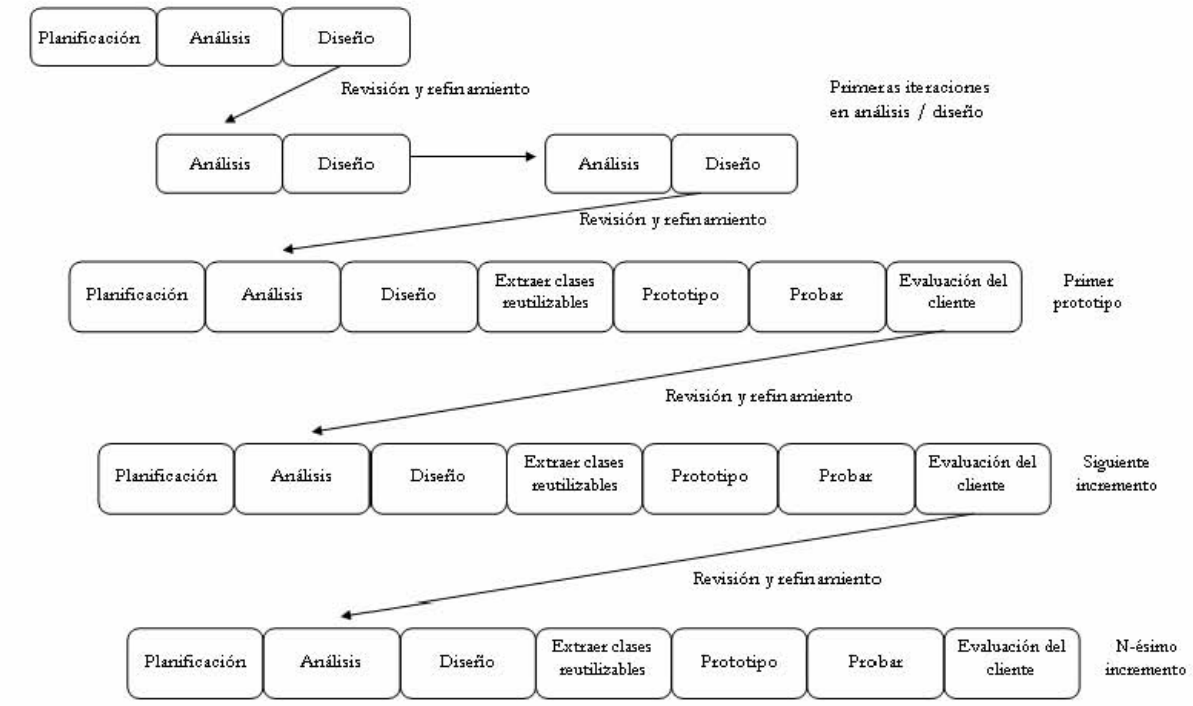


Figura I.1 Proceso recursivo paralelo.

I.2.2 El paradigma orientado a objetos

Desde comienzos de la década de los 80, el paradigma "orientado a objetos" ha ido madurando como un enfoque de desarrollo de software alternativo a la programación estructurada o modular. Se empezaron a crear diseños de aplicaciones de todo tipo usando una forma de pensar orientada a los objetos, y a implementar estos diseños utilizando lenguajes orientados a objetos. Sin embargo, el análisis de requisitos se quedó atrás. No se desarrollaron técnicas de análisis específicamente orientadas a objetos.

Esta situación ha ido cambiando poco a poco, a medida que se desarrollaban técnicas de análisis específicas para desarrollar software orientado a objetos e incluso como complemento de otros métodos de análisis. Ejemplos de estas nuevas técnicas son los métodos de Coad/Yourdon [COA91], Jacobson [JAC92], Booch [BOO91] y Rumbaugh (OMT) [RUM91].

El Análisis Orientado a Objetos (AOO) se basa en conceptos sencillos, conocidos desde la infancia y que aplicamos continuamente: objetos y atributos, el todo y las partes, clases y miembros. Puede parecer llamativo que se haya tardado tanto tiempo en aplicar estos conceptos al desarrollo de software. Posiblemente, una de las razones es el éxito de los métodos de análisis estructurados, basados en el concepto de flujo de información, que monopolizaron el análisis de sistemas y software durante los últimos veinte años.

En cualquier caso, el paradigma orientado a objetos ha sufrido una evolución similar al paradigma de programación estructurada: primero se empezaron a utilizar los lenguajes de programación estructurados, que permiten la descomposición modular de los programas; esto condujo a la adopción de técnicas de diseño estructuradas y de ahí se pasó al análisis estructurado. El paradigma orientado a objetos ha seguido el mismo camino: el uso de la Programación Orientada a Objetos (POO) ha modificado las técnicas de diseño para adaptarlas a los nuevos lenguajes y ahora se están empezando a utilizar técnicas de análisis basadas en esta nueva forma de desarrollar software.

I.2.3 Conceptos básicos

Las técnicas orientadas a objetos se basan en organizar el software como una colección de objetos discretos que incorporan tanto estructuras de datos como comportamiento. Esto contrasta con la programación convencional, en la que las estructuras de datos y el comportamiento estaban escasamente relacionadas.

Las seis ideas básicas que caracterizan el enfoque orientado a objetos son: (1) objetos, (2) clases, (3) mensajes, (4) encapsulación (5) herencia y (6) polimorfismo.

1. Objeto. Un objeto es una representación en computadora de alguna cosa o evento del mundo real: hardware, software, documentos, seres humanos etc. Los objetos pueden tener atributos y comportamientos.

2. Clases. Una clase es una categoría de objetos similares. Los objetos se agrupan en clases. Una clase define el conjunto de atributos y comportamientos compartidos que se encuentran en cada objeto de la clase. El programador debe definir las clases en el programa. Cuando el programa se ejecuta se pueden crear objetos a partir de las clases establecidas. Se usa el termino “ocurrencia” cuando se crea un objeto de una clase.

3. Mensajes. Se puede enviar información de un objeto a otro. Estos mensajes no son de forma libre en ningún sentido, ya que la clase emisora ha sido programada para transmitir un mensaje bajo determinadas circunstancias. La clase receptora ha sido programada para reaccionar al mensaje en alguna forma.

4. Encapsulación. Por lo regular, la información acerca de un objeto esta encapsulada por su comportamiento. Esto significa que un objeto mantiene datos acerca de cosas del mundo real a las que representa en un sentido verdadero. Típicamente, a un objeto se le debe "pedir" o "decir" que cambie sus propios datos con un mensaje, en vez de esperar que tales datos de procesos externos cambien la naturaleza de un objeto. Puede parecer trivial el que un atributo de un objeto sea cambiado alterando directamente sus datos o enviando un mensaje al objeto para que active el comportamiento interno que cambia ese dato. Pero esta diferencia es una característica extremadamente importante de los programas Orientados a Objetos. Los datos encapsulados pueden ser protegidos en tal forma que solamente el objeto mismo puede hacer tales cambios por medio de su propio comportamiento. Esto facilita la construcción de objetos que son muy confiables y consistentes, debido a que ellos tienen control completo sobre sus propios atributos. También hace que sea mucho mas fácil el mantenimiento y cambio de programas.

5. Herencia. Las clases pueden tener "hijos", esto es, una clase puede ser creada a partir de otra clase. La clase original, o madre, es llamada "clase base". La clase hija es llamada "clase derivada". Una clase derivada puede ser creada en forma tal que herede todos los atributos y comportamientos de la clase base. Una clase derivada puede tener también atributos y comportamientos adicionales. La herencia reduce la labor de programación reutilizando fácilmente objetos antiguos. Algunos lenguajes de programación Orientada a Objetos proporcionan herencia múltiple. En estos casos especiales se puede crear una clase derivada en forma tal que herede todos los atributos y comportamientos de más de una clase base.

6. Polimorfismo. Este término se refiere a comportamientos alternos entre clases derivadas relacionadas. Cuando varias clases heredan atributos y comportamientos, puede haber casos donde el comportamiento de una clase derivada deba ser diferente del de su clase base o de sus clases derivadas parientes. Esto significa que un mensaje puede tener diferentes efectos, dependiendo de exactamente qué clase de objeto recibe el mensaje.

I.3 Metodología OO para el desarrollo de software

I.3.1 Definición del problema

La recopilación de requisitos y datos es siempre la primera actividad a realizar en el desarrollo de cualquier aplicación. Al igual que en el análisis y diseño estructurado, la recopilación de requisitos y datos puede ser un rápido encuentro en el cual el cliente y el desarrollador acuerdan definir los requisitos básicos del sistema (Hardware, Software).

Para esto se puede hacer uso de algunas técnicas usadas en el enfoque estructurado tales como:

Identificación de problemas, oportunidades y objetivos:

Esta fase es crucial para el éxito del resto del proyecto, pues nadie estará dispuesto a desperdiciar su tiempo dedicándolo al problema equivocado. La primera etapa requiere que el analista observe de forma objetiva, clara y honesta lo que ocurre en la empresa. Luego, en conjunto con los otros miembros de la organización hará resaltar los problemas de ésta. Muchas veces éstos ya fueron identificados previamente; y por esta razón, es que se llama al analista.

Las oportunidades son aquellas situaciones que el analista considera que pueden perfeccionarse mediante el uso de los sistemas de información computarizados. Al aprovechar las oportunidades, la empresa puede lograr una ventaja contra sus competidores o en ciertos casos llegar a establecer un estándar industrial.

La identificación de los objetivos es un componente importante. En primera instancia, el analista deberá descubrir lo que la empresa intenta realizar. Y luego, estará en posibilidad de determinar si el uso de los sistemas de información apoyaría a la empresa para alcanzar sus metas, al encaminarla a problemas u oportunidades específicas.

Determinación de los requerimientos de información:

Para identificar los requerimientos de información dentro de la empresa, pueden utilizarse diversos instrumentos, los cuales incluyen: el muestreo, el estudio de los datos y

formas usadas por la organización, la entrevista, los cuestionarios, la observación de la conducta de quien toma las decisiones, así como de su ambiente y también el desarrollo de prototipos.

En esta etapa se hace todo lo posible por identificar qué información requiere el usuario para desempeñar sus tareas. Los métodos para establecer las necesidades de información, obligan al analista a relacionarse directamente con los usuarios. Esta etapa sirve para elaborar la imagen que se tiene de la organización y de sus objetivos. En ocasiones, se llegan a concluir sólo las dos primeras etapas.

Por lo regular las personas involucradas en esta fase son los analistas y los usuarios, regularmente los administradores y trabajadores de las operaciones. Aquí se debe conocer a detalle las funciones actuales del sistema: el quién, qué, donde, cuándo y cómo del negocio bajo estudio.

A continuación se presenta una descripción de la manera en la que se recaba la información mediante la observación, la búsqueda de documentos y la entrevista [2] ya que son los medios que se emplearán para el caso práctico.

Observación

Esta metodología esta basada en la observación visual y práctica de las funciones y métodos de los diversos departamentos de la organización de acuerdo a los requerimientos.

Teniendo dos clases de actividades o procedimientos

Los Repetitivos: Donde el sistema se encuentra en una base cíclica y es donde se concentra el 90 % de la acción de la organización.

Los específicos: Que incluyen el trabajo no rutinario, las excepciones, las decisiones, la investigación, los estudios y las habilidades propias de los ejecutivos.

Esta última no se presta fácilmente al análisis y planeación de los sistemas, donde los ciclos y las rutinas rinden sus frutos.

En esta etapa de observación se tiene que ver de cerca todas las rutinas de la empresa, documentarlas y sobre todo confirmarlas para saber que se esta en lo cierto.

---

[2] http://jorgevilar.tripod.com.mx/busqueda.htm

Búsqueda de Documentos

Esta fase tiene como objetivo, encontrar si en la corporación ya existen procedimientos escritos sobre las diferentes funciones o rutinas que tiene el personal, también es la búsqueda de políticas escritas o tácitas que tiene el personal y la corporación en su ética y actuación.

Al hacer esta búsqueda se pueden encontrar tres vertientes.

1. Existen procedimientos pero no son lo que realmente sé esta haciendo. Se llega a esta conclusión llevando a cabo una observación. Si es así se desechan y se escriben de nuevo, teniendo cuidado en lo que son políticas. Para esto se busca apoyo y para saber, por qué no se están llevando a cabo.
2. No existen, todo es de manera informal y platicada. Hay que escribirlos de acuerdo a lo que se observo, y pedir apoyo para ver sí se esta en lo correcto.
3. Existen y son altamente respetados por el personal. Este es el que deberían tener todas las empresas. Hay que estudiarlos e integrar los nuevos requerimientos que se pretendan.

Hay que recordar que los procedimientos y políticas son la base. Y que ellos son el gran comienzo para el éxito. Estos deben ser escritos, publicados, capacitados, comprendidos, acordados y respetados.

Entrevistas

Primeramente se debe planear la entrevista antes de ir a ella, con las siguientes vertientes.

- Debe conocer la premisa del requerimiento, con ello se pueden emplear otras metodologías (búsqueda de documentos, observación, etc.). para no llegar a la entrevista sin un conocimiento previo aunque no sea profundo.
- Con lo anterior se debe elegir qué información o cuestionarios, se necesitan obtener como resultado final de la entrevista.
- Si falta información o se tienen dudas, hay que preguntarse primero si debería el entrevistado tener esa información, y si la tiene que esta pueda estar disponible en la entrevista. Al conocer lo anterior se puede fijar un tiempo para la obtención de dicha información.
- Se debe fijar un tiempo razonable para la entrevista.
- Hacer un plan tácito de la información que se puede obtener según el tiempo planeado.
- Determinar como se debe pedir la información.

- Qué se debe hacer para establecer armonía, siendo este papel muy importante para ganar la confianza del entrevistado.
- Qué otras personas, se deben de entrevistar para el cruce de información, y hacer más fuerte el requerimiento.
- Qué pasa si la información no la ofrecen fácilmente. Se debe pensar en un plan para poder tenerla sin resistencias.

Considerando los puntos anteriores:

- Qué se necesita hacer o tener antes de la entrevista.
- Cómo romperé el "hielo" en la entrevista.
- Cómo controlaré la entrevista para que no se salga de control (estrategias de control).
- Cómo terminaré la entrevista.
- Cómo haré para continuar las relaciones interpersonales establecidas para mis siguientes entrevistas.

Además hay que estar alerta y considerar lo siguiente:

- Evitar la discusión argumentativa.
- Evitar palabras como: problemas...reducción...etc.
- Elaborar preguntas para obtener mas... de un si o un no.
- Dejar siempre las puertas abiertas para regresar.
- Controlar el tiempo de la entrevista sin divagar.

Y recordar que la entrevista no es una receta de cocina. Mucho dependerá de lo que se quiera obtener de información, de que entienda completamente el requerimiento y que recuerde que el profesional en el tema es el entrevistado, no el analista.

I.3.2 Planificación

La planificación para proyectos orientados a objetos es complicada por la naturaleza iterativa del marco de trabajo del proceso. Para esto Lorenz y Kidd sugieren un conjunto de métricas que pueden ayudar durante esta planificación del proyecto, las cuales se exponen a continuación:

Número de Iteraciones Principales

El proceso OO se mueve a través de una espiral evolutiva que comienza con la comunicación con el usuario. Es aquí donde se define el dominio del problema y se identifican las clases básicas del problema. El trabajo técnico asociado con la ingeniería de software OO sigue el camino iterativo mostrado en la caja sombreada de la imagen I.2. La ingeniería de

Software Orientada a Objetos hace hincapié en la reutilización. Por lo tanto, las clases se buscan en una biblioteca (de clases OO existentes) antes de construirse. Cuando una clase no puede encontrarse en la biblioteca, el desarrollador de software aplica análisis, diseño, programación y pruebas orientadas a objetos para crear la clase y los objetos derivados de la clase. La nueva clase se pone en la biblioteca de tal manera que pueda ser reutilizada en el futuro.

La visión orientada a objetos demanda un enfoque evolutivo de la ingeniería del software. Como es excesivamente difícil definir las clases necesarias para un gran sistema o producto en una sola iteración. A medida que el análisis OO y los modelos de diseño evolucionan, se vuelve aparente la necesidad de clases adicionales.

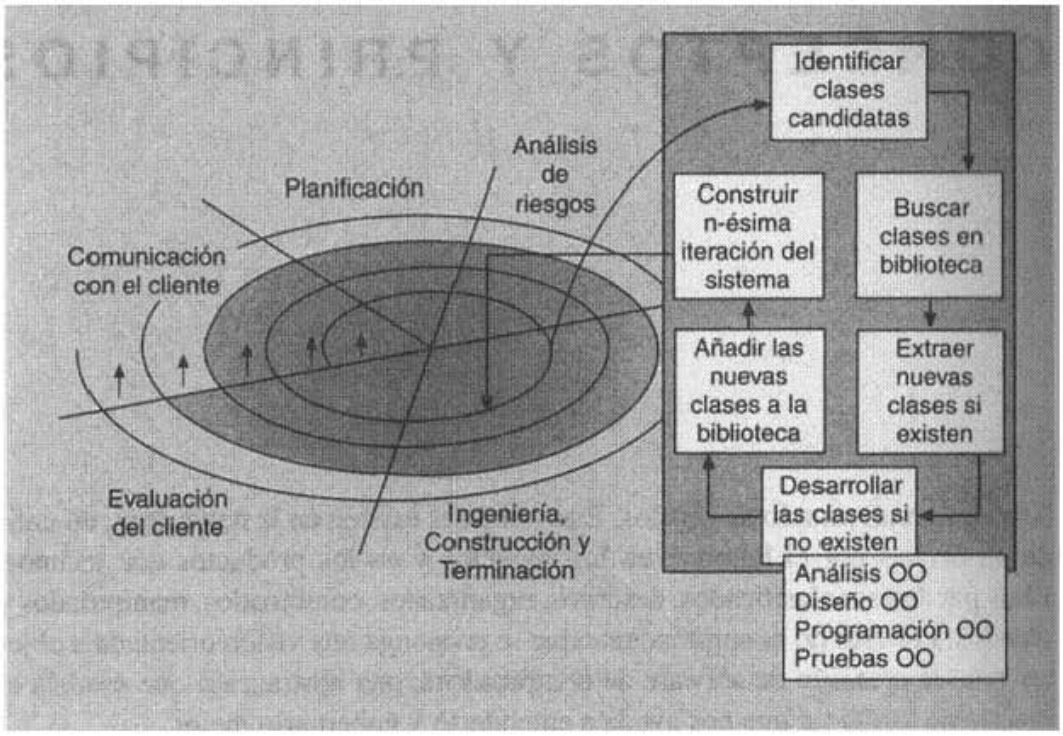


Figura I.2 Modelo de proceso OO.

En la figura I.2 se puede ver que una iteración principal corresponderá a un recorrido de 360 grados de la espiral. El modelo de proceso recursivo / paralelo engendrará un número de mini-espirales (iteraciones localizadas) que suceden durante el progreso de la iteración principal. Se sugiere que iteraciones de entre 2.5 y 4 meses de duración son más fáciles de seguir y gestionar.

Número de Contratos Completos

Un contrato es *un grupo de responsabilidades públicas relacionadas generadas por los subsistemas y clases a sus clientes*[3] . Un contrato es un punto de referencia excelente, y debe asociarse al menos un contrato a cada iteración del proyecto. Un jefe de proyecto puede usar contratos completos como un buen indicador del progreso en un proyecto OO.

I.3.3 Seguimiento del Progreso en un Proyecto Orientado a Objetos

Aunque el modelo de proceso recursivo / paralelo es el mejor marco de trabajo para un proyecto OO, el paralelismo de tareas dificulta el seguimiento del proyecto. El jefe del proyecto puede tener dificultades estableciendo fases o etapas en un proyecto OO debido a que cierto número de cosas están ocurriendo a la vez. En general, las siguientes etapas pueden considerarse "completados" (terminados) al cumplirse los criterios mostrados:

Etapa: análisis OO terminado

- Todas las clases, y la jerarquía de clases, están definidas y revisadas.
- Los atributos de clases y las operaciones asociadas a una clase se han definido y revisado.
- Las relaciones entre clases se han establecido y revisado.
- Se ha creado y revisado un modelo de comportamiento.
- Se han marcado clases reutilizables.

Etapa: diseño OO terminado

- El conjunto de subsistemas se ha definido y revisado.
- Las clases se han asignado a subsistemas y han sido revisadas.
- Se ha establecido y revisado la asignación de tareas.
- Se han identificado responsabilidades y colaboraciones.
- Se han diseñado y revisado los atributos y operaciones.
- El modelo de mensajería (paso de mensajes) se ha creado y revisado.

Etapa: programación OO terminada

---

[3] Roger S. Pressman Ingeniería del Software Un Enfoque Practico, Mc Graw Hill, 4ª edición.

- Cada nueva clase ha sido implementada en código a partir del modelo de diseño.
- Las clases extraídas (de una biblioteca de reutilización) se han integrado.
- Se ha construido un prototipo o incremento.

Etapa: prueba OO

- La corrección y compleción del análisis OO y del modelo de diseño han sido revisados.
- Se ha desarrollado y revisado una red de clases – responsabilidades – colaboraciones.
- Se han diseñado casos de prueba y ejecutado pruebas a nivel de clases para cada clase.
- Se han diseñado casos de prueba y completado pruebas de agrupamientos y las clases se han integrado.
- Las pruebas a nivel de sistema se han terminado.

Es importante destacar que cada uno de estos hitos puede ser visitado nuevamente al entregar diferentes avances al usuario.

I.4 Planteamiento del Problema

Una vez que se han presentado los elementos teóricos se empezara a trabajar sobre el SCG, para esto se utilizara el marco de proceso común (MPC) expuesto por Roger S. Pressman[ROG00].

Sé que muchos se preguntaran ¿En qué puede serle útil un software al instructor y/o al dueño de un gimnasio? Pues bien durante el tiempo que estuve entrenando me pude dar cuenta de varias situaciones que, con la ayuda de un software, serian mas fáciles y/o rápidas de hacer entre las cuales se encuentran:

- La realización de los cálculos necesarios para determinar la composición corporal y el *somatotipo* correspondiente a cada atleta en un menor tiempo ya que se entregaba un día después de la evaluación.
- Llevar el control del pago de las mensualidades, ya que este se lleva a cabo en una libreta, y si por alguna razón ésta se extraviara se perdería la información, y se tendría que confiar en los atletas o en la buena memoria de los instructores. Además si el instructor necesitara saber cuándo se le vence la mensualidad a algún atleta tiene que revisar la libreta hasta encontrar el último pago de éste.
- También es difícil para el instructor saber cuando le toca evaluación a cada atleta, a menos que éste último lo pida. Aunque en muchos de los casos por cuestiones de tiempo o simplemente para evitar ser cuestionados, por no llevar a cabo las indicaciones y no progresar en su condición, no la solicitan, o lo hacen cuando ya se le han juntado

varias evaluaciones al instructor. Causando con esto una carga de trabajo mayor para los instructores y un descuido de otros detalles. Por lo cual seria muy útil para el instructor saber a qué atletas les toca evaluación cada día.

- Una vez realizada la evaluación y determinado el *somatotipo* viene la dieta, la cual se proporciona a los atletas en un formato impreso junto con los resultados de la composición corporal, el somatotipo, y el tiempo durante el cual se tiene que seguir dicha dieta. Este formato lo debe presentar cada atleta en su siguiente evaluación, pero si lo llegará a extraviar no se cuenta con un archivo y/ o respaldo que permita conocer los datos en él contenido y por lo tanto no se puede saber cuánto ha progresado o retrocedido.
- Por otra parte en el gimnasio también se imparten cursos para atletas, terapias de rehabilitación y otros tratamientos de los cuales seria de gran importancia llevar un control adecuado.

Estas son las observaciones con las que se inicio el análisis del problema.

En el caso del gimnasio, las principales actividades que se pretenden automatizar mediante el sistema son:

- Registro de Atletas: Los Atletas se registran en una libreta en la cual escriben. Su nombre, la fecha en que están ingresando, el importe en pesos de lo que se pagó y su firma. Si se llegara a extraviar esta libreta, se perdería la información antes descrita de los atletas.
- Control del pago de las mensualidades: Actualmente en el gimnasio, si el instructor necesita saber cuándo se le vence la mensualidad a algún atleta, se tiene que buscar en toda la libreta hasta encontrar el último pago realizado por este atleta. Esta búsqueda se hace nombre por nombre, puesto que en la libreta no se lleva un orden para el registro de los pagos. Los atletas se registran conforme van llegando y pagando.
- Control de evaluaciones: Es difícil para el instructor saber cuándo le toca evaluación a cada atleta, a menos que este último lo solicite. Aunque en muchos de los casos por cuestiones de tiempo o simplemente para evitar ser cuestionados, por no llevar a cabo las indicaciones y no progresar en su condición física no la solicitan, o lo hacen cuando ya se le han juntado varias evaluaciones al instructor. Causando con esto una carga de trabajo mayor para los instructores y un descuido de otros detalles. Por lo cual sería muy útil para el instructor saber a que atletas les toca evaluación el día de hoy, por ejemplo.
- Asignación de dietas y determinación de la composición corporal: La dieta se proporciona a los atletas en un formato impreso junto con los resultados de la composición corporal, el somatotipo, y el tiempo durante el cual se tiene que seguir dicha dieta. Este formato lo debe presentar cada atleta en su siguiente evaluación, pero

si lo llega a extraviar no se cuenta con un archivo y/ o respaldo que permita conocer los datos en él contenidos y por lo tanto no se puede saber que tanto ha progresado o retrocedido el atleta.

- Cursos: En el gimnasio se imparten cursos para atletas e instructores de los cuales seria muy útil llevar un control de los pagos, de las inscripciones así como de los asistentes a cada curso.
- Terapias y tratamientos: Además de los cursos se brindan terapias de rehabilitación y otros tratamientos de los cuales también seria de gran importancia llevar un control y registro adecuado.
- En el caso del gimnasio se tiene que la realización de los cálculos necesarios para determinar la composición corporal y el somatotipo correspondiente a cada atleta, seria más rápida si se utilizara la computadora.
- Además, se le puede dar la oportunidad al instructor de preparar distintas dietas que pueda guardar en la computadora, para después asignárselas a los atletas y de esta manera aprovechar el tiempo libre que llegue a tener el instructor.

Los objetivos específicos del Sistema de Control de Gimnasios (SCG) son:

- Reducir el tiempo que se emplea para realizar la evaluación del atleta y la determinación de su composición corporal y somatometría.
- Facilitar el registro y control de los cursos, servicios, tratamientos, pagos, etc.
- Reducir tiempos en el procedimiento de inscripción de los atletas a los diferentes servicios del gimnasio.
- Presentar gráficas comparativas para conocer el avance del atleta de manera visual.
- Presentar de manera impresa los formatos empleados para informar al atleta sus resultados.

Cuento con la observación realizada durante el tiempo que estuve entrenando en el gimnasio. Lo que me dio una perspectiva sobre el manejo del mismo y me ayudó a identificar algunos de los problemas y las posibles oportunidades. Por otra parte me dio también cierta familiaridad y confianza con el personal que labora en él.

A continuación se presenta una descripción de la manera en la que se recaba la información mediante la observación, la búsqueda de documentos y la entrevista[4] ya que son los medios que se emplearán para el caso practico.

Algunas de las observaciones que se realizaron fueron las siguientes:

[4] http://jorgevilar.tripod.com.mx/busqueda.htm 2003

- Al llegar un nuevo atleta se le pide anotarse en una libreta. En esta libreta el atleta escribe su nombre y la fecha en que esta pagando además de su firma. Posteriormente se le informa al atleta que siempre que asista al gimnasio, antes de entrenar debe anotar en una lista su nombre y la hora a la que llego.
- Se le practica una evaluación para determinar composición corporal y somatometría al inscribirse y de esta manera al final del mes saber cuanto ha progresado. Los cálculos se realizan con una calculadora y por lo regular se le entregan al atleta un día después de la evaluación.
- Se hace una dieta para el atleta dependiendo de su somatotipo, la cual se le entrega un día después de la evaluación, con los resultados de la composición corporal y somatometría en un formato preimpreso que es llenado a mano.
- En el caso del gimnasio me fueron proporcionados los formatos que se emplean para registrar las medidas del atleta, determinar la composición corporal y finalmente entregar los resultados los cuales muestro a continuación en las figuras I.3, I.4 y I.5.

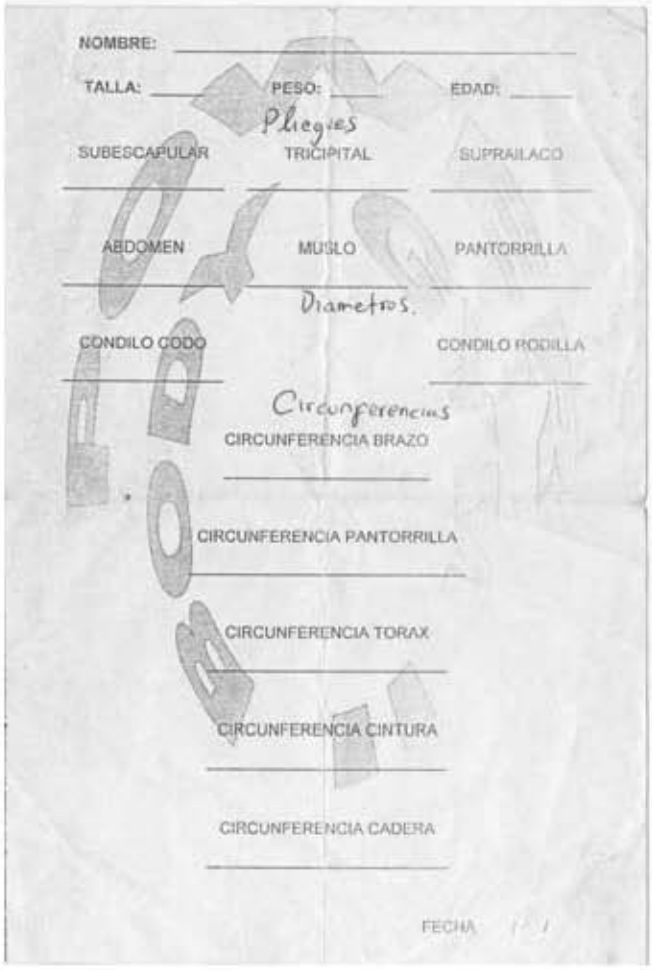
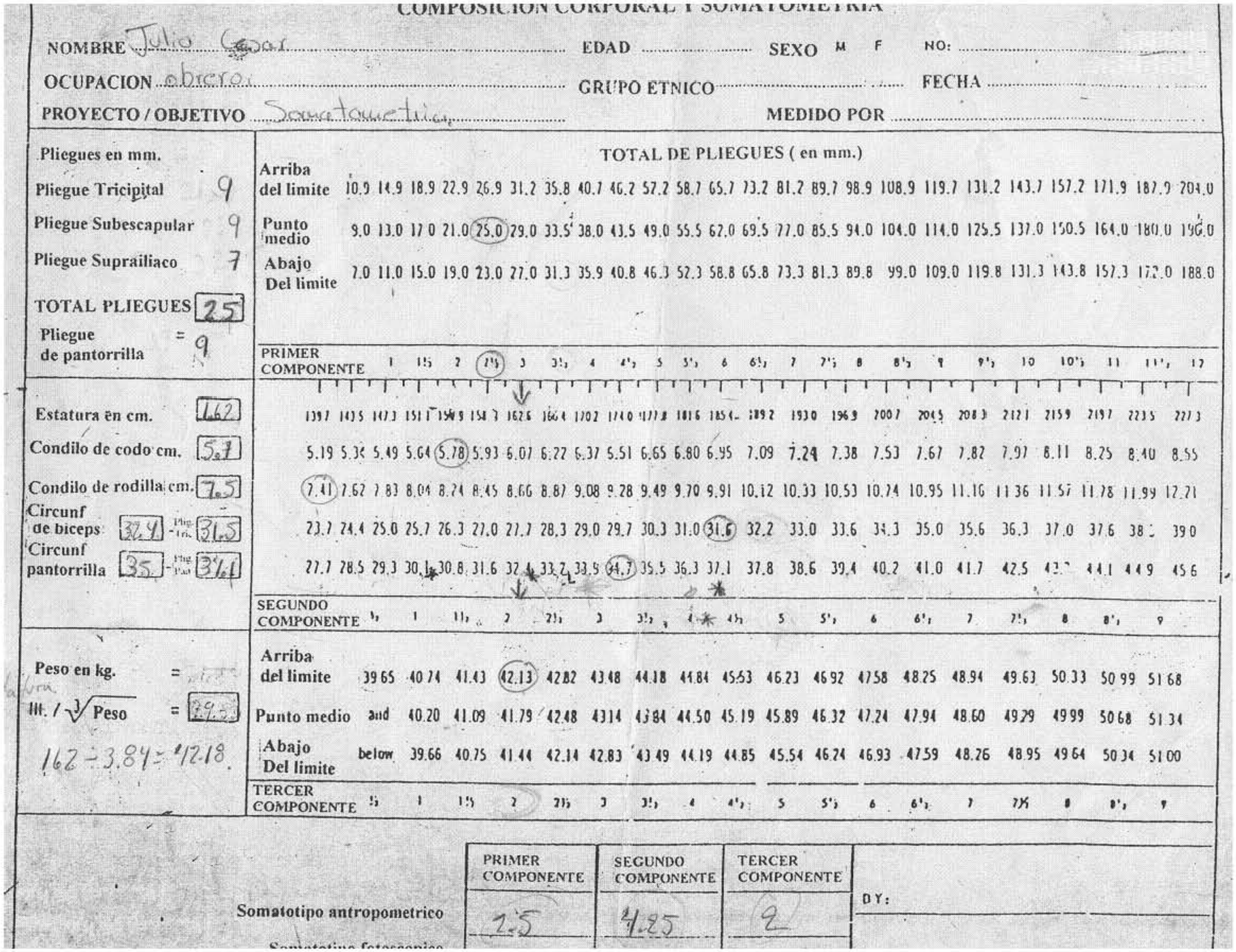


Figura I.3 Registro de las medidas de el(a) atleta.



COMPOSICION CORPORAL Y SOMATOMETRIA

NOMBRE Julio Cesar EDAD SEXO M F NO:
OCUPACION obrero GRUPO ETNICO FECHA
PROYECTO / OBJETIVO Somatometría MEDIDO POR

Pliegues en mm.		TOTAL DE PLIEGUES (en mm.)
Pliegue Tricipital	9	Arriba del limite 10.9 14.9 18.9 22.9 26.9 31.2 35.8 40.7 46.2 52.2 58.7 65.7 73.2 81.2 89.7 98.9 108.9 119.7 131.2 143.7 157.2 171.9 187.9 204.0
Pliegue Subescapular	9	Punto medio 9.0 13.0 17.0 21.0 25.0 29.0 33.5 38.0 43.5 49.0 55.5 62.0 69.5 77.0 85.5 94.0 104.0 114.0 125.5 137.0 150.5 164.0 180.0 196.0
Pliegue Suprailiaco	7	Abajo Del limite 7.0 11.0 15.0 19.0 23.0 27.0 31.3 35.9 40.8 46.3 52.3 58.8 65.8 73.3 81.3 89.8 99.0 109.0 119.8 131.3 143.8 157.3 172.0 188.0
TOTAL PLIEGUES	25	
Pliegue de pantorrilla	= 9	PRIMER COMPONENTE 1 1½ 2 2½ 3 3½ 4 4½ 5 5½ 6 6½ 7 7½ 8 8½ 9 9½ 10 10½ 11 11½ 12
Estatura en cm.	162	139.7 143.5 147.3 151.1 154.9 158.8 162.6 166.4 170.2 174.0 177.8 181.6 185.4 189.2 193.0 196.9 200.7 204.5 208.3 212.1 215.9 219.7 223.5 227.3
Condilo de codo cm.	5.7	5.19 5.34 5.49 5.64 5.78 5.93 6.07 6.22 6.37 6.51 6.65 6.80 6.95 7.09 7.24 7.38 7.53 7.67 7.82 7.97 8.11 8.25 8.40 8.55
Condilo de rodilla cm.	7.5	7.41 7.62 7.83 8.04 8.24 8.45 8.66 8.87 9.08 9.28 9.49 9.70 9.91 10.12 10.33 10.53 10.74 10.95 11.16 11.36 11.57 11.78 11.99 12.21
Circunf de biceps	32.4 - Plieg. tri. 31.5	23.7 24.4 25.0 25.7 26.3 27.0 27.7 28.3 29.0 29.7 30.3 31.0 31.6 32.2 33.0 33.6 34.3 35.0 35.6 36.3 37.0 37.6 38.2 39.0
Circunf pantorrilla	35 - Plieg. Pan. 34.1	27.7 28.5 29.3 30.1 30.8 31.6 32.4 33.2 33.9 34.7 35.5 36.3 37.1 37.8 38.6 39.4 40.2 41.0 41.7 42.5 43.3 44.1 44.9 45.6
		SEGUNDO COMPONENTE ½ 1 1½ 2 2½ 3 3½ 4 4½ 5 5½ 6 6½ 7 7½ 8 8½ 9
Peso en kg.	=	Arriba del limite 39.65 40.74 41.43 42.13 42.82 43.48 44.18 44.84 45.53 46.23 46.92 47.58 48.25 48.94 49.63 50.33 50.99 51.68
Ht. / $\sqrt[3]{Peso}$	= 42.5	Punto medio and 40.20 41.09 41.79 42.48 43.14 43.84 44.50 45.19 45.89 46.32 47.24 47.94 48.60 49.29 49.99 50.68 51.34
162 ÷ 3.84 = 42.18		Abajo Del limite below 39.66 40.75 41.44 42.14 42.83 43.49 44.19 44.85 45.54 46.24 46.93 47.59 48.26 48.95 49.64 50.34 51.00
		TERCER COMPONENTE ½ 1 1½ 2 2½ 3 3½ 4 4½ 5 5½ 6 6½ 7 7½ 8 8½ 9

	PRIMER COMPONENTE	SEGUNDO COMPONENTE	TERCER COMPONENTE	DY:
Somatotipo antropometrico	2.5	4.25	2	

Figura I.4 Formato de composición corporal y somatometría.

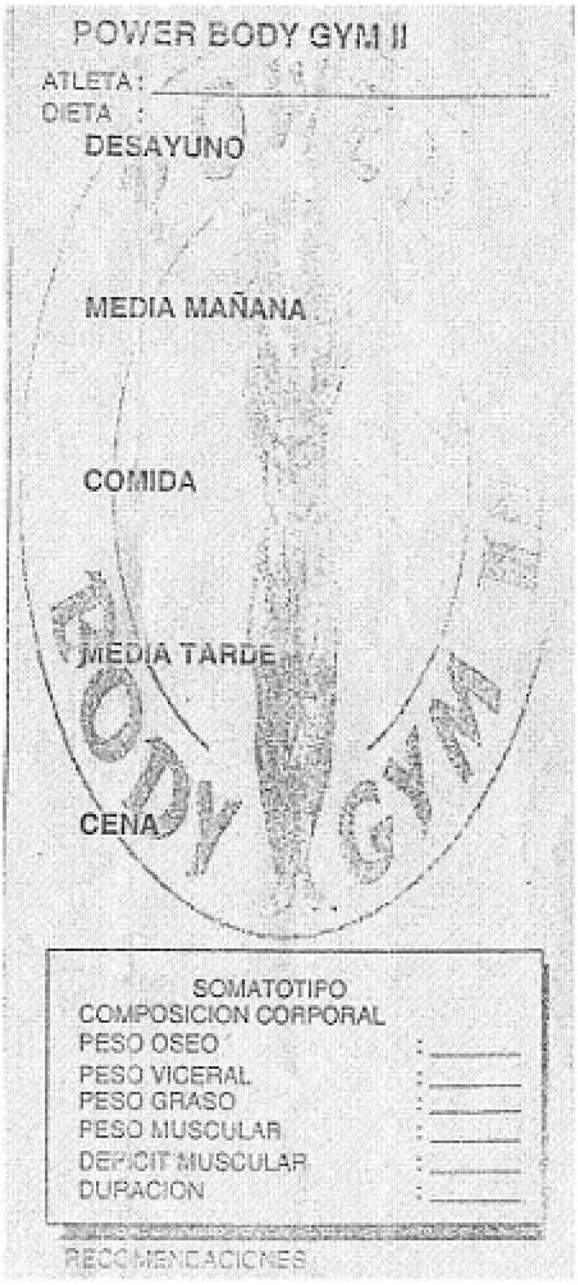


Figura I.5 Formato para entrega de los resultados.

Capítulo II Análisis del Sistema

Pensar es el trabajo más difícil que existe. Quizá esa sea la razón por la que haya tan pocas personas que lo practiquen.

HENRY FORD

II.1 Metodologías de análisis orientado a objetos OO

Al igual que en el enfoque estructurado, en la visión Orientada a Objetos se cuenta con diversas metodologías para el análisis orientado a objetos las cuales se enuncian a continuación con sus principales características de manera general.

El Método de Booch: abarca un micro proceso y un macro proceso. En el primero se definen un conjunto de tareas que se aplican en cada etapa del segundo (Conceptualización, Análisis, Diseño, Evolución y Mantenimiento). Los pasos fundamentales del micro proceso en la etapa de análisis son:

- Identificar clases y objetos
- Identificar la semántica de clases y objetos
- Identificar relaciones entre clases y objetos
- Especificar interfaces e implementación de las clases y objetos

El Método de Coad y Yourdon: Es uno de los métodos más sencillos, tanto por su simplicidad en la notación de modelado, como por sus reglas en el análisis. Surgió de un

esfuerzo por trasladar a la orientación a objetos las técnicas e ideas del análisis estructurado. El análisis consiste en:

- Identificar objetos
- Definir relaciones: generalización/especialización
- Definir relaciones: todo/parte
- Identificar temas
- Definir atributos
- Definir servicios

El Método de Jacobson: Es una versión simplificada del método OOSE (Object Oriented Software Engineering), también llamado Objectory por la herramienta que anteriormente le daba soporte. La característica más relevante de este método es la utilización de los "casos de uso". Los casos de uso pueden ser considerados una capa contra la que se verifican los requerimientos en todas las fases del ciclo de vida (Modelo de casos de uso del dominio, del análisis, diseño, implementación y comprobación).

En el análisis hay que:

- Identificar los usuarios del sistema y sus responsabilidades globales
- Construir un modelo de requisitos
- Definir los actores y sus responsabilidades
- Identificar los casos de uso para cada actor
- Preparar una visión inicial de los objetos del sistema y sus relaciones
- Revisar el modelo usando los casos de uso como escenarios para determinar su validez
- Construir un modelo de análisis
- Identificar objetos de interfaz usando información del tipo de actor-interacción
- Crear vistas estructurales de los objetos de interfaz
- Representar el comportamiento del objeto
- Aislar subsistemas y modelos para cada uno
- Revisar el modelo usando casos de uso con escenarios para determinar su validez

El Método de Rumbaugh: Conocido como la metodología OMT. Básicamente consiste en construir varios modelos con los diferentes aspectos del sistema:

- Desarrollar una declaración del ámbito del problema
- Desarrollar un modelo de objetos (estático)
- Identificar las clases relevantes del problema
- Definir atributos y asociaciones
- Definir enlaces entre objetos

- Organizar las clases utilizando la herencia
- Desarrollar el modelo dinámico
- Preparar los escenarios
- Definir eventos y desarrollar una traza de eventos para cada escenario
- Construir un diagrama de flujo de eventos
- Desarrollar un diagrama de estados
- Revisar el comportamiento para comprobar consistencia y compleción
- Desarrollar el modelo funcional
- Identificar entradas y salidas
- Usar diagramas de flujo de datos para representar transformaciones del flujo
- Desarrollar la especificación del proceso para cada función
- Especificar criterios y restricciones de optimización

Como puede observarse, las metodologías antes expuestas tienen puntos en común, sin embargo unos son mas complicados que otros.

En cuanto al análisis de este sistema se empleara la metodología de Coad y Yourdon expuesta por Kendall [KEN00] principalmente por las siguientes razones:

- Utiliza una notación muy simple, esto facilita el explicarle al usuario los distintos componentes que va a tener el sistema, así como la manera en que van a interactuar los mismos.
- Toma el método de análisis estructurado como punto de partida, lo cual agiliza la compresión de esta metodología a las personas que aún no están familiarizadas con los conceptos orientados a objetos.
- Las técnicas para descubrir objetos son básicamente las mismas que para descubrir procesos y entidades de datos. A excepción de algunos criterios, los cuales se explicarán más adelante.
- Se manejan conceptos que había manejado durante la carrera como: los diagramas de flujo de datos, árboles de decisión, etc.

II.2 Metodología de análisis orientado a objetos de Coad & Yourdon

Dado que para el sistema SCG se empleará la metodología de Coad & Yourdon, por los motivos antes expuestos, a continuación se presenta una descripción detallada de dicha metodología para después aplicar los conceptos al caso practico que se va a trabajar.

II.2.1 Capas

La metodología de Coad y Yourdon para el Análisis Orientado a Objetos (AOO) está basada en un modelo de 5 capas el cual se muestra en la figura II.1. Estas capas añaden una estructura tridimensional a la notación de análisis y diseño que da mayor poder para la representación de complejidad en sistemas flexibles. Las cuales se enuncian a continuación:

1. Capa Clase/Objeto. Indica las clases y objetos.
2. Capa de Estructura. Captura diversas estructuras de clases y objetos, tales como las relaciones uno a muchos y la herencia.
3. Capa de Atributos. Detalla los atributos de las clases.
4. Capa de Servicios. Indica los mensajes y comportamientos del objeto (servicios y métodos).
5. Capa de Tema. Divide el diseño en unidades de implementación o asignaciones de equipos.

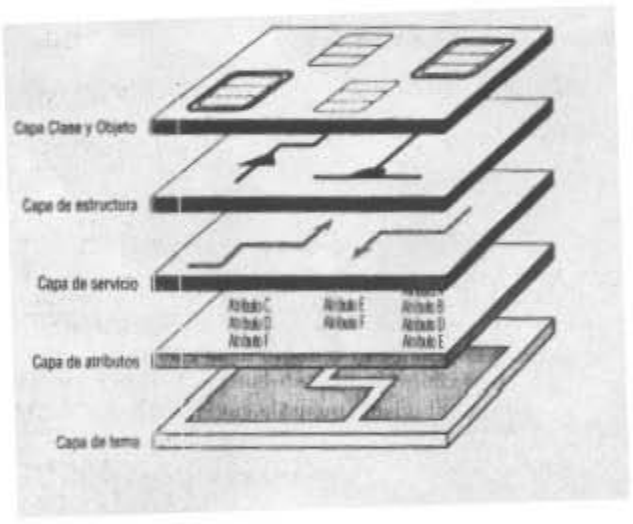


Figura II.1 Diagrama de Capas

II.2.2 Clases y Objetos

Coad y Yourdon distinguen Objeto, Clase y, Clase y Objeto de la siguiente forma:

- Objeto: es una abstracción de algo en un dominio de un problema que refleja las capacidades de un sistema para llevar información acerca de ello, interactuar con ello o ambas cosas. Una encapsulación de valores de atributos y sus servicios exclusivos. Sinónimo: una ocurrencia.

- Clase: una descripción de uno o más objetos con un conjunto de atributos y servicios uniformes, incluyendo una descripción de cómo crear nuevos objetos de la clase.
- Clase y objeto: un término que se refiere tanto a la clase como a los objetos que ocurren en la clase.

Hay 5 tipos generales de objetos en función de lo que representan y que pueden descubrirse durante el análisis.

1. Objetos que representan cosas tangibles.
2. Objetos que representan papeles actuados por personas u organizaciones. Dichos papeles incluyen objetos como cliente, propietario, departamento.
3. Objetos derivados de incidentes o eventos, tales como vuelos, accidentes o reuniones. Los incidentes suceden típicamente en un momento específico.
4. Objetos que indican interacciones tales como una venta o un matrimonio. Las interacciones tienen una cualidad de transacción o contrato.
5. Objetos que detallan especificaciones. Las especificaciones tienen estándares o una cualidad de definición y, por lo general, implican que otros objetos representarán ocurrencias de cosas tangibles.

A continuación se muestra la notación para Clase, Objeto y Clase y Objeto. Las clases son representadas por cuadros rectangulares redondeados (bubtángulos) divididos en tres partes. El nombre de la clase se muestra en la división superior del cuadro. Las otras dos divisiones se usan para las capas de atributo y servicio. Cabe señalar que en lo que respecta a la elaboración de los distintos diagramas de capas, se utilizara un software llamado visio2000©, ya que maneja la notación de la metodología de Coad y Yourdon que es la que se esta empleando en el presente trabajo.

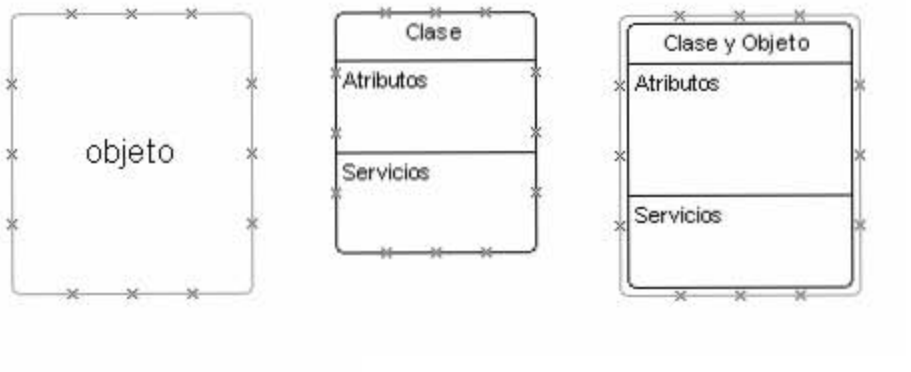


Figura II.2 notación que representa los conceptos de clase, objeto y clase y objeto.

Las técnicas para descubrir objetos son básicamente las mismas que para descubrir procesos y entidades de datos. Sin embargo, existen criterios que se pueden usar para determinar si se justifica una nueva clase de objetos:

1. Hay una necesidad de recordar el objeto. Esto es, el objeto puede ser descrito en un sentido definido y sus atributos son relevantes para el problema.
2. Hay una necesidad de determinados comportamientos del objeto. Esto es, aunque un objeto no tenga atributos, hay servicios que debe proporcionar o estados de objeto que deben ser llamados.
3. Usualmente un objeto tendrá varios atributos. Los objetos que tienen solamente uno o dos atributos sugieren diseños sobre analizados.
4. Usualmente una clase tendrá más de una instancia de objeto, a menos que sea una clase base.
5. Usualmente los atributos tendrán siempre un valor significativo para cada objeto de la clase. Los objetos que producen un valor NULO para un atributo, o para los que no es aplicable un atributo, por lo general implican una estructura generalización-especialización (Gen-Spec), la cual se explicará a detalle en la siguiente sección.
6. Usualmente los servicios siempre se comportarán en la misma forma para todos los objetos de una clase. Los servicios que varían dramáticamente para algunos objetos de una clase o que regresan sin realizar acción para algunos objetos también sugieren una estructura generalización-especificación.
7. Los objetos deben implementar requerimientos que son derivados del problema y no de la tecnología de solución. La parte de análisis del proyecto Orientado a Objeto no debe llegar a ser dependiente de una tecnología de implementación particular, tal como un sistema de computadora específico o un lenguaje de programación específico. Los objetos que atienden tales detalles técnicos no deben aparecer sino hasta muy tarde en la etapa de diseño. Los objetos dependientes de la tecnología sugieren que el proceso de análisis tiene fallas.
8. Los objetos no deben duplicar atributos o servicios que pueden ser derivados de otros objetos en el sistema.

II.2.3 Estructuras

Hay dos tipos básicos de estructuras que deben ser impuestas en las clases y objetos. Estas son la estructura generalización-especialización (“Gen-Spec”) y la estructura Todo-partes.

- Estructura GEN-SPEC. La herencia se crea con las estructuras Gen-Spec. Estas relaciones entre clases son a veces llamadas relaciones de clasificación, subtipo o ISA. Las estructuras Gen-Spec son indicadas por un semicírculo con su lado redondo hacia la clase generalizada. Estas estructuras siempre conectan clases a clases. Son típicamente de forma jerárquica. La figura II.3 muestra un ejemplo de una estructura Gen – Spec entre clases y objetos en donde las clases Autobús y Motocicleta heredan todas las propiedades de la clase Automóviles.

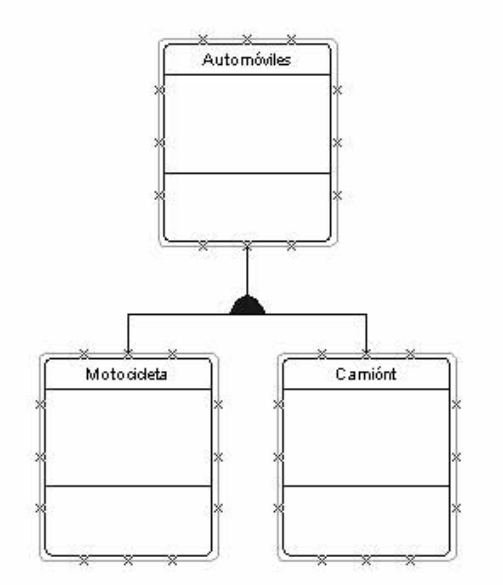


Figura II.3 Estructura Gen-Spec.

- Estructura TODO-PARTES. Estas estructuras indican conjuntos de diferentes objetos que componen otro objeto completo. Tales relaciones entre objetos son a veces llamadas relaciones de ensambles, agregaciones o TIENEUN. Las estructuras Todo-partes son indicadas por un triángulo que apunta hacia el objeto "completo". Estas estructuras conectan siempre objeto con objeto. En la figura II.4 se muestra una estructura Todo-Partes, en donde se muestran los objetos: Vehículo que a su vez está compuesto de los objetos motor y chasis.

Las estructuras Todo-partes también tienen cardinalidad, representada por uno a muchos y muchos a muchos. La notación "0, m" específica que, por ejemplo, un vehículo puede no tener motor (0) o uno o mas motores (m). La notación "0, 1" específica que un motor puede no ser parte de un vehículo (0) o de un vehículo (1), pero nunca mas de uno. El "1, m" específica que un vehículo puede tener uno o más chasis, pero nunca menos de uno. El "1" específica que un chasis siempre esta relacionado con uno y sólo un vehículo.

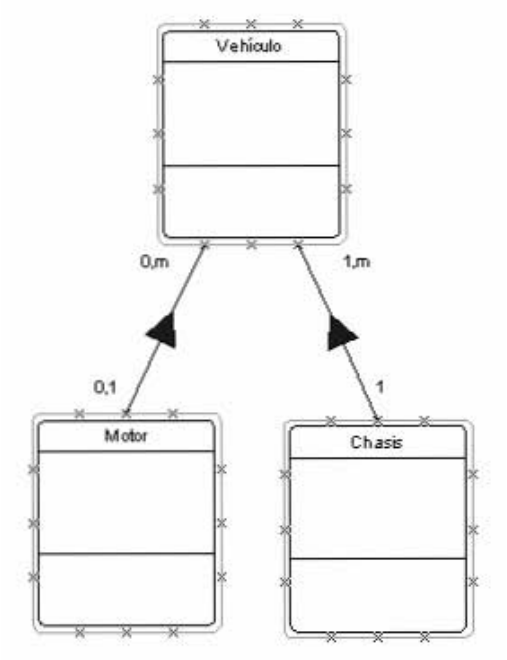


Figura II.4 Estructura Todo-Partes.

II.2.4 Atributos

Los nombres de los atributos de una clase se escriben en la sección central del cuadro de clase en el paquete de diseño. Aquí son relevantes tres ideas nuevas desde la perspectiva Orientada a Objetos.

- Primero, los atributos son siempre más propensos al cambio que las clases. Si una estructura o un conjunto de clases parece estar saturándose, debido a que un objeto esta cambiando de clase a clase, tal vez la clase y objeto en cuestión debiera simplemente llegar a ser un conjunto de atributos en otra clase.
- Segundo, los atributos deben mantenerse tan alto como sea posible en las estructuras Gen-Spec. Esta restricción reduce la programación y mantenimiento, debido a que un cambio hecho en un objeto Gen será automáticamente heredado por todos los objetos Spec.
- Tercero, las asociaciones o relaciones entre objetos (aparte de las estructuras) deben ser detalladas como conexiones de ocurrencia, en vez de llaves foráneas.

Para no saturar el paquete de diseño, no se especifican los atributos de llave primaria. Por consecuencia, las referencias entre objetos, tales como asociaciones y relaciones, se indican por una sola línea entre los objetos con la misma notación de cardinalidad usada en las estructuras Todo-partes. Las conexiones de ocurrencia suceden siempre entre objetos y no entre clases.

Esto lleva a realizar el modelo Entidad Relación (ER) de la Base de Datos para, de esta manera, determinar las relaciones existentes y sus respectivas cardinalidades. Existen varias notaciones para este fin (Chen, Barker, IDEF1X, etcétera), en el presente trabajo se empleará la notación IDEF1X, la cual es utilizada en varias herramientas CASE como es el caso del ERWIN 2000© software en el cual se realizarán los diagramas del modelo Entidad Relación (E/R).

Antes de explicar el porque se eligió IDEF1X se describen algunas de las notaciones mas populares con sus características más importantes.

Notación de Peter Chen

La notación original de Chen usa rectángulos para las entidades y diamantes para las relaciones (binarias o más grandes). Los atributos pueden estar definidos pero son excluidos del diagrama ER. Los roles son mostrados como parte de las relaciones, sus nombres pueden ser mostrados opcionalmente al final de la relación.

Chen formaliza las relaciones en términos de tuplas ordenadas de entidades, conservando el orden si fuera necesario excluirse por estar el nombre del role en uso (como con los nombres de atributos en el modelo relacional). Las relaciones pueden tener atributos, pero no pueden jugar un rol en otras relaciones. Los roles pueden tener anotaciones para indicar una cardinalidad máxima de uno o muchos.

Chen usa sustantivos en el diagrama para nombrar las relaciones eliminando la expresión natural. Aún cuando sean usados verbos, la dirección en la cual deben leerse es formalmente indecisa, a menos que se agreguen algunas marcas o reglas adicionales.

Este problema se agrava si el verbo usado para nombrar la relación esta acotada por una sola palabra (trabaja, lee, etcétera), desafortunadamente en la practica aún es algo muy común. Para eliminar ambigüedades se tienen que agregar nombres o verbos en los roles con una dirección definida.

Las entidades débiles se denotan por un rectángulo con bordes dobles, esto significa que debe tener una relación con otra entidad.

La punta de flecha al final de la relación indica la existencia de dependencia. Una mejor aproximación a la realidad es el introducir el concepto de role obligatorio, como en otras notaciones. Esta habilidad de establecer una multiplicidad de al menos uno para cualquier role dado esta ausente en la notación original de Chen.

Han sido desarrolladas muchas variantes para esta notación algunas veces se muestran elipses con nombre para representar entidades, que están conectadas por una flecha a elipses dobles que identifican a los atributos.

Las herramientas actuales de diseño de diagramas E/R de Chen usan hexágonos en lugar de diamantes. Un problema con estas variantes de E/R es que existen muchas versiones sin un solo estándar.

En la industria las versiones más populares de E/R puros son las notaciones de Barker y de Ingeniería de Información (IE) las cuales se exponen a continuación. Otra notación popular es la IDEF1X la cual es un híbrido de E/R y notación relacional.

Notación de Barker

Es la notación discutida en el tratado clásico de Richard Barker. Originado en CACI en el reino unido. Esta notación fue adoptada por Oracle corporation en sus herramientas de diseño CASE.

Entre sus convenciones básicas se encuentran las siguientes:

Las entidades son mostradas como rectángulos con las esquinas redondeadas con el nombre de la entidad en mayúsculas. Los atributos se escriben abajo del nombre de la entidad. Algunos símbolos para indicar restricciones pueden aparecer antes del nombre de un atributo.

Un "#" indica que el atributo o es el identificador o es componente del identificador de la entidad.

Un "*" o un punto duro "•" indica que el atributo es obligatorio, es decir que no acepta valores nulos. Cualquier instancia que forme parte del identificador primario dé la entidad debe tener un indicador de estos.

Un circulo "○" indica que el atributo es opcional algunos modeladores también usan un "." para indicar que el atributo no es parte del identificador.

Las relaciones se restringen a binarias (no unarias, ternarias, o mayores) y son mostradas como líneas con un nombre de relación y son mostradas como líneas con un nombre de relación que se coloca de la manera en que debe ser leído.

La colocación del nombre termina con la ambigüedad de dirección que se aprecia en la notación de Chen. Ambas lecturas hacia adelante y a la inversa pueden mostrarse en una relación binaria una sobre cada lado de la línea.

Cada línea o mitad de la línea que muestra una relación representa un role. Barker maneja por separado los roles de opcionalidad y cardinalidad. Incluyéndolos juntos bajo un solo y sencillo concepto. Una mitad de línea sólida indica un role obligatorio y una mitad de línea punteada indica un role opcional. Para la cardinalidad, una pata de cuervo intuitivamente indica muchos, por sus varios dedos y la ausencia de esta pata de cuervo indica uno.

Barker recomienda la siguiente disciplina de nombrado para relaciones dadas ARB denotando una relación R del tipo de entidad A al tipo de entidad B. Nombre R de manera que se cumplan cada uno de los siguientes 4 parámetros de la siguiente sentencia en ingles.

Each A (must|may) be R (one and only one B|one or more B – plural form).

Use "must" debe o "may" puede cuando el primer role es obligatorio u opcional, respectivamente. Use "one and only one" o "one or more" cuando la cardinalidad en el segundo role es uno o muchos respectivamente.

En esta notación una barra "|" cruzando el final de la relación indica que esta es un componente del identificador primario de la entidad que esta al final de dicha relación.

Las restricciones de exclusión sobre 2 o mas roles es mostrada como un arco exclusivo conectado a los roles con un pequeño punto • o circulo.

Para declarar que 2 o mas roles son mutuamente excluyentes y completamente disjuntos, la notación de Barker usa el arco exclusivo, pero cada role es mostrado como obligatorio (línea sólida).

Notación de la ingeniería de información (IE)

La notación de la ingeniería de información (IE) por sus siglas en ingles empezó con el trabajo de Clive Finkelstein en Australia y CACI en el Reino Unido y fue después adaptado por James Martin, existen diferentes versiones de IE sin un estándar.

IE es soportada por muchas herramientas de modelado de datos y es una de las más populares notaciones para diseño de bases de datos.

En la IE las entidades son representadas con rectángulos nombrados. Los atributos son frecuentemente mostrados en un compartimiento debajo del nombre de la entidad. Pero algunas veces son mostradas separadamente, algunas versiones soportan restricciones básicas sobre los atributos.

Las relaciones son restringidas a solo asociaciones binarias. Las cuales son representadas con líneas nombradas conectando las entidades. Los nombres de las relaciones son leídos de izquierda a derecha y de arriba hacia abajo.

Al igual que en la notación de Barker, una mitad de línea o una línea completa corresponde a un role. La opcionalidad y cardinalidad son indicados por anotaciones al final de las líneas. Para indicar que un role es opcional se coloca un circulo "○" al final de la línea significa una mínima participación esto es una frecuencia de 0. Para indicar que un role es obligatorio se coloca una barra "|" al final de la línea indicando con esto una mínima participación de 1. Experimentando después con diferentes notaciones para una cardinalidad de muchos, Finkelstein coloca el intuitivo símbolo de pata de cuervo sugerido por el Dr. Gordon Everest.

En conjunción con una frecuencia mínima de 0 o 1, una barra "|" es frecuentemente usada para indicar una frecuencia máxima de 1. Es decir una combinación "○|" indica al menos 1 y la combinación "||" indica exactamente 1. Sin embargo, como ya se menciono anteriormente existen diversas convenciones de IE.

Notación IDEF1X

En los 70as, la fuerza aérea de los Estados Unidos de Norteamérica empezó a trabajar en un programa de ayuda integral para manufacturación de computación (ICAM) por sus siglas en ingles. Este fue el génesis de una familia de lenguajes de modelado IDEF. El acrónimo "IDEF" originalmente denotaba "ICAM DEFinition" pero actualmente quedo como "Integration DEFinition". Reflejando esto la posibilidad de intercambiar información con diferentes lenguajes de modelado. Más que especificar un lenguaje de modelado universal. El proyecto ICAM definió los siguientes lenguajes para diferentes tareas.

- IDEF0: Actividad de modelado
- IDEF1: Modelado conceptual de datos
- IDEF2: Modelado de simulación

Después otros lenguajes fueron agregados, incluyendo los siguientes:

- IDEF1X: Modelado lógico de datos
- IDEF3: Modelado de procesos
- IDEF4: Diseño de Software Orientado a Objetos.
- IDEF5: Ingeniería del conocimiento de la ontología de empresas
- IDEF1X$_{97}$: Modelado lógico de datos con extensiones a orientación de objetos

El nombre "IDEF1X" quedo por "IDEF1 eXtended". Sin embargo basándose en el lenguaje conceptual IDEF1, tenemos que IDEF1X ha cambiado enfocándose al modelado lógico de datos.

Actualmente IDEF0 e IDEF1X son los más populares lenguajes IDEF, ambos son soportados por varias herramientas CASE y ampliamente usados por varios sectores del gobierno de los Estados Unidos de Norteamérica especialmente el de defensa.

IDEF1X es un lenguaje híbrido que combina algunas notaciones conceptuales (entidad, relación) con construcciones de bases de datos relacionales (llaves primarias). Esta notación fue aceptada como un estándar por el instituto nacional de estándares y tecnología (NIST por sus siglas en ingles) en 1993. Un propuesto sucesor llamado $IDEF1X_{97}$ fue aprobado en junio de 1998 por el IEEE-SA Standards Board. Este lenguaje también conocido como $IDEF1X_{object}$ es una extensión del IDEF1X con aspectos orientados a objetos para hacerlo idóneo para implementaciones en bases de datos orientadas a objetos y lenguajes de programación, mientras mantiene la compatibilidad con el IDEF1X para implementación de bases de datos relaciónales. El mantener esta compatibilidad da a $IDEF1X_{97}$ una ventaja sobre UML por esto la adopción del $IDEF1X_{97}$ por diversos sectores del gobierno de Estados Unidos de Norteamérica.

Esta notación es explicada a detalle en la sección III.2.2 del presente trabajo. Ahora bien si se quiere saber más acerca de la notación IDEF1X las personas interesadas pueden consultar el libro de Terry Halpin [TERRY] donde resume esta notación o bien si se desea profundizar se puede leer el trabajo del autor de esta notación Bruce [BRU92].

Aquí se expresan las razones por las cuales utilizo esta notación, para la realización del diagrama correspondiente al modelo E/R.

- El aprender una nueva notación para modelos E/R, ya que soy de la idea de que no hay mejor forma de aprender que con la practica.
- Es la notación utilizada por el ERWIN 2000 software que se empleara para realizar los diagramas del modelo E/R.
- Al ser un híbrido maneja diversos conceptos, de los cuales, la mayoría me fueron instruidos a lo largo de la carrera.
- La curiosidad que despertó en mí, por el hecho de ser usada en sectores del gobierno de Estados Unidos como el de defensa.
- Aplicar esta notación a un problema y de esta manera tener un punto de comparación con las notaciones que me enseñaron en la carrera como son las de Chen y Barker.

Ahora retomando lo que es la capa de atributos. Con la introducción de los atributos necesitamos detalles de análisis adicional para dar soporte al paquete de diagrama en capas. En esta etapa del análisis estos detalles toman en cuenta solamente descripciones de los atributos y restricciones de sus valores. Coad y Yourdon recomiendan una *plantilla de especificación* que proporcione un esquema similar al diccionario de datos. Esta plantilla puede extenderse o modificarse conforme continúa el análisis.

II.2.5 Servicios

Los servicios también llamados métodos o procedimientos, llegan a ser parte de los objetos, en forma muy similar a los atributos. Debido a que los servicios involucran frecuentemente cambios en el estado de un objeto, es común que sean analizados y diseñados usando diagramas de estado. Por consecuencia, el análisis de servicios consiste de tres actividades: análisis del estado del objeto, especificación de servicio y especificación de mensaje.

Análisis del estado del objeto

Se pueden descubrir cambios de estado encontrando los atributos de cada objeto que afectan su comportamiento. Mientras se examina cada atributo, hay que preguntarse, "¿Cambiará el comportamiento del objeto cuando cambie el valor de este atributo?" cuando ningún atributo cambia el comportamiento del objeto, pero todavía se sabe que el objeto se comportara diferente bajo ciertas condiciones, se debe añadir un atributo de "estado". Los diagramas de estado se agregan a las plantillas de especificación para documentar tales atributos de estado conforme se necesita.

Especificación de servicio

Los servicios son categorizados como simples o compuestos.

- Los servicios simples involucran muy pocas condiciones u operaciones, y frecuentemente se aplican a cada Clase y Objeto en un sistema. Estos incluyen servicios tales como crear objeto, almacenar objeto, recuperar objeto, conectar objeto (hacer una ocurrencia de conexión), accesar objeto (obtener o poner los valores de los atributos) y borrar objeto. Los servicios simples son implícitos, a veces especificados una sola vez en el diseño y nunca vueltos a mencionar. Ocasionalmente, servicios simples muy obvios no son mencionados para nada en el diseño, sino que son dejados para que los programadores los construyan cuando los necesiten durante la implementación. Los servicios simples pueden aparecer en las plantillas de especificación, pero no son mencionados en el paquete del diagrama en capas.
- Los servicios compuestos involucran ciclos, muchas operaciones o condiciones compuestas. Estos servicios se aplican solamente a una Clase y Objeto. Los servicios complejos frecuentemente ocasionan servicios "compañeros" o "privados" que son similares a los módulos de subrutinas. Estos servicios complejos son mostrados siempre en la capa de servicio. Los nombres de tales servicios aparecen en la sección inferior de los cuadros de clase. Especificamos servicios complejos en forma mas completa en la plantilla de especificación. Se puede usar casi cualquier herramienta de especificación procedural, tales como diagramas de flujo de programa, diagramas Warnier-Orr, tablas de decisión o lenguaje estructurado.

Los servicios privados son subrutinas internas, de las cuales solo el objeto mismo sabe y puede activar.

Los servicios compañeros son subrutinas usadas por servicios complejos, y también pueden ser activadas como servicios distintos por mensajes de otros objetos del sistema.

Especificación de mensaje

Los mensajes detallan cómo el comportamiento de un objeto puede activar el comportamiento de otro objeto. Esto es, los mensajes son generados por un objeto con la intención de activar un servicio en otro objeto. Esencialmente, los mensajes documentan la dependencia de un proceso sobre otro proceso en un objeto diferente. Los mensajes existen solamente para comunicarse entre servicios, y ocasionan flujo de control y flujo de datos.

Los mensajes que se refieren a servicios simples no son documentados en el paquete de diagrama en capas, debido a que están implícitos, por lo general, en los servicios simples. Los mensajes dirigidos a las clases, tales como crear objeto y borrar objeto, generalmente tampoco son diagramados. Por consecuencia, la mayoría de los mensajes diagramados terminan siendo de objeto a objeto, y no de clase a clase o de objeto a clase.

Los mensajes se muestran como líneas anchas en la capa de servicio o en el paquete de diagrama en capas. Debido a que los mensajes detallan servicios complejos, se alinean típicamente con el servicio emisor y el servicio receptor. Sin embargo incluso en sistemas moderadamente complejos habrá muy pocos servicios complejos y, por consecuencia muy pocos mensajes en el diagrama. Este hecho tiende, naturalmente, a resaltar las funciones realmente importantes del sistema.

Los servicios complejos que no son activados por mensajes, tienden a ser activados por eventos temporales o interacción humana. Una clase sin un mensaje de entrada, pero con un servicio complejo, necesitara, por lo general, algún tipo de interfaz humana.

Cuando se añaden detalles de servicio a la plantilla de especificación Orientada a Objetos esta parte del análisis se inunda de detalles. Los diagramas de estado, lenguaje estructurado y diagrama de flujo pueden ser largos. El formato exacto de la especificación puede variar considerablemente de análisis a análisis. En la figura II.5, se puede apreciar un guión básico para una *Plantilla de Especificación*. Es posible añadir cualquier detalle de objeto importante que necesite ser indicado a los diseñadores y programadores.

Plantilla de Especificación

- Nombre de Clase y Objeto
- Atributos
  - Dominio de Valores
- Entrada externa
- Salida externa
- Diagrama de estado
- Restricciones adicionales
- Notas
- Servicios
  - Valores de entrada
  - Valores de salida
  - Ingles estructurado
  - Diagrama Warnier-Orr
  - Tabla de Decisión
- Códigos de error
- Códigos de estado
- Requerimientos de tiempo
- Requerimientos de memoria

Figura II.5 Formato de una Plantilla de Especificación.

II.2.6 Análisis de temas

En el caso de sistemas muy grandes, se pueden usar una capa adicional en el paquete de diagrama en capas OO para organizar el trabajo de análisis, diseño e implementación. Esta capa llamada "capa de tema" proporciona un medio para subdividir una especificación compleja en unidades de trabajo lógicas. Una capa de tema es solamente necesaria en proyectos grandes que involucran muchas clases. Los temas son resaltados trazando una línea de guiones ancha que indica las fronteras de un tema particular en el diagrama OO. El nombre del tema se anota en una esquina del cuadro de tema.

Por lo general, un tema tendrá una clase "propietaria" aparente. Esta es una clase que esta conectada centralmente con todas las clases y objetos en el espacio del tema.

- No se incluye el diagrama de la capa de tema, puesto que, en este momento el sistema no es grande. Más delante de ser necesario por el tamaño del programa, se incluirá el diagrama de la capa de tema.

II.3 Análisis para el sistema de control de gimnasios SCG

Finalmente, se aplicarán los conceptos expuestos en este trabajo. Para este fin el presente trabajo se concentrara en el programa SCG14 (Evaluación de los Atletas). Puesto que dentro del sistema es el más completo y de los mas importantes. Ya que en este programa se registran datos tales como: el nombre y la clave del atleta, la fecha de la evaluación, las medidas del atleta, la clave del instructor, la clave y descripción del somatotipo, además de calcular los tres componentes, determinar el somatotipo y abrir algunos de los catálogos.

Para el programa SCG14 las primeras 4 clases que se identifican son: Atletas, Instructores, Evaluaciones y Somatotipos. De manera tal que el diagrama inicial de la capa clase y objeto se muestra en la figura II.6.

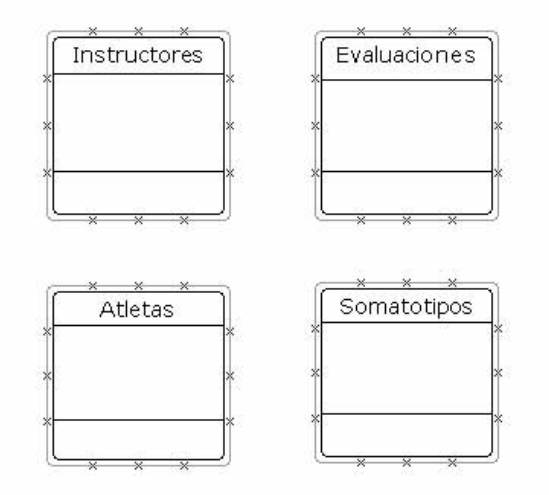


Figura II.6 Diagrama Inicial de la Capa Clase y Objeto

Por consiguiente para las 4 clases antes mencionadas se tiene que el diagrama inicial de la capa de atributos quedaría como se muestra en la figura II.7

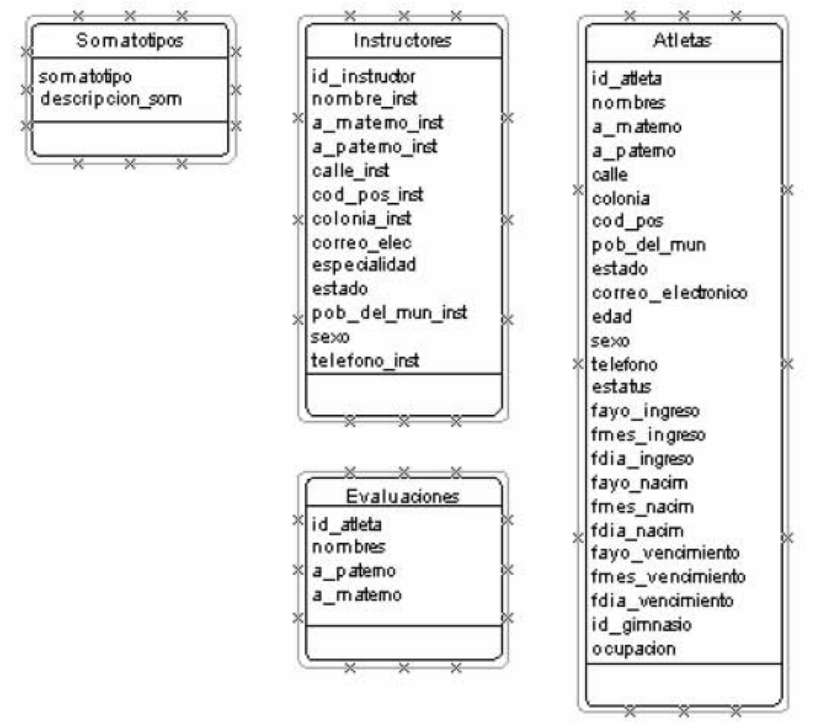


Figura II.7 Diagrama Inicial de la Capa de Atributos

Antes de empezar a realizar el modelo E/R para el programa SCG14 es necesario tomar en cuenta lo siguiente:

- Crear una tabla donde se almacene la clave y el nombre del atleta al que se le está practicando la evaluación, esto porque los atletas no siempre acuden al gimnasio para el

servicio de pesas, puede ser que sólo asistan a un curso o para recibir algún tratamiento o terapia de rehabilitación. Por esta razón, hay verificar que la clave tecleada en el programa se encuentre en el Catálogo de Atletas para evitar errores en la captura y por consiguiente se ocasione una duplicidad en los datos afectando la integridad y consistencia de los mismos.

- En este programa se va a determinar el somatotipo, por esta razón se tiene que validar contra el Catálogo de Somatotipos, que la clave determinada por el programa exista.
- La evaluación no siempre es realizada por el mismo Instructor, por lo tanto, se tiene que verificar que la clave de instructor tecleada en el programa exista en el Catálogo de Instructores.
- Las medidas del atleta así como el resultado de cada evaluación se deben guardar, para que sea posible realizar una comparación de resultados, y de esta manera ver la evolución del atleta a través del tiempo. En la figura II.8 se puede observar el modelo E/R final para este programa.

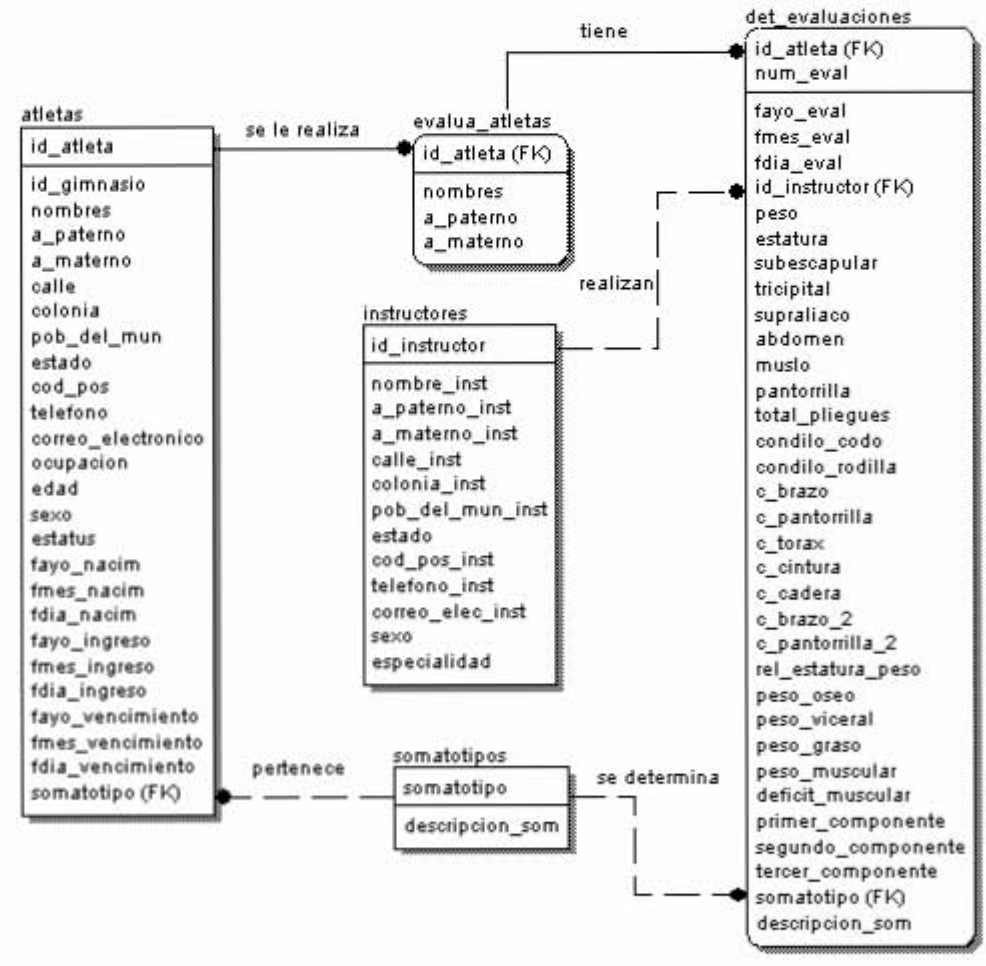


Figura II.8 Modelo Entidad Relación del Programa SCG14

En el diagrama se puede apreciar que:

- La entidad Atletas tiene una relación identificable[1] con la entidad evalua_atletas. Una relación identificable, ocurre cuando la llave primaria de la entidad padre, pasa a formar parte de la llave primaria de la entidad hijo. Cuando usamos una relación de este tipo, la entidad hijo se vuelve una entidad dependiente del padre, tanto en su identificación como en su existencia.
- Una relación identificable se representa por una línea que conecta las dos entidades, con un punto sobre el extremo "muchos" (correspondiente a la entidad hijo), y un nombre (generalmente un verbo) sobre la línea.
- La entidad atletas tiene una relación no identificable con la entidad somatotipos. Una relación no identificable, ocurre cuando la llave primaria de la entidad padre, pasa a formar parte de uno de los atributos de la entidad hijo, sin formar parte de la llave primaria de esta última.
- Una relación no identificable se representa por una línea punteada conectando las dos entidades, con un punto en el extremo "muchos". En las relaciones no-identificables, la existencia del hijo puede aún depender del padre, aunque su identidad no dependa de él. Si la relación es obligatoria desde la perspectiva del hijo, entonces la existencia del hijo depende del padre. Si es opcional, la existencia del hijo no depende del padre. Las relaciones opcionales son denotadas con un rombo blanco en el extremo correspondiente al padre.
- La entidad evalua_atletas tiene una relación identificable con la entidad det_evaluaciones. Ya que la clave primaria de la entidad evalua_atletas pasa a formar parte de la llave primaria de la entidad det_evaluaciones.
- La entidad somatotipos tiene una relación no identificable con la entidad det_evaluaciones. Esto debido a que la llave primaria de la entidad somatotipos pasa a formar parte de los atributos de la entidad det_evaluaciones, sin ser parte de la llave primaria de esta última.
- La entidad instructores tiene una relación no identificable con la entidad det_evaluaciones. Puesto que su llave primaria pasa a formar parte de los atributos de la entidad det_evaluaciones, sin ser parte de la llave primaria de det_evaluaciones.

---

[1] el término relación identificable y no identificable fue la traducción que me pareció mas lógica o adecuada para el término identifying relationship y nonidentifying relationship, que se desprenden del texto en inglés que consulte sobre la notación IDEF1X [BRU92]

- En la entidad det_evaluaciones se guardan todas las medidas del atleta, así como los resultados de la evaluación, incluyendo el somatotipo al que se determino que pertenece el atleta, además de la clave y el nombre del instructor que realizo la evaluación.

Después de revisar el diagrama del modelo E/R la capa de atributos quedaría como lo muestra la figura II.9.

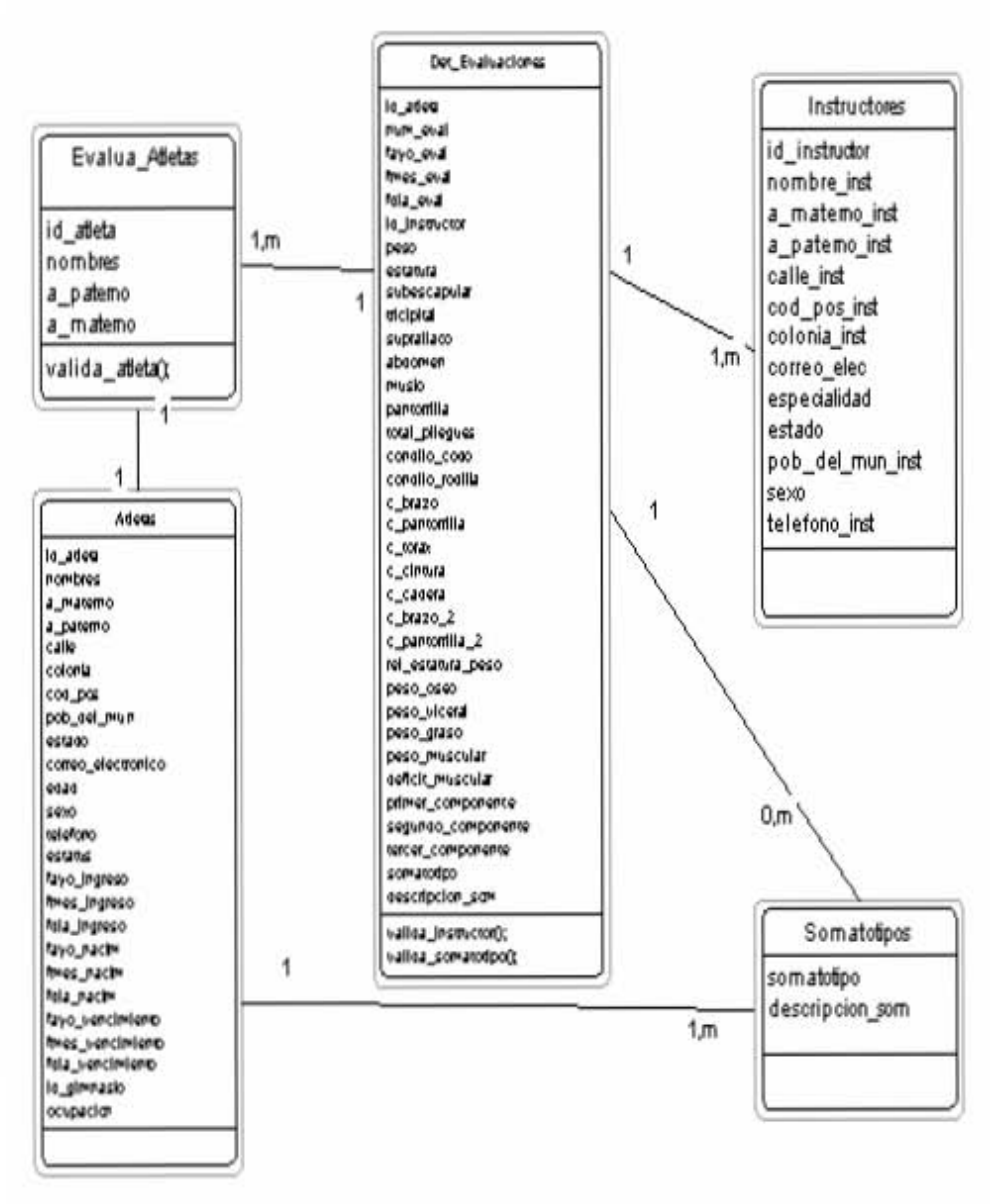


Figura II.9 Diagrama de la Capa de Atributos para el SCG14

En la figura II.9 se puede apreciar que se tienen identificados tres servicios que son: valida_atleta, valida_instructor y valida_somatotipo. Estos servicios, como su nombre lo indica se encargarán de verificar que las claves de atleta, de instructor y de somatotipo que se

introduzcan existan en los catálogos del sistema. Esto con el fin de evitar errores en la captura de estos datos.

A continuación en la ffigura II.10 se muestra la plantilla de especificación preliminar inicial después de terminar la primera iteración del análisis. A lo largo del diseño esta plantilla se complementará, agregando las clases que sean necesarias, al ir avanzando el proceso de diseño.

Plantilla de Especificación
- Atletas
- Atributos
  - id_atleta
  - num_eval
  - fayo_eval
  - fmes_eval
  - fdia_eval
  - id_instructor
  - peso
  - estatura
  - subescapular
  - tricipital
  - supraliaco
  - abdomen
  - muslo
  - pantorrilla
  - total_pliegues
  - condilo_codo
  - condilo_rodilla
  - c_brazo
  - c_pantorrilla
  - c_torax
  - c_cintura
  - c_cadera
  - c_brazo_2
  - c_pantorrilla_2
  - rel_estatura_peso
  - peso_oseo
  - peso_viceral
  - peso_graso
  - peso_muscular
  - déficit_muscular
  - primer_componente
  - segundo_componente
  - tercer_componente
  - somatotipo
  - descripción_som
- Entrada externa
  - Mediante el menú scgm1 o mediante el programa de atletas.
- Salida externa
- Diagrama de estado
- Restricciones adicionales
- Notas
- Servicios
  - Valida_Atleta.
  - Valida_Instructor.
  - Valida_Somatotipo.
  - Valores de entrada
  - El valor tecleado en los campos de Atletas, Instructores Y Somatotipos.
  - Valores de salida
  - Si el valor tecleado existe en el catálogo de Atletas, Instructores o Somatotipos, devuelva el nombre del Atleta, del Instructor o la descripción del somatotipo elegido.
  - Si el valor tecleado no existe en el catálogo de Atletas, Instructores o Somatotipos, devuelva el mensaje de error correspondiente.
  - Ingles estructurado
  - Diagrama Warnier-Orr
  - Tabla de Decisión
- Códigos de error
  - Esa clave no se localiza en el Catálogo de Atletas.
  - Esa clave no se localiza en el Catálogo de Instructores.
  - Esa clave no se localiza en el Catálogo de Somatotipos.
- Códigos de estado
- Requerimientos de tiempo
- Requerimientos de memoria

Figura II.10 plantilla de especificación inicial para el programa SCG14.

Con esto se puede decir que se concluyo el 1er. Hito técnico: análisis OO del programa scg14 terminado. En su primera iteración, ya que, como se ha mencionado anteriormente, este es un proceso iterativo, razón por la cual, no se puede considerar totalmente terminado el análisis, sin tomar en cuenta los cambios que sufrirá el scg14 a lo largo del diseño, el cual se aborda en el siguiente capítulo.

Capítulo III Diseño del Sistema

Quien poco piensa, mucho yerra.

LEONARDO DA VINCI

En este capítulo se presenta el diseño, de la misma forma que se ha hecho en los capítulos anteriores. Primero se presentan las generalidades de la metodología Coad y Yourdon para posteriormente aplicarla al caso especifico del SCG.

Las actividades de diseño en el enfoque de Coad y Yourdon emplean las herramientas de análisis para completar el conjunto de especificaciones en la implementación. Cuando el análisis es razonablemente independiente de la tecnología, las actividades del diseño están cada vez más enfocadas hacia un lenguaje OO particular y al ambiente de desarrollo.

III.1 Diseño orientado a objetos de Coad & Yourdon

Las actividades del diseño OO están agrupadas en cuatro componentes principales del sistema final: el problema, la interfaz humana, el manejo de datos y el manejo de tareas.

Por otro lado, cada una de las cinco capas del análisis OO se extiende conforme se necesita a lo largo de los cuatro componentes de diseño.

Toda la documentación del análisis debe llevar directamente hacia la etapa de diseño. En este punto se necesitan pocas herramientas nuevas. El paquete de diagrama en capas y la plantilla de especificación siguen siendo los componentes principales del diseño. Estos documentos no son complementados o reemplazados, sino que en vez de ello, son ampliados para incluir los detalles de implementación restantes durante la fase de diseño.

Durante el diseño por lo regular se usan prototipos (versiones burdas de los objetos) que se prueban con los cuatro componentes. Por lo cual, los diseñadores usan frecuentemente el lenguaje de implementación esperado como el mecanismo para escribir especificaciones completas de las clases.

III.1.1 Diseño del componente de dominio problema (PDC)

El componente de dominio problema (PDC) es el conjunto básico de objetos funcionales que llega de la etapa de análisis. Estos objetos resuelven directamente el problema que se pretende sea resuelto por el sistema que se está construyendo. Los otros componentes, tales como la interfaz humana y el manejo de datos, son funciones incidentales que deben ser añadidas al PDC para "hacer que trabaje".

Por consecuencia, el diseño del PDC se termina en su mayor parte en la etapa de análisis. Solamente se necesitan tres actividades para completar el diseño de PDC: diseño de reuso, estructuras de implementación y acomodo al lenguaje.

Diseño de re-uso

Se realiza cuando deseamos añadir nuevas clases al PDC para rehusar objetos. Por ejemplo, hay paquetes comerciales de clases altamente generalizadas para objetos. Una organización de programación OO con experiencia, por lo general posee una biblioteca de clases desarrollada en casa para los objetos. Estas bibliotecas y paquetes pueden contener clases

que tienen atributos y servicios para objetos similares a los requeridos en este diseño. Se pueden añadir esas clases reusables a nuestro diseño como clases base en una estructura Gen-Spec. Las clases derivadas en estas estructuras son desarrolladas originalmente en la etapa de análisis.

Estructuras de implementación

Esta actividad se realiza cuando se desean añadir otras estructuras a nuestro diseño, simplemente por razones de implementación; o tal vez se quieren usar estructuras de agregación para crear puntos de entrada naturales para listas o filas, o una estructura Gen-Spec para permitir que varias clases de objetos compartan un protocolo o estructura de datos. Estas estructuras usan el concepto de herencia para hacer mucho más fácil la tarea de programación.

Acomodo al lenguaje

Cuando se necesita corregir el diseño para que las estructuras puedan ser construidas en el lenguaje de programación seleccionado, debido a que estos lenguajes pueden tener diferentes patrones de herencia se efectúa un acomodo al lenguaje. Algunos lenguajes incluyen herencia múltiple, otros solo incluyen herencia simple y otros todavía no incluyen herencia. En los casos más restrictivos, los patrones de herencia del diseño deben ser modificados para permitir las capacidades del lenguaje de implementación.

III.1.2 Diseño del componente de interfaz humana (HIC)

En esta actividad se crean los menús, reportes y pantallas interactivas que emplearán los usuarios del sistema. Estas actividades son básicamente las mismas que las de diseño de la interfaz de usuario. Sin embargo, en el contexto OO esta fase culmina en una especificación para nuevas clases componente de interfaz humana (HIC) que deben ser añadidas al diseño.

Por lo general, es posible apoyarse en gran medida en las clases de una biblioteca para hacer el diseño de clases HIC. Esta es un área donde la reusabilidad de las clases OO ha probado ser muy efectiva. Las clases de biblioteca por lo general proporcionan generalizaciones de menús, ventanas, control de ratón, iconos, control de tipos de letra y utilerías de cortar y pegar.

Los prototipos son muy útiles durante el diseño HIC para suavizar como trabajarán las clases de biblioteca con los objetos PDC. La interacción del usuario con los prototipos puede proporcionar información extremadamente útil acerca de la efectividad del diseño.

III.1.3 Diseño del componente de administración de tarea y datos (TMC y DMC)

Estos dos componentes están enlazados muy fuertemente con la tecnología de implementación. El manejo de tareas está en función de la configuración del hardware, y el manejo de datos esta determinado por el software disponible cuando el sistema esté en ejecución.

El componente de manejo de tareas (TMC) es más importante cuando el sistema se está ejecutando en varios procesadores o computadoras. Una "tarea" es un conjunto de servicios relacionados que se deben ejecutar juntos (tal vez en el mismo procesador). Las tareas son activadas por el tiempo transcurrido o por un evento. Los objetos del TMC obedecen a activadores de tarea, asignación de procesadores y prioridades cuando son llamados los servicios.

El componente TMC aparece, por lo general, como se muestra en la figura III.1. Este nuevo componente TMC se añade al paquete de diagrama de capas existente y después se implementa creando nuevos objetos tarea si son requeridos por el sistema.

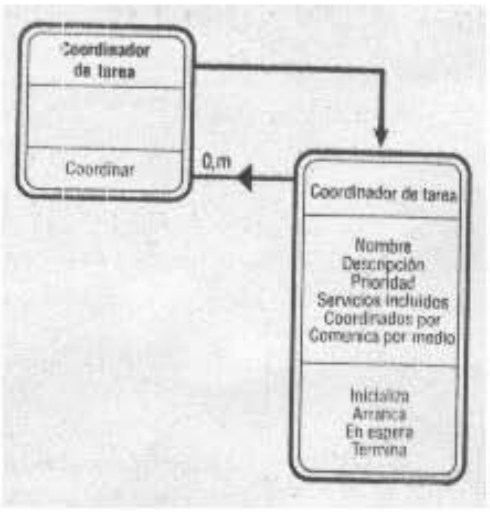


Figura III.1 Componente de Manejo de Tareas.

El componente de manejo de datos (DMC) comprende, por lo general, clases y objetos necesarios para almacenar y recuperar otros objetos del sistema. Además, varía considerablemente dependiendo de sí la tecnología de tiempo de ejecución subyacente es una base de datos orientada a objetos, una base de datos relacional o un sistema de archivos "plano" ordinario. En un ambiente de base de datos orientada a objetos, el DMC es provisto casi

completamente por la base de datos. En un ambiente de base de datos relacional o de archivo plano, el DMC debe proporcionar servicios de almacenamiento al sistema.

Hay tres formas diferentes para diseñar el DMC. Un enfoque es construir servicios de almacenamiento en cada clase y objeto en el diseño. Esto involucra, por lo general, una cantidad considerable de programación de diseño adicional.

Otra alternativa es crear una clase y objeto, servidor objetos, que proporcione todos los servicios de base de datos. Esta alternativa involucra un objeto muy complejo que sepa cómo guardar o recuperar todos los objetos del sistema. Cualquier petición de almacenamiento se hace por medio de mensajes a este único objeto.

Finalmente un tercer método es crear una clase Almacenable. Este tercer enfoque es una combinación de los dos enfoques anteriores. La clase Almacenable incluye servicios básicos *guárdame* y *recupérame* en forma generalizada. Cada objeto del sistema que deba ser guardado o recuperado es conectado luego en una estructura Gen-Spec con la clase Almacenable. Esto trabajará solamente, por lo general, en aquellos casos donde se disponga de herencia múltiple en la tecnología de implementación.

III.1.4 Diseño del SCG

A continuación se presenta la aplicación y uso de la metodología antes descrita en el programa que se ha de construir.

Para este programa cuento con la biblioteca de clases desarrollada por Distribuidores y Asesores en Informática Sistemas y Administración lugar en el que suscribe se encuentra laborando actualmente. De esta biblioteca se va a usar la clase CDaisaForm como clase base para la clase CEvalua_AtletasForm, además de las clases CBuscador, CClasif, CMensajes, Campos, CDib, CColores y CNomdep.

Para el desarrollo de este sistema se empleo el Lenguaje de Programación Visual C++ versión 4 por las siguientes razones:

Los programas ejecutables creados con este lenguaje no ocupan mucho espacio en disco, facilitando su portabilidad.

Estos ejecutables no emplean muchos recursos de maquina facilitando su ejecución en maquinas que no están muy equipadas en cuanto a procesador, memoria RAM y otros componentes se refiere.

Esto fue de gran ayuda debido a que la maquina donde se piensa instalar el sistema es una Pentium II a 450 mhz, con 64 Mb en RAM, 10Gb de disco duro, CD-ROM de 52x y un drive 3 ½.

Si al momento de compilar el código tecleado se llegan a encontrar errores de sintaxis o de algún otro tipo el Visual C++ versión 4 establece aproximadamente en que línea y archivo se encuentra dicho error, de la misma manera que ocurre en el lenguaje C, esto facilita en gran medida la depuración de los programas, además de ahorrar mucho tiempo en lo que corresponde a la búsqueda del error.

El editor de Visual C++ versión 4 en su vista de resources permite la opción de diseñar las pantallas (botones, cuadros de texto, etc.) reduciendo con esto el tiempo y el código necesarios para construirlas.

Esto además de ser un lenguaje orientado a objetos que como tal facilita la aplicación de los conceptos que maneja la metodología.

No se eligió Java puesto que no se tiene planeado usar el programa en red, ni sobre Internet, lo cual implicaría un desperdicio de las capacidades de este lenguaje.

A continuación en la figura III.2 se presenta el diagrama después de terminar el componente de dominio del problema.

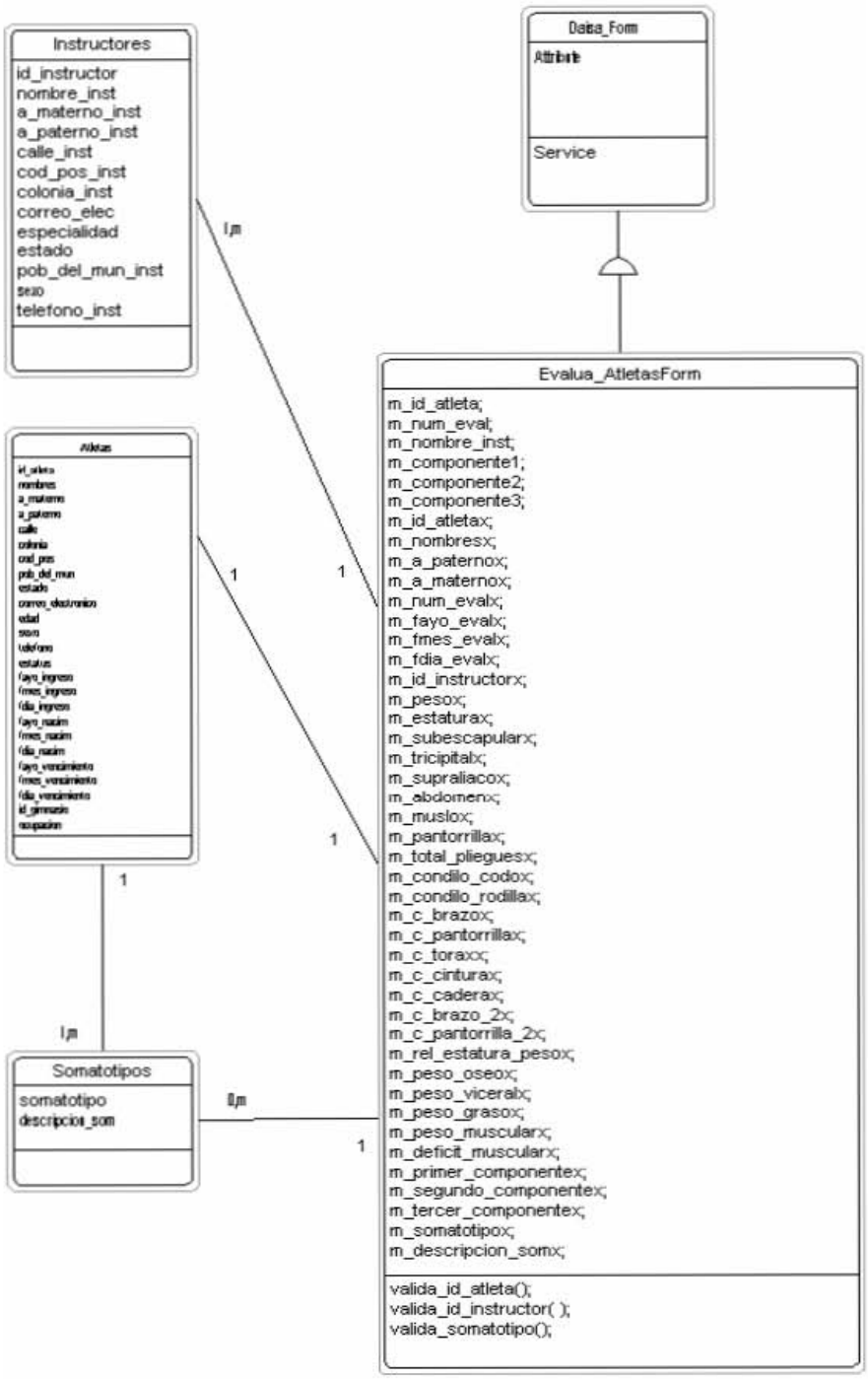


Figura III.2 Diagrama al terminar el componente de dominio del problema

Como se menciono anteriormente la vista de recursos del Visual C++ permite diseñar las pantallas que se emplearán en el sistema las cuales se muestran a continuación en las figuras III.3, III.4, III.5 y III.6.

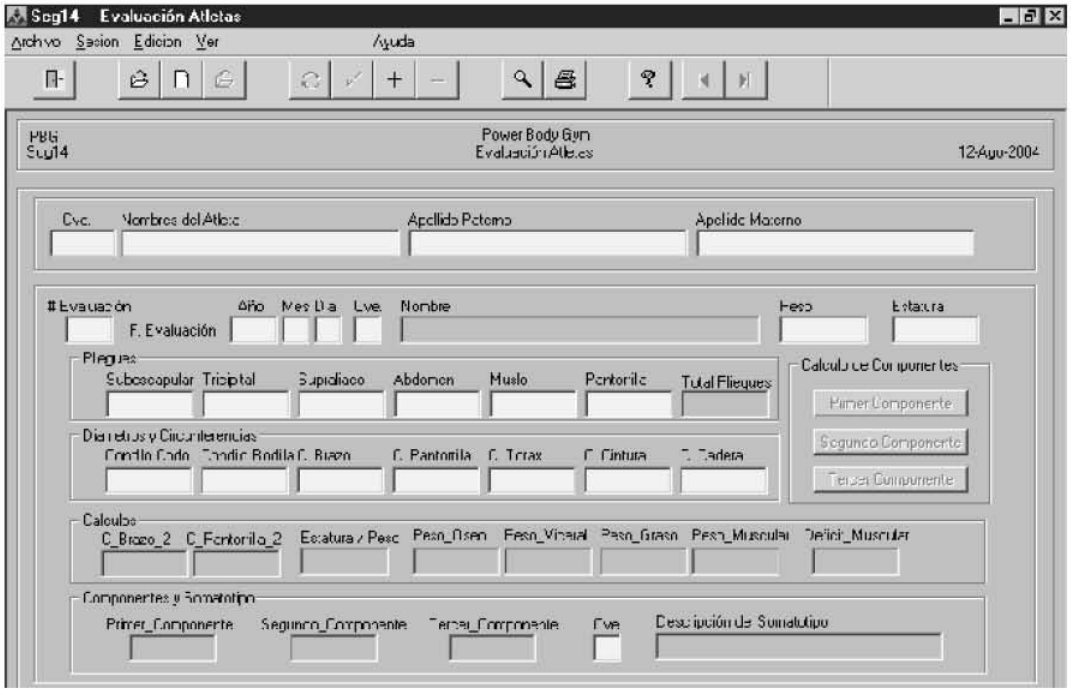


Figura III.3 Pantalla principal de captura y consulta.

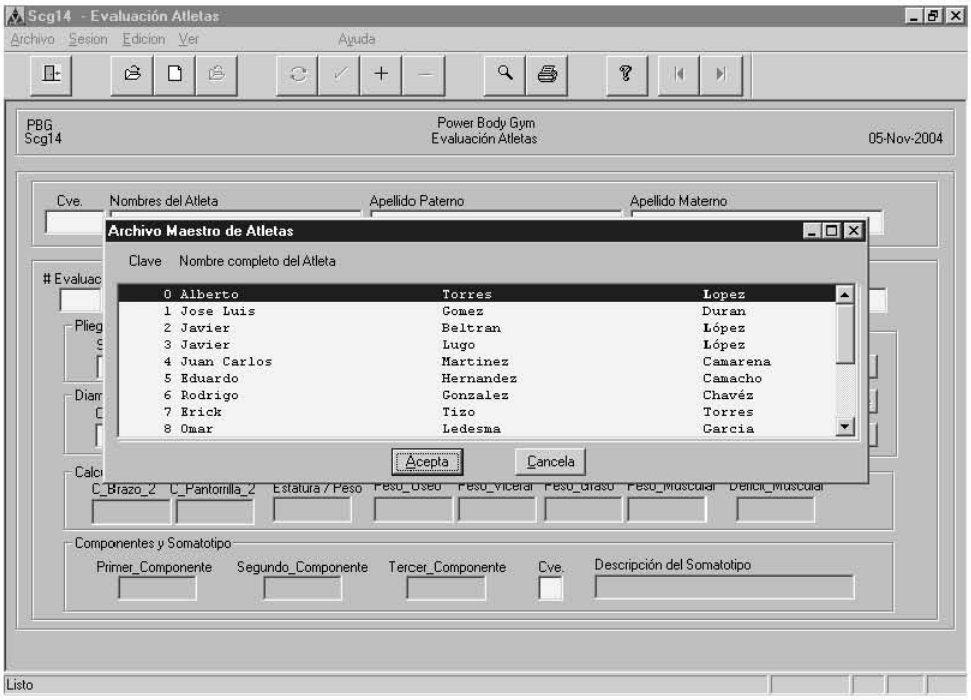


Figura III.4 Picklist de Atletas.

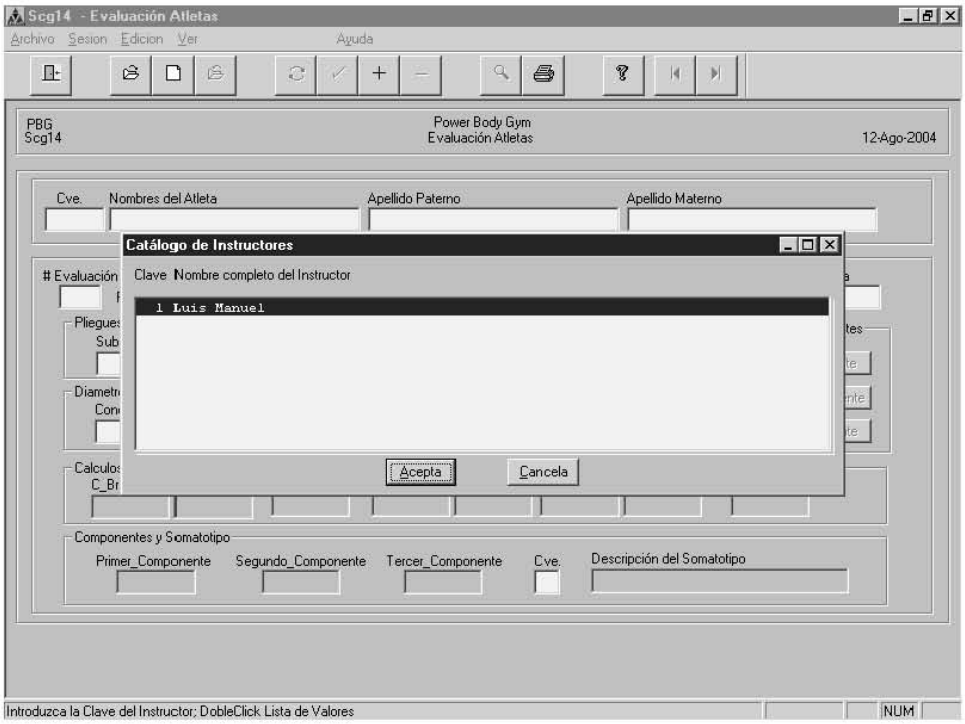


Figura III.5 Picklist de Instructores.

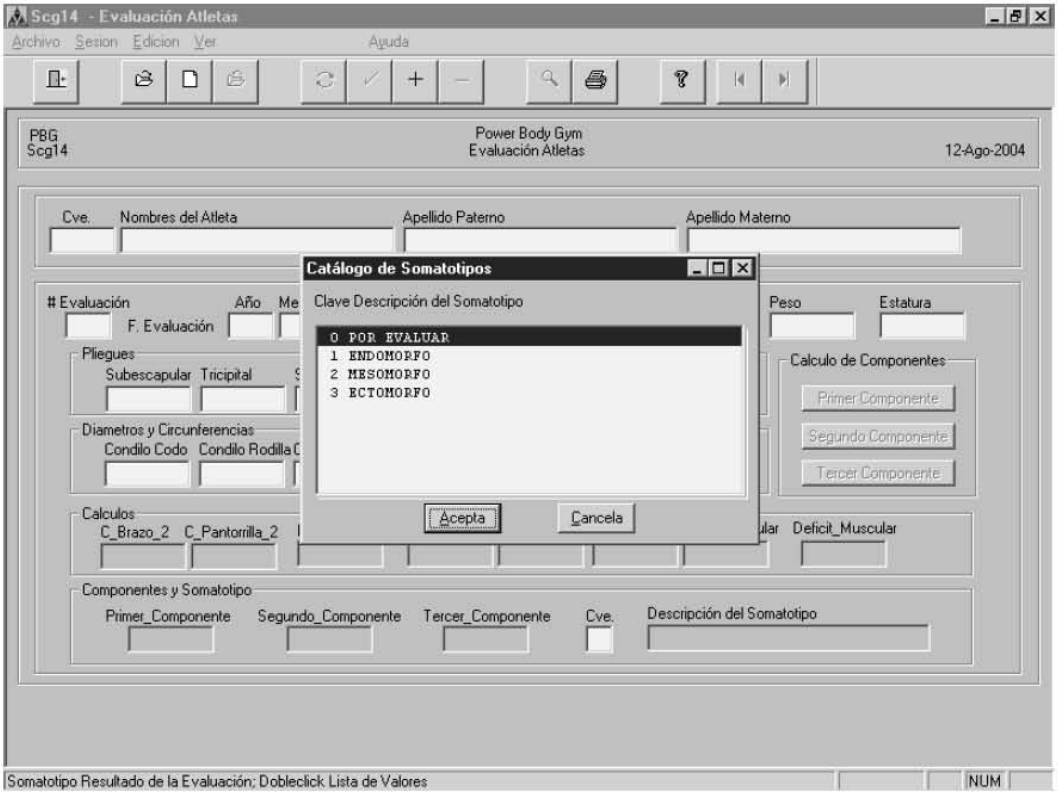


Figura III.6 Picklist de Somatotipos.

El uso de estas pantallas implica incluir cinco nuevas clases que son: CAboutDialog, CInstructoresDial, CAtletasDial, CSomatotiposDial y CMainFrame.

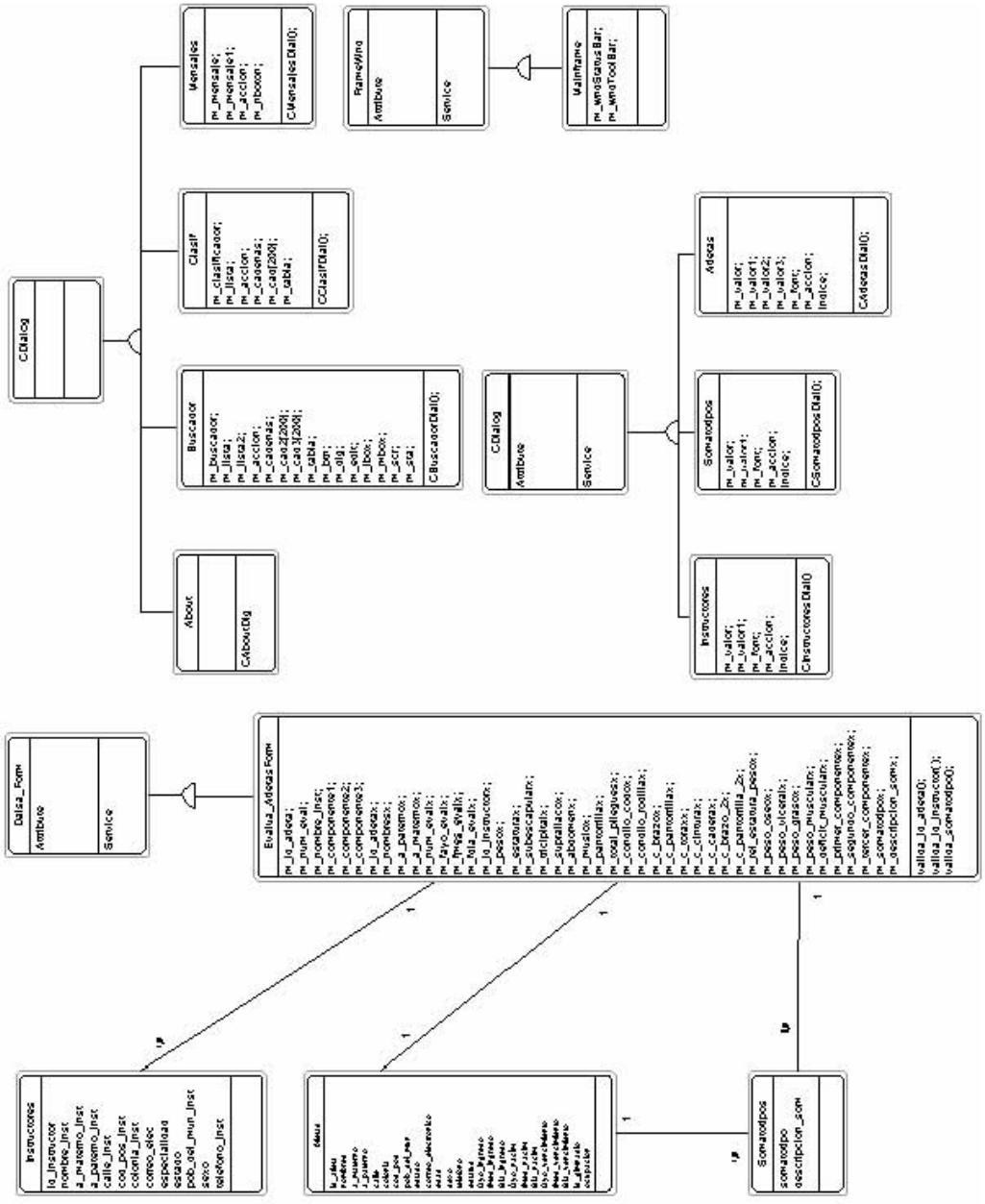


Figura III.7 diagrama despues de terminar el componente de interfaz grafica

En este programa no será necesario implementar el TMC, debido a que el sistema funcionara en una PC con un solo procesador.

Aquí se va a emplear la primera opción que es construir servicios de almacenamiento en cada clase y objeto. Esto implicara incluir 9 nuevas clases que son: CAtletasSet,

CDet_EvaluacionesSet, CSomatotiposSet, CInstructoresSet, CCuentaSet, CNomDep, CColoresSet, CScg14Doc, CScg14App.

Elegí esta opción para poder reusar estas clases en otros programas, sin necesidad de tener que hacer grandes modificaciones en ellas.

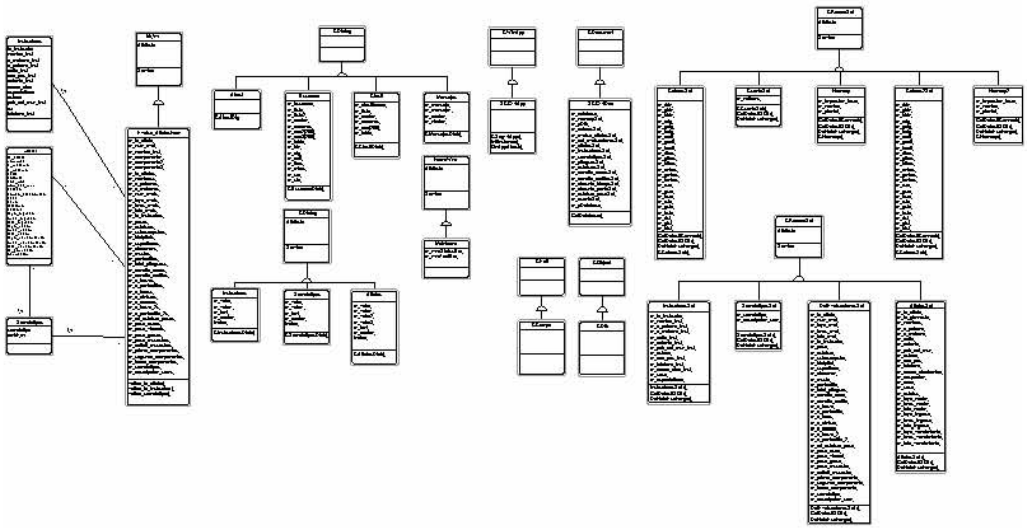


Figura III.8 Diagrama después de terminar el componente de Administración de Tareas y Datos.

III.2 Diseño de la base de datos

El almacenamiento de datos es considerado por algunos como la parte médular de los sistemas de información. Los cinco objetivos para el diseño del almacenamiento de datos son: disponibilidad de datos, integridad de los datos, almacenamiento eficiente de datos, actualización y recuperación eficiente, y recuperación de información para un propósito.

Para cumplir estos objetivos se tienen dos enfoques. El primero es guardar los datos en archivos individuales, cada uno de ellos único para una aplicación particular.

El segundo enfoque involucra la construcción de una base de datos, la cual se empleara para este sistema.

Cabe señalar que en lo que se refiere a la notación IDEF1X [BRU92], solo se van a explicar los aspectos que se emplearán, o aquellos que se consideren de suma importancia, para la comprensión del presente trabajo[1].

III.2.1 Conceptos de bases de datos

Una base de datos es un almacén de datos formalmente definido y centralmente controlado para ser usado por muchas aplicaciones diferentes. La parte médular de una base de datos es el DBMS (DataBase Management System) que permite la creación, modificación y actualización de la base de datos, la recuperación de datos y la generación de reportes. La persona que se encarga de asegurar que la base de datos satisfaga sus objetivos es el DBA (Data Base Administrator). Los objetivos de efectividad de una base de datos incluyen:

1. Asegurarse de que la base de datos pueda ser compartida entre los usuarios de una diversidad de aplicaciones.
2. Mantener datos que sean precisos y consistentes.
3. Asegurarse de que todos los datos requeridos para las aplicaciones actuales y futuras estén fácilmente disponibles.
4. Permitir que la base de datos evolucione y que las necesidades de los usuarios crezcan.
5. Permitir que los usuarios construyan su vista personal de los datos sin preocuparse de la forma en que estén físicamente guardados.

Una base de datos, como cualquier otra herramienta, tiene sus desventajas entre las que se encuentran:

1. Todos los datos están almacenados en un solo lugar por lo tanto son vulnerables a catástrofes y requieren respaldos completos.
2. Existe el riesgo de que el DBA sea el único con capacidades suficientes para acercarse a los datos.
3. Los procedimientos burocráticos requeridos para modificar, o hasta para actualizar, la base de datos pueden parecer insuperables.
4. Es difícil mantener en una cantidad tolerable el tiempo requerido para insertar, actualizar, borrar y recuperar datos.

---

[1] Si alguien esta interesado en saber más acerca de esta notación puede consultar [TERRY] o [BRU92].

5. Es alto el costo de mantenimiento de los datos.

El proceso de desarrollo de bases de datos comienza con los requerimientos de información, a partir de los cuales se siguen los siguientes pasos:

- Modelo conceptual de datos. Se analizan los requerimientos de información y se elabora un modelo lo más aproximado posible al mundo real (Modelo E/R).
- Diseño de bases de datos. Aquí se pasa el modelo E/R a tablas relacionales, se toman decisiones en cuanto a hardware y software se refiere, además de refinar el diseño inicial (normalizar).
- Construcción de la base de datos. Teclear las instrucciones (SQL) para la creación de las tablas, vistas y todo lo necesario para que la base de datos funcione.
- Y termina al tener una base de datos operacional.

En la sección II.3 Análisis para el sistema de control de gimnasios SCG del capítulo 2 se expuso la manera en que fué analizado y realizado el modelo entidad – relación (E/R) que se muestra a continuación en la figura III.9.

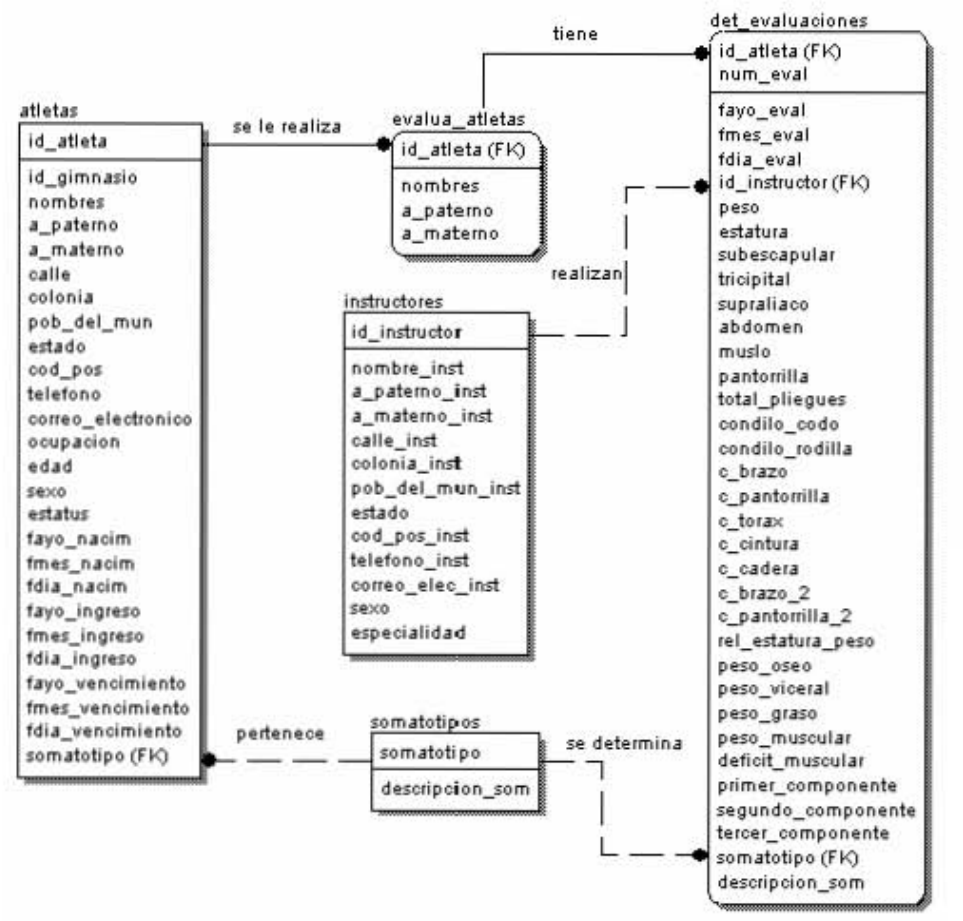


Figura III.9 Modelo Entidad Relación (E/R).

III.2.2 Conceptos básicos de la notación IDEF1X

Entidades y atributos

Una e*ntidad* es una abstracción de un objeto del mundo real. Por ejemplo, en el caso del gimnasio se podría incluir una entidad llamada ATLETAS, la cual represente la información que se necesita mantener acerca de los Atletas. Un *atributo* es una propiedad de la entidad. Por ejemplo, un atleta tiene un nombre y apellido. Así, "nombre" y "apellido" son atributos de la entidad ATLETAS.

Tanto los atributos como las entidades son abstracciones del mundo real. Una *instancia* de una entidad con sus correspondientes valores de atributos representa un objeto concreto del mundo real. Por lo tanto, se puede decir que una entidad describe un conjunto de objetos del mundo real llamados instancias.

En IDEF1X, una entidad es representada por un rectángulo con el nombre de la entidad arriba y los atributos de la entidad listados dentro del rectángulo. Los nombres de la entidad van siempre en singular y en mayúsculas. Los atributos en minúsculas.

La *clave primaria* es un atributo o grupo de atributos que han sido elegidos como un identificador único de una entidad. Una *clave candidata* es un atributo o grupo de atributos que podrían ser elegidos como clave primaria.

Una clave candidata, para ser aceptada como clave primaria, debe identificar unívocamente cada instancia de la entidad, no puede ser nula o tener alguna parte nula y todos los atributos no-clave deben depender completamente de ella y no de un subconjunto de sus atributos. Los atributos que forman la clave primaria son colocados arriba de una línea que divide la zona de la clave primaria de la zona de datos. Los atributos no-clave, se ubican debajo de la línea. A continuación en la figura III.10 se presenta un ejemplo de una entidad.

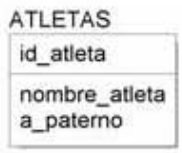


Figura III.10 ejemplo de una Entidad

Relaciones

Una relación es una asociación entre varias entidades. En la figura III.11 se muestra una relación en la que evalua asocia entidades de ATLETAS con entidades de EVALUACIONES.

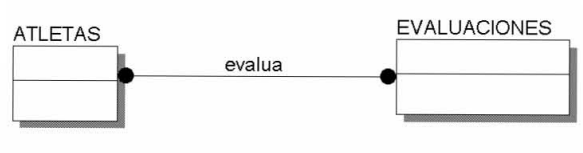


Figura III. 11 Ejemplo de una Relación

Un Diagrama Entidad Relación DER puede ser construido con distintos niveles de detalle. El diagrama de arriba no muestra detalle de atributos. Es más un modelo conceptual que un modelo de diseño. En los modelos conceptuales las relaciones muchos a muchos son permitidas y las claves, generalmente, no se encuentran presentes. Estos modelos conceptuales generalmente se usan para mostrar una visión de alto nivel del negocio y se construyen en las etapas iniciales del análisis.

Los modelos de diseño son modelos que se encuentran en 3ra forma normal. Todas las entidades, atributos, claves y relaciones se encuentran presentes. Cubre el mismo alcance que un modelo conceptual pero con un grado de detalle mucho mayor. En IDEF1X, a diferencia de otros lenguajes de modelado, todas las relaciones deben ser binarias, es decir, deben conectar exactamente dos entidades (podría ser la misma, relación reflexiva). Esto no significa que relaciones n-arias no sean necesarias, sino que ellas en IDEF1X van a ser manejadas por medio de las *entidades asociativas*.

Relaciones uno-a-muchos

En este tipo de relación, una y solo una instancia de la primera entidad está conectada a muchas instancias de la segunda entidad. La entidad del lado de "uno" es llamada la entidad padre, y la otra, entidad hijo.

Por ejemplo: "Un somatotipo puede pertenecer a muchos atletas y un atleta tiene un único somatotipo". Estas relaciones son representadas por una línea conectando las dos entidades, con un punto sobre el extremo "muchos" (correspondiente a la entidad hijo), y un nombre (generalmente un verbo) sobre la línea.

Por ejemplo, la figura III.12 muestra la relación "tiene" entre SOMATOTIPO y ATLETAS, donde un SOMATOTIPO pertenece a muchos ATLETAS y un ATLETA tiene un único SOMATOTIPO.

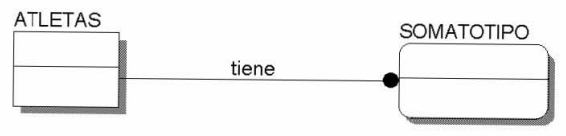


Figura III.12 Relación uno a muchos

La noción de "muchos" (denotada por el punto) no significa necesariamente que debe haber siempre más de una instancia de la entidad hijo conectada a la entidad padre. A menos que se indique explícitamente otra cosa, esto significa que puede haber *cero, una o más* instancias de la entidad hijo asociadas a la entidad padre. El número de instancias esperado en cada extremo de la relación es llamado *cardinalidad*. Se usan distintos símbolos en el extremo donde está el punto para indicar distintas cardinalidades:

- "P" indica uno o más.
- "Z" indica cero o una.
- "N" indica exactamente N.
- Ausencia de símbolo indica cero o más.

Por ejemplo, la relación "un equipo está formado por 22 jugadores" sería representada por el diagrama mostrado en la figura III.13, donde el 22 sobre el punto indica claramente que cada instancia de un EQUIPO debe estar relacionada con exactamente 22 instancias de JUGADOR.

Puede haber una o cero instancias en el extremo del padre (que no tiene el punto). En el ejemplo superior de la figura III.13, de acuerdo a la notación empleada, siempre deberá haber una y sólo una instancia. Mas adelante se vera como se puede expresar la posibilidad de que no exista ninguna.

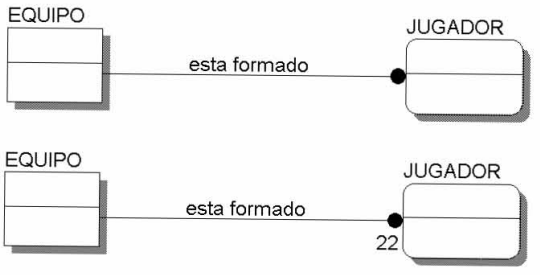


Figura III.13 Ejemplos de cardinalidades

Relaciones muchos-a-muchos

Cuando existen relaciones uno-a-muchos en ambas direcciones, estamos frente a una relación *muchos-a-muchos*. Aquí la clasificación padre-hijo pierde sentido. Estas relaciones, también conocidas como *no-específicas*, son dibujadas, en una etapa preliminar del diagrama, como una línea con un punto en ambos extremos.

La figura III.14 muestra un ejemplo de una relación muchos-a-muchos, donde una persona alquila muchas copias de películas y una copia es alquilada a lo largo del tiempo por muchas personas.

Figura III.14 Relación muchos a muchos

Una meta de IDEF1X es eliminar todas las relaciones muchos-a-muchos, convirtiéndola en un par de relaciones uno-a-muchos. Para ello, se vale del uso de una entidad asociativa en medio de las otras dos entidades, como se muestra a continuación en la figura III.15.

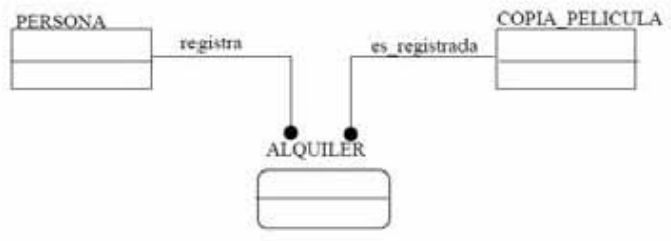


Figura III.15 Ejemplo de Entidad Asociativa

De esta forma, se puede decir que una persona registra muchos alquileres y cada instancia de un ALQUILER corresponde a una única persona. Por otro lado, una copia de una película es registrada por muchos alquileres y cada instancia de ALQUILER registra el alquiler de una única copia.

Relaciones identificables y no-identificables - migración de la clave

En IDEF1X, existen dos tipos de relaciones: *identificables* y *no-identificables*. Una relación *identificable* causa que la clave primaria de la entidad padre pase a ser parte de la clave primaria de la entidad hijo. Estas relaciones son indicadas por una línea llena conectando las dos entidades,

con un punto en el extremo "muchos". La figura III.16 muestra un ejemplo de una relación identificable, donde se puede ver cómo la clave primaria del padre migra hacia el área de clave primaria del hijo.

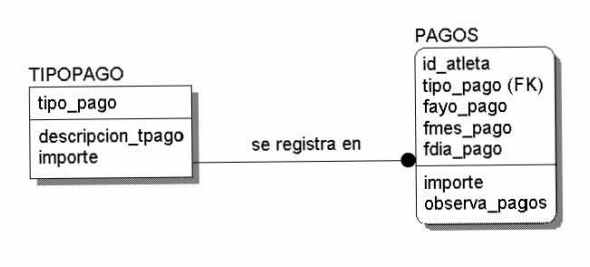


Figura III.16 Ejemplo de relación identificable

Una relación identificable lleva consigo una regla del negocio que dice que se ha hecho una elección intencional para incluir la clave primaria de la entidad padre como parte de la clave de la entidad hijo. En el ejemplo entre TIPO_PAGO y PAGOS, se pudo haber hecho otra elección: identificar el pago con un número único propio; en lugar de eso, se decidió armar la clave de pagos usando la clave de TIPO_PAGO y agregando una segunda parte ("id_atleta") así como la fecha para diferenciar un pago de otro. Es decir, en este caso, se tomo la decisión de que la relación sería identificable.

Cuando se usa una relación identificable, la entidad hijo pasa a ser una entidad *dependiente*, ya que depende ahora del padre para su identidad: el hijo no puede ser identificado si no se conoce la clave del padre. En las relaciones identificables, no sólo la identidad del hijo depende del padre, sino que también su existencia depende del padre, ya que si el padre no existe, parte de la clave falta y por lo tanto el hijo no puede existir. Es por esto último que las relaciones identificables son siempre *obligatorias*: el hijo no existe si no está relacionado con un padre.

Por otro lado, están las relaciones *no-identificables*. Una relación *no-identificable* causa que la clave primaria de la entidad padre pase a ser parte de los atributos no-clave de la entidad hijo. Se indican por una línea punteada conectando las dos entidades, con un punto en el extremo "muchos". Dado que, en una relación no-identificable, la clave primaria del padre no migra a la clave del hijo, el hijo no es identificado por el padre, y la relación puede, desde la perspectiva del hijo, ser *opcional* o, lo que es lo mismo, una instancia de la entidad hijo puede encontrarse no relacionada con instancia alguna de la entidad padre.

En las relaciones no-identificables, la existencia del hijo puede aún depender del padre, aunque su identidad no dependa de él. Si la relación es obligatoria desde la perspectiva del hijo, entonces la existencia del hijo depende del padre. Si es opcional, la existencia del hijo no

depende del padre. Las relaciones opcionales son denotadas con un rombo blanco en el extremo correspondiente al padre.

A continuación en la figura III.17 se muestra un ejemplo de una relación no-identificable y obligatoria. Es no-identificable porque la identidad de una instancia de CURSOS no depende de la identidad de un instructor, ya que su clave no está formada por la clave de INSTRUCTORES; y además, es obligatoria porque la existencia de un curso depende de la existencia de un instructor (no hay curso sin instructor).

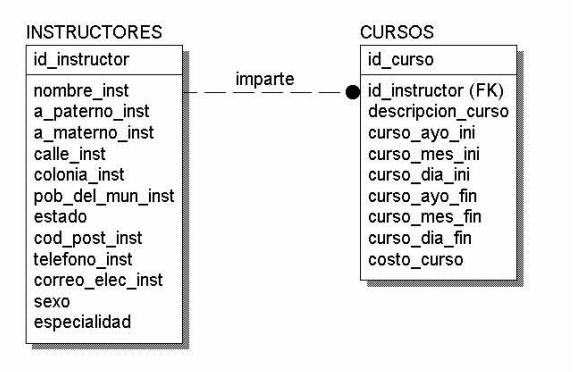


Figura III.17 Ejemplo de relación no identificable y obligatoria

Por otro lado la figura III.18 muestra una relación no-identificable opcional. La identidad de INSTRUCTORES no depende de ESTADOS así como tampoco su existencia. En efecto, la clave de INSTRUCTORES no depende de la clave de ESTADOS y se pueden tener instancias de INSTRUCTORES que no estén relacionadas con ninguna de ESTADOS; esto último significa que el atributo "estado" que está en el área de datos de INSTRUCTORES puede contener un valor nulo.

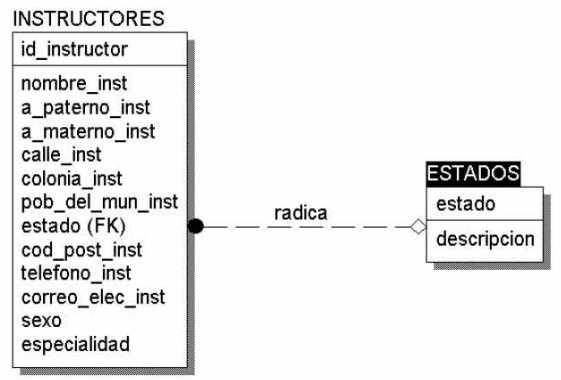


Figura III.18 Ejemplo de relación no identificable opcional

Finalmente en la figura III.19 se presenta un cuadro resumiendo las posibles combinaciones para relaciones identificables y no-identificables.

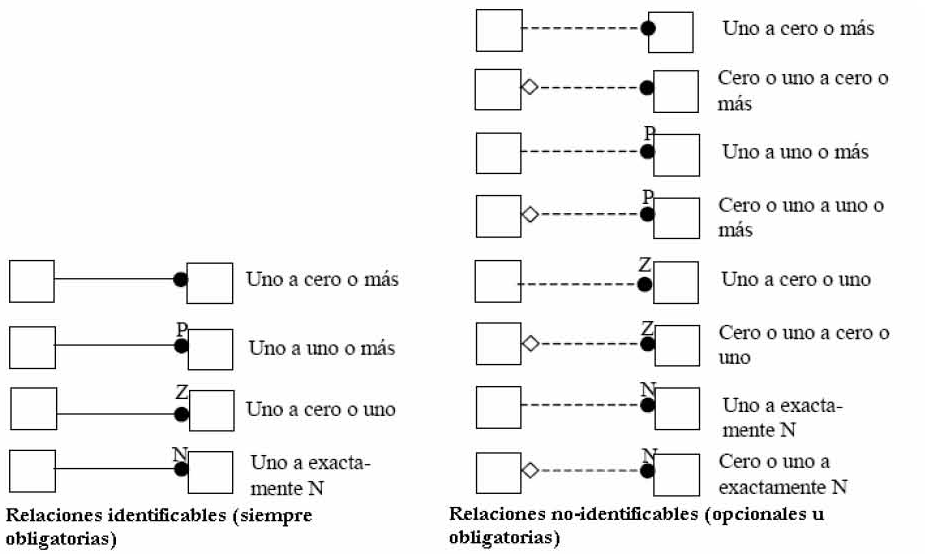


Figura III.19 Combinaciones para relaciones identificables y no-identificables

Entidades dependientes e independientes

Las entidades son designadas como *dependientes* o *independientes* de acuerdo a cómo adquieran sus claves. Las entidades *independientes* no dependen de otra entidad para su identificación. Las *dependientes* dependen de una o más entidades para su identificación. Las entidades independientes son representadas por rectángulos y las dependientes por rectángulos con los ángulos redondeados.

Figura III.20 Ejemplo de Entidad Dependiente e Independiente

En la figura III.20 se muestra un ejemplo de entidad dependiente (CLAVES_ACCESO) y un ejemplo de entidad independiente (PROGRAMAS). Un caso especial de entidad dependiente son las entidades asociativas, las cuales arman su clave a partir de las claves primarias de dos o más entidades. Un ejemplo es la entidad DET_DIETAS, mostrada en la figura III.21.

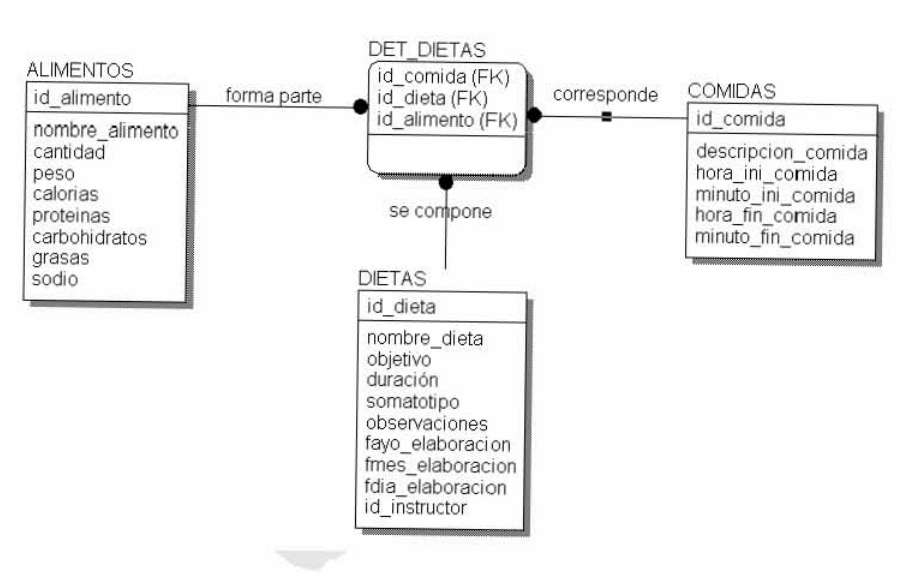


Figura III.21 Ejemplo de una entidad asociativa

Después de ver estos conceptos básicos de la notación IDEF1X. Se continuara con las actividades de diseño de base de datos las cuales incluyen:

1. Mapeo.
2. Refinación del modelo inicial.
3. Toma de decisiones en cuanto a hardware y software.

Para esto se utilizara el modelo relacional publicado por el Dr. Edgar F. Codd [COD70] explicado por Amparo López Gaona [AMP00]

III.2.3 El modelo relacional

Después de terminar el modelo E/R hay que traducir ese esquema lógico conceptual en el modelo de datos formal requerido por el Sistema Administrador de Bases de Datos (SABD). Aquí es donde entra el modelo relacional el cual esta basado en la teoría de conjuntos matemáticos.

Estructura

Una base de datos relacional (BDR) es una base de datos cuya estructura lógica se construye como una colección de relaciones, es decir, los datos se almacenan en tablas bidimensionales denominadas relaciones. Cada relación tiene un número fijo de columnas denominadas atributos y una cantidad, dinámica (dependiente del tiempo) de renglones denominados tuplas. La definición especifica el contenido de cada columna de la tabla pero no

incluye los datos. Cada renglón representa una relación entre un conjunto de valores. Como se muestra en la figura III.22.

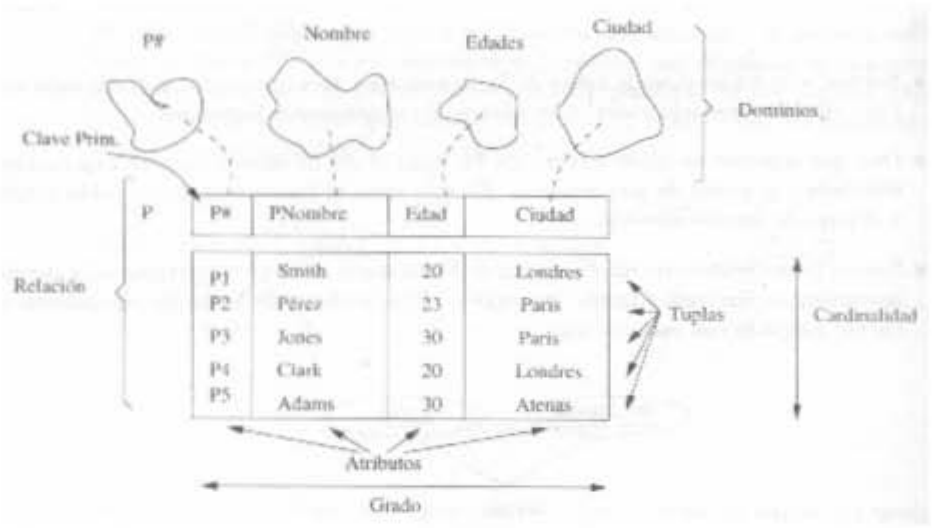


Figura III.22 Componentes de una Base de Datos Relacional

Cada atributo debe definirse exactamente sobre un dominio.

Dominio = conjunto de valores escalares (atómicos) de los cuales los atributos específicos toman sus valores. Varios atributos pueden tener el mismo dominio (cadena de caracteres). La importancia de los dominios es que evitan comparaciones sin sentido entre datos que pertenecen a distintos dominios. Este es un aspecto conceptual, es decir el dominio no se almacena en la BD, solo se especifica en la definición de BD y luego cada definición de atributo deberá incluir una referencia al dominio correspondiente. Un valor que es miembro de todos los dominios posibles es el valor nulo, que indica que el valor es desconocido o no existe.

De manera más formal una relación R, sobre un conjunto de dominios D_1, D_2,…., Dn no necesariamente distintos consta de 2 partes: encabezado y cuerpo.

- El encabezado consta de un conjunto fijo de atributos o mas precisamente parejas <nombre-atributo, nombre-dominio> (< A_i, D_i >, i = 1, 2,…, n, con $A_i \neq A_j$ para todo i, j). Este encabezado se conoce también como esquema de la relación.
- El cuerpo o ejemplar, consta de un conjunto de tuplas, cada una definida como un conjunto de parejas. <nombre-atributo, valor-atributo > (< A_i, v_{ji} >, con i = 1,…, n, y j = 1,…, m) con m = cantidad de tuplas en el conjunto (cardinalidad) y n = grado. Una tupla es ejemplar de la relación y forma parte de la extensión de la relación.

El hecho de que el modelo relacional defina una relación como un conjunto de tuplas tiene las siguientes consecuencias:

- **No hay tuplas duplicadas.** Lo cual tiene como consecuencia que exista siempre una clave primaria, por lo menos la combinación de todos los valores de los atributos.
- **El orden de las tuplas es irrelevante.** Aunque al almacenarlas, dibujarlas y/o desplegarlas tengan cierto orden, ése no forma parte de la definición de la relación, debido a que la relación intenta representar el hecho a nivel abstracto. Se pueden especificar muchas formas lógicas para ordenar los datos. Por ejemplo nombre, estado, etc. Ambas relaciones se consideran idénticas.
- **Los atributos están desordenados.** Estos se referencian por nombre no por posición, no existe el concepto del n-ésimo atributo.
- **Todos los atributos tienen valores atómicos.** Debido a los dominios. En cada lugar de la tabla (renglón, columna) hay siempre un valor no una colección de valores. lo importante no es lo que sea el dominio sino la manera en que se utilizan sus elementos.

Enseguida se presenta la forma de realizar la conversión de un esquema E/R a tablas.

- Las claves primarias permiten que los conjuntos de entidades y los de relaciones se expresen de manera uniforme como tablas que representan el contenido de la BD.
- Una BD que conforma un diagrama E/R puede representarse como una colección de tablas.
- Para cada conjunto de entidades y cada conjunto de relaciones existe una tabla única con el nombre de tal conjunto.
- Cada tabla tiene un número de columnas, generalmente corresponden a los atributos. Por tanto deben tener nombres únicos.
- Convertir un diagrama E/R a formato de tablas es la base para derivar un diseño de BDR de un diagrama E/R.

Una entidad fuerte se convierte en una tabla con el mismo nombre y con los mismos atributos.

Sean D una entidad débil con atributos a_1, a_2,…, a_n y F la entidad fuerte de la que depende D con clave primaria formada por los atributos b_1, b_2,…, b_m. Esta entidad débil se representa mediante una tabla denominada D con columnas para cada atributo del conjunto $\{a_1, a_2,\ldots, a_n\} \cup \{b_1, b_2,\ldots, b_m\}$

Representación de relaciones como tablas

Sean:

R una relación,

$\{a_1, a_2, \ldots, a_n\}$ el conjunto de atributos formados por la unión de las claves primarias de cada una de las entidades que participan en R, y

$b_1, b_2, \ldots, b_m$ los atributos de R (si los hay).

La relación se representa mediante una tabla llamada R con una columna por cada atributo del conjunto: $\{a_1, a_2, \ldots, a_n\} \cup \{b_1, b_2, \ldots, b_m\}$

Una relación que asocia una entidad débil a una fuerte. Estas son muchos a uno y no tienen atributos descriptivos. Además la clave primaria de una entidad débil incluye la clave primaria de la entidad fuerte.

Modelado de restricciones

En la definición de una BD es necesario establecer *reglas de integridad* con objeto de informar al Sistema Administrador de Bases de Datos de ciertas restricciones en el mundo real y por tanto prevenir tales configuraciones imposibles. Estas pueden ser tan arbitrarias como se quiera sin embargo esto ocasionará que se complique la verificación de las mismas, lo habitual es limitarse a reglas que se puedan verificar con un mínimo de sobrecarga como las que se presentan a continuación.

- **Claves (llaves primarias PK).** Atributo(s) que identifica(n) de manera única a cada tupla. No existen dos tuplas con valor igual en él (llos).

  Integridad de entidad: no existe clave con valor nulo.

  Los otros atributos pueden tener a lo más un valor.

- **Claves externas (llaves foráneas FK).** Cuando una clave C de una relación R1 es atributo A de otra relación R2 se dice que A es una clave externa. El valor de la clave externa es una referencia a la tupla que contiene el valor de la clave primaria. En la figura III.23 se muestra un ejemplo.

Tabla	DET_EVALUACIONES							
Columna	id_atleta	num_eval	fayo_eval	fmes_eval	fdia_eval	id_instructor	peso	estatura
Tipo de Llave	PK	PK				FK		
Nulos/Unicos	NN, U	NN	NN	NN	NN	NN	NN	NN
Ejemplos								

Figura III.23 Ejemplo de una clave externa

Por definición se requiere que cada valor de una clave externa aparezca como valor de su correspondiente clave primaria. Sin embargo, la inversa no es un requisito, esto es, la clave

candidata correspondiente a alguna clave externa dada puede contener un valor que no aparezca como el valor de una clave externa, además pueden repetirse.

- Integridad referencial: La BD no debe tener valores para A que no aparezcan en C aunque el contrario no es requisito, es decir en C puede haber valores que no existan en A. Esto significa que si B referencia a A entonces debe existir A.

Por ejemplo el valor de id_instructor en DET_EVALUACIONES debe aparecer en INSTRUCTORES, sin embargo, puede haber un id_instructor en INSTRUCTORES que no aparezca en DET_EVALUACIONES.

- **Restricciones de Dominio.** Todo atributo debe tomar un valor atómico de un dominio.
- **Restricciones particulares** de cada aplicación. Por ejemplo que la edad esté entre 18 y 60.

Normalización

La normalización es la transformación de vistas de usuario complejas y almacenes de datos a un conjunto de estructuras de datos estables más pequeñas. Además de ser más simples y más estables, las estructuras de datos normalizadas son más fáciles de mantener. [KEN00]

Es muy probable que la relación derivada de la vista de usuario o del almacén de datos esté no normalizada.

Existen 3 formas normales:

Primera Forma Normal. (1NF) La primera etapa del proceso es eliminar los grupos de repetición y la identificación de la llave primaria. Para hacer esto, la relación necesita ser dividida en dos o más relaciones. En este momento, las relaciones ya pueden estar en la tercera forma normal, aunque es muy probable que se necesiten más pasos para transformar las relaciones a la tercera forma normal.

Segunda Forma Normal. (2NF) El segundo paso asegura que todos los atributos que no son llave sean completamente dependientes de la llave primaria. Todas las dependencias parciales son eliminadas y puestas en otra relación.

Tercera Forma Normal. (3NF) El tercer paso elimina cualquier dependencia transitiva. Una dependencia transitiva es aquella en la cual atributos que no son llave son dependientes de otros que tampoco son llaves.

Con la experiencia que se va adquiriendo en el campo de trabajo, se puede ir reduciendo desde un inicio la redundancia. Al ir identificando las posibles tablas junto con sus llaves primarias. Aunque nadie es perfecto y es casi seguro que se comentan errores razón por la cual no se debe omitir nunca este proceso (Normalización).

III.2.4 Diseño de la base de datos para el SCG

A continuación se presenta el detalle del diseño de la base de datos para el caso particular del programa SCG14.

Se esta modelando una realidad donde se tienen atletas y evaluaciones las cuales son practicadas a los atletas. "evalua" es una relación que asocia los atletas con las evaluaciones. Esto es representado como una *relación* entre el concepto abstracto ATLETAS y el concepto abstracto EVALUACIONES. Dependiendo del propósito, se puede especificar esta realidad, solamente con dos entidades (ATLETAS y EVALUACIONES) y una relación (evalua), como muestra la figura III.24. Los puntos sobre la relación indican que "evalua" es una relación muchos-a-muchos, es decir, que una instancia de ATLETAS puede estar relacionada a través de la relación "evalua" con cero ó más instancias de EVALUACIONES, y que una instancia de EVALUACIONES puede encontrarse relacionada con cero o más instancias de ATLETAS.

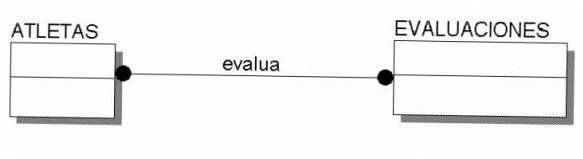


Figura III.24 Relación entre Atletas y Evaluaciones.

Representación de entidades como tablas

En esta sección se tomará el modelo Entidad – Relación (E/R) del programa SCG14. Es decir, el modelo E/R incluyendo solamente las tablas que empleará el programa SCG14 (evaluación de atletas) para ejemplificar el paso del modelo E/R a tablas. A esta acción también se le denomina "mapeo".

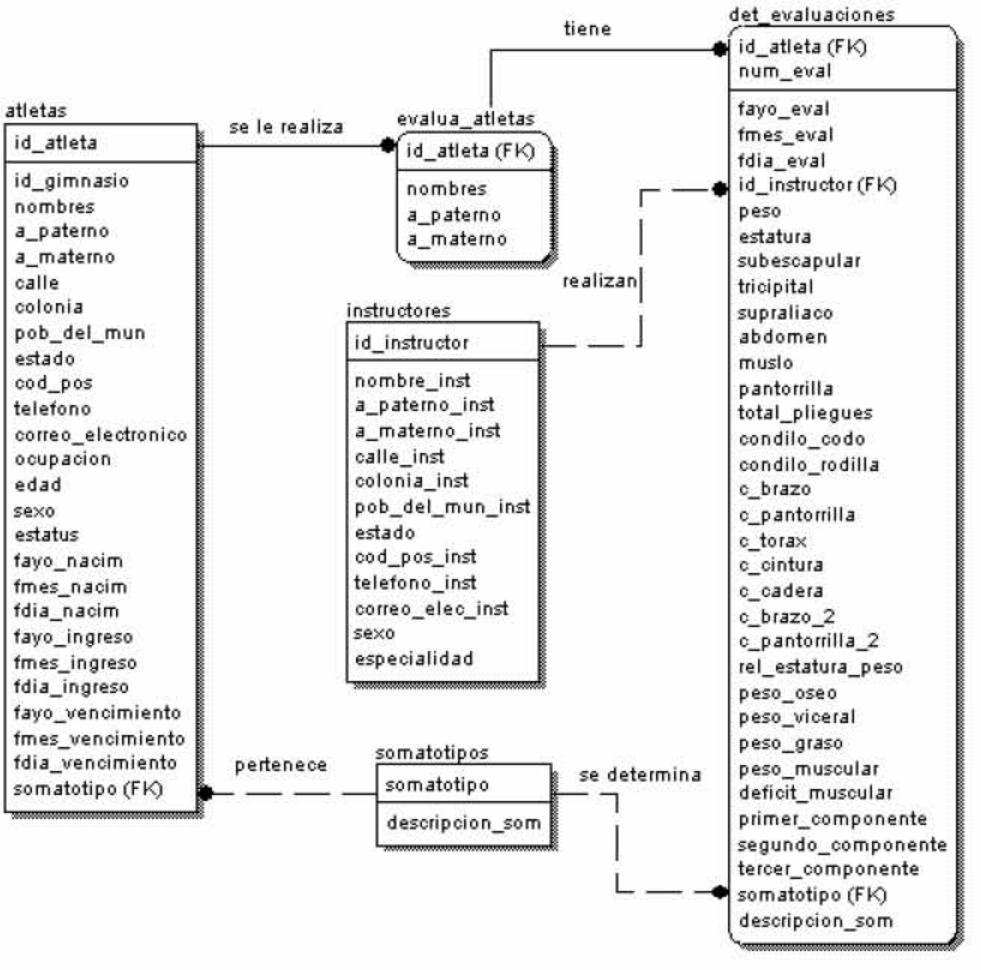


Figura III.25 Modelo E/R del programa SCG14.

La entidad fuerte es SOMATOTIPO donde los atributos son somatotipo y descripcion_som.

SOMATOTIPOS

Columna	somatotipo	descripcion_som
Tipo de Llave		
Nulos/Unicos		
Ejemplos		

Figura III.26 Representación de la entidad SOMATOTIPO como la tabla SOMATOTIPOS.

La entidad débil en este caso es EVALUA_ATLETAS y ATLETAS es la entidad fuerte donde el atributo b_1 es id_atleta los atributos a_1, a_2 y a_3 son nombres, a_paterno y a_materno.

Tabla	EVALUA_ATLETAS			
Columna	id_atleta	nombres	a_paterno	a_materno
Tipo de Llave				
Nulos/Unicos				
Ejemplos				

Figura III.27 Representación de la entidad EVALUA_ATLETA como la tabla EVALUA_ATLETAS.

En este caso se empleara la relación DET_DIETA mostrada en la figura III.23

DET_DIETAS

Columna	id_comida	id_dieta	id_alimento
Tipo de Llave			
Nulos/Unicos			
Ejemplos			

Figura III.28 Representación de la relación DET_DIETA en la tabla DET_DIETAS.

En general la tabla para esta relación es redundante y por tanto no necesita representarse de esta manera.

El diseño de las tablas para el programa SCG14 (evaluacion_atletas) se presenta a continuación:

Tabla ATLETAS

Columna	id_atleta	id_gimnasio	nombres	a_paterno	a_materno
Tipo de Llave	PK	FK1			
Nulos/Unicos	NNU	NN	NN	NN	
Ejemplos	1	1	JUAN	LOMELI	SOSA

calle	colonia	pob_del_mun	estado
			FK2
			NN
EJE 2 MNZ 2 L-1 C-3	LOMAS DE CARTAGENA	TULTITLAN	15

cod_pos	telefono	correo_electronico	ocupacion	edad	sexo	estatus
					FK3	FK4
					NN	NN
54958		wolf@hotmail.com	estudiante	17	M	A

fayo_nacim	fmes_nacim	fdia_nacim	fayo_ingreso	fmes_ingreso	fdia_ingreso
			NN	NN	NN
1987	7	15	2004	10	13

fayo_vencimiento	fmes_vencimiento	fdia_vencimiento	somatotipo
			FK5
NN	NN	NN	NN
2004	11	13	1

Figura III.29 Tabla ATLETAS.

Tabla EVALUA_ATLETAS

Columna	id_atleta	nombres	a_paterno	a_materno
Tipo de Llave	PK			
Nulos/Unicos	NNU	NN	NN	
Ejemplos	1	JUAN	LOMELI	SOSA

Figura III.30 Tabla EVALUA_ATLETAS

Tabla DET_EVALUACIONES

Columna	id_atleta	num_eval	fayo_eval	fmes_eval	fdia_eval
Tipo de Llave	PK, FK1	PK			
Nulos/Unicos	NN, U1	NN, U2	NN	NN	NN
Ejemplos	1	1	2004	10	13

id_instructor	peso	estatura	subescapular	tricipital	supraliaco
FK2					
NN	NN	NN	NN	NN	NN
1	75	1.7	30	20	10

abdomen	muslo	pantorrilla	total_pliegues	condilo_codo	condilo_rodilla
NN	NN	NN	NN	NN	NN
15	25	10	60	5	10

c_brazo	c_pantorrilla	c_torax	c_cintura	c_cadera	c_brazo_2
NN	NN	NN	NN	NN	NN
60	30	115	75	90	58

c_pantorrilla_2	rel_estatura_peso	peso_oseo	peso_viceral	peso_graso
NN	NN	NN	NN	NN
29	13	10.53	18	0.066

peso_muscular	deficit_muscular	primer_componente	segundo_componente
NN	NN	NN	NN
24.15	17.78		

tercer_componente	somatotipo	descripcion_som
	FK3	
NN	NN	NN
	1	ENDOMORFO

Figura III.31 Tabla DET_EVALUACIONES.

Tabla INSTRUCTORES

Columna	id_instructor	nombre_inst	a_paterno_inst	a_materno_inst
Tipo de Llave	PK			
Nulos/Unicos	NNU	NN	NN	NN
Ejemplos	1	JOAQUIN	ORTEGA	PINEDA

calle_inst	colonia_inst	pob_del_mun_inst	estado
			FK1
			NN
EJE 2 MNZ 2 L-1 C-3	LOMAS DE CARTAGENA	TULTITLÁN	15

cod_post_inst	telefono_inst	correo_elec_inst	sexo	especialidad
			FK2	
NN			NN	
54958		wolf@hotmail.com	M	TECNICO

Figura III.32 Tabla INSTRUCTORES

Tabla SOMATOTIPOS

Columna	somatotipo	descripcion_som
Tipo de Llave	PK	
Nulos/Unicos	NNU	NN
Ejemplos	1	ENDOMORFO

Figura III.33 Tabla SOMATOTIPOS

En lo que respecta a las tablas del programa SCG14 usando los diagramas mostrados en las figuras III.29, III.30, III.31, III.32 y III.33 se aprecia que:

- En ninguna de las tablas se tienen grupos de repetición por tanto se cumple con la primera forma normal 1NF.
- En todas las tablas los atributos son funcionalmente dependientes de la llave primaria con lo cual, cumplen con la segunda forma normal 2NF.
- El siguiente paso es revisar que no existan dependencias funcionales transitivas. Esto es que alguno de los atributos que no es parte de la llave, dependa además de la llave primaria de otro atributo que tampoco forma parte de la llave.
- En ninguna tabla se tienen dependencias transitivas así que se puede decir que se cumple con la tercera forma normal 3NF.

Con esto se puede decir que se concluyo el 2ndo. Hito técnico: Diseño OO del programa scg14 terminado. Cuando menos en su primera iteración, ya que, como se ha mencionado anteriormente, este es un proceso iterativo, razón por la cual, no se puede considerar totalmente terminado el diseño, sin tomar en cuenta los cambios que sufrirá el scg14 a lo largo del desarrollo, el cual se aborda en el siguiente capítulo de este trabajo.

Capítulo IV Desarrollo del Sistema

La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica.

Aristóteles

IV.1 Desarrollo de las Clases

Llego el momento de empezar con el desarrollo del SCG, como antes se utilizara como ejemplo el programa SCG14 Evaluación de los Atletas. Para mostrar como se aplico lo descrito en los capítulos anteriores análisis (Capítulo 2) y Diseño (Capítulo 3) con el lenguaje de programación orientado a objetos Visual C++ V 4.0 y como base de datos se empleo ACCESS.

Se eligió ACCESS® como base de datos sobre otras como ORACLE®, SQL SERVER®, DB2® y demás por que no se podía obligar al usuario a realizar un gasto tan grande como lo es adquirir una licencia de ORACLE®, SQL SERVER® u otras. Además por que el usuario ya tiene el ACCESS® instalado en su equipo, puesto que estaba incluido en la versión de OFFICCE® que le instalaron al momento de adquirir su equipo.

IV.1.1 Estructura de los Programas

Antes que nada se dará una descripción de como está distribuido cada programa de este sistema. El código está guardado en dos archivos: en el primero (*.h) esta la definición de la

clase, de todas las variables y de las estructuras que empleará dicha clase a lo largo del programa. En el segundo archivo (*.cpp) están guardadas las funciones que se emplean en la implementación y el funcionamiento de cada una de las clases definidas y guardadas en los archivos (*.h).

A su vez estas clases están guardadas en los siguientes archivos: X.cpp, Xdoc.cpp, Xf01.cpp, Xfrm.cpp, Xp01.cpp, Xs01.cpp X.rc, X.h, Xdoc.h, Xf01.h, Xfrm.h, Xp01.h, Xs01.h y X.rc2. En base a la función que desempeña cada clase dentro del sistema, esto es:

- El archivo X.* incluye la clase CScg14App que se encargará de arrancar el programa.
- En el archivo Xdoc.* se incluye la clase CScg14Doc que se encargará de hacer el enlace con la base de datos.
- El archivo Xf01.* contiene la clase CEvalua_atletasForm encargada de manejar gran parte de la funcionalidad del programa como llamar las funciones y la interacción entre la mayoría de las clases. Además de ser la interfaz con el usuario.
- Las clases que se encuentran en los archivos Xfrm.* se encargan de controlar los botones de la barra de herramientas, la barra de auto scroll y la caja de mensajes que dice "El registro ha sido modificado; ¿Desea abandonar los cambios efectuados?".
- Las clases definidas en el archivo Xp01.* son las que se utilizan para los picklist (las cajas que presentan las claves existentes con la opción de seleccionar la que se requiera además de en algunos casos obtener algunos de los datos mostrados).
- Las que se encuentran en el archivo Xs01.* son las encargadas de obtener y manipular la información guardada en las tablas de la base de datos.
- Estos además de los archivos DAISA.h y DAISA.cpp[1].

A continuación se presentarán algunas de las clases que se emplearon, para lo cual se utilizarán extractos de código. Comenzando con las figuras IV.1 y IV.2 que presentan la clase CScg14App.

[1] Bibliotecas de funciones propiedad de Distribuidores y Asesores en Informática Sistemas y Administración (DAISA) que es una empresa que brinda servicios de consultoría, venta de equipo, desarrollo de aplicaciones, entre otros.

```
// scg14.h : main header file for the Scg14 application
//
/////////////////////////////////////////////////////////////////////////////
extern "C" extern void WINAPI InitDaisaDLL();

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif
#include "scg14res.h"       // main symbols
/////////////////////////////////////////////////////////////////////////////
// CScg14App:
// See Scg14.cpp for the implementation of this class
//

class CScg14App : public CWinApp
{
public:
    CScg14App();

// Overrides
    virtual BOOL InitInstance();

// Implementation
    //{{AFX_MSG(CScg14App)
    afx_msg void OnAppAbout();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

Figura IV.1 Archivo SCG14.H

```
/////////////////////////////////////////////////////////////////////////////
// CScg14App
BEGIN_MESSAGE_MAP(CScg14App, CWinApp)
    //{{AFX_MSG_MAP(CScg14App)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
    //}}AFX_MSG_MAP
    ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CScg14App construction
CScg14App::CScg14App()
{
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CScg14App object
CScg14App NEAR theApp;

/////////////////////////////////////////////////////////////////////////////
// CScg14App initialization
BOOL CScg14App::InitInstance()
{
    SetDialogBkColor();
    LoadStdProfileSettings();
    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
        RUNTIME_CLASS(CScg14Doc),
        RUNTIME_CLASS(CMainFrame),
        RUNTIME_CLASS(CEvalua_atletasForm));
    AddDocTemplate(pDocTemplate);
    m_nCmdShow = SW_SHOWMAXIMIZED;
    OnFileNew();
```

Figura IV.2 Fragmento de el archivo SCG14.CPP

Esta clase permite crear y cerrar ventanas además de crear una barra de botones estándar, sin dejar de mencionar el menú en la parte superior izquierda de la pantalla.

A continuación se muestran las figuras IV.3 y IV.4 con la clase CSCG14Doc.

```
// Scg14DOC.h : interface of the CScg14Doc class
/////////////////////////////////////////////////////////////////////////////
class CScg14Doc : public CDocument
{
protected: // create from serialization only
    CScg14Doc();
    DECLARE_DYNCREATE(CScg14Doc)
// Attributes
protected:
    CDatabase m_database;
public:
    CNomdep                     m_nomdepSet;
    CDib*                       m_pDib;
    CColoresSet                 m_coloresSet;
    CEvalua_atletasSet          m_evalua_atletasSet;
    CDet_evaluacionesSet        m_det_evaluacionesSet;
    CAtletasSet                 m_atletasSet;
    CInstructoresSet            m_instructoresSet;
    CSomatotiposSet             m_somatotiposSet;
    CPlieguesSet                m_plieguesSet;
    CEstaturasSet               m_estaturasSet;
    CCondilo_codosSet           m_condilo_codosSet;
    CCondilo_rodillasSet        m_condilo_rodillasSet;
    CCircunfe_bicepsSet         m_circunfe_bicepsSet;
    CCircunfe_pantoSet          m_circunfe_pantoSet;
    CEstatura_pesoSet           m_estatura_pesoSet;
    CCuentaSet                  m_cuentaSet;
    CInstructores2Set           m_instructores2Set;
    CAtletas2Set                m_atletas2Set;
// Operations
public:
    CDatabase* GetDatabase();
// Implementation
public:
```

Figura IV.3 Fragmento del Archivo SCG14Doc.H

```
// scg14doc.cpp : implementation of the CScg14Doc class
#include "stdafx.h"
#include "scg14.h"
#include "daisa32.h"
#include "Scg14s01.h"
#include "Scg14p01.h"
#include "scg14doc.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CScg14Doc
IMPLEMENT_DYNCREATE(CScg14Doc, CDocument)
BEGIN_MESSAGE_MAP(CScg14Doc, CDocument)
    //{{AFX_MSG_MAP(CScg14Doc)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()
/////////////////////////////////////////////////////////////////////////////
// CScg14Doc construction/destruction
CScg14Doc::CScg14Doc()
{
    m_pDib = new CDib;
}
CScg14Doc::~CScg14Doc()
{
    delete m_pDib;
}
void CScg14Doc::DeleteContents()
{
    delete m_pDib;
    m_pDib = new CDib;
}
```

Figura IV.4 Fragmento del archivo SCG14Doc.CPP

La clase SCG14Doc es derivada de la clase CDocument que junto con la clase CView encapsulan el patrón de diseño de la vista y el documento; y trabajan con las clases derivadas de CWnd para desplegar el contenido del CDocument.

En seguida se muestran las figuras IV.5 y IV.6 con la definición de la clase CEvalua_atletasForm.

```
// scg14f01.h : interface of the CEvalua_atletasForm class
/////////////////////////////////////////////////////////////////////////////
class CEvalua_atletasSet;
class CDet_evaluacionesSet;
class CAtletasSet;
class CInstructoresSet;
class CSomatotiposSet;
class CEvalua_atletasForm : public CDaisaForm
{
protected: // create from serialization only
    CEvalua_atletasForm();
    DECLARE_DYNCREATE(CEvalua_atletasForm)
public:
    //{{AFX_DATA(CEvalua_atletasForm)
    enum { IDD = IDD_SCG14_FORM };
    CEvalua_atletasSet* m_pSet;
    //}}AFX_DATA
    CDet_evaluacionesSet* m_p2Set;
    CAtletasSet* m_p3Set;
    CInstructoresSet* m_p4Set;
    CSomatotiposSet* m_p5Set;
    CCuentaSet* m_pCSet;
    long        m_id_atleta;
    int         m_num_eval;
    CString     m_nombre_inst;
    double m_componente1;
    double m_componente2;
    double m_componente3;
    struct control_pag {
      short m_nreg1;
      int   m_ncar2;
```

Figura IV.5 Fragmento del Archivo SCG14F01.H

```
// scg14f01.cpp : implementation of the CEvalua_atletasForm class
#include "stdafx.h"
#include "scg14.h"
#include "math.h"
#include "daisa32.h"
#include "Scg14s01.h"
#include "Scg14p01.h"
#include "scg14doc.h"
#include "scg14f01.h"
#define ID_EDICION_ACTUALREGISTRO       32779
#define USER_MESS  WM_USER+1
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CEvalua_atletasForm
IMPLEMENT_DYNCREATE(CEvalua_atletasForm, CDaisaForm)
BEGIN_MESSAGE_MAP(CEvalua_atletasForm, CDaisaForm)
    //{{AFX_MSG_MAP(CEvalua_atletasForm)
    ON_WM_PAINT()
    ON_COMMAND(IDC_INITAB, OnInitab)
    ON_COMMAND(IDC_FINTAB, OnFintab)
    ON_UPDATE_COMMAND_UI(IDC_INITAB, OnUpdateInitab)
    ON_UPDATE_COMMAND_UI(IDC_FINTAB, OnUpdateFintab)
    ON_COMMAND(IDC_RESTAURA, RestauraVariables)
    ON_UPDATE_COMMAND_UI(IDC_RESTAURA, OnUpdateRestaura)
    ON_COMMAND(IDC_MODIFICAR, ModificaRegistro)
    ON_UPDATE_COMMAND_UI(IDC_MODIFICAR, OnUpdateMcdifica)
    ON_UPDATE_COMMAND_UI(ID_SESION_RECUPERAR, OnUpdateSesionRecuper
    ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
    ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
    ON_BN_CLICKED(IDC_BUTTON3, OnButton3)
    //}}AFX_MSG_MAP
```

Figura IV.6 Fragmento del archivo SCG14F01.CPP

Esta clase es derivada de la clase CDaisaForm que a su vez se deriva de la clase CRecordView. Sobre esta clase recae la mayor parte del trabajo del programa, ya que, es la encargada de controlar la interacción entre el usuario y la base de datos. En esta clase también se disparan la mayoría de las funciones del programa entre ellas las de validación de claves y fechas, impresión, despliegue de picklist, cálculo de somatotipos, etc.

```
case 0:
m_c1[i]->m_tipo = 1;
SEdit(i, j, k);
m_c1[i]->m_tabla = 0;
m_c1[i]->m_longitud = 6;
m_c1[i]->m_nombreHelp = "id_atleta";
m_c1[i]->m_llave = 1;
m_c1[i]->m_requerido = 0;
m_c1[i]->m_doubleClick = 3;
m_c1[i]->m_leyenda = (CString)"Introduzca la Clave del Atleta; Dobleclick Lista de
i++;
break;
case 1:
m_c1[i]->m_tipo = 0;
SEdit(i, j, k);
m_c1[i]->m_tabla = 0;
m_c1[i]->m_longitud = 30;
m_c1[i]->m_nombreHelp = "nombres";
m_c1[i]->m_llave = 0;
m_c1[i]->m_requerido = 0;
m_c1[i]->m_doubleClick = 0;
m_c1[i]->m_leyenda = (CString)"Introduzca el Nombre del Atleta";
i++;
break;
case 2:
m_c1[i]->m_tipo = 0;
SEdit(i, j, k);
m_c1[i]->m_tabla = 0;
m_c1[i]->m_longitud = 30;
m_c1[i]->m_nombreHelp = "a_paterno";
m_c1[i]->m_llave = 0;
m_c1[i]->m_requerido = 1;
m_c1[i]->m_doubleClick = 0;
m_c1[i]->m_leyenda = (CString)"Introduzca el Apellido Paterno del Atleta";
```

Figura IV.7 Fragmento del archivo SCG14F01.CPP Propiedades de los campos

En la figura IV.7 se muestra un fragmento del archivo SCG14F01.CPP en el cual se definen entre otras cosas el tipo de campo (1 = numérico ó 0 = de texto), longitud del campo (número de caracteres), si es llave, si es requerido, si tiene picklist (m_doubleClick = 3), leyenda que es el texto que aparecerá en la barra de estado.

```
void CEvalua_atletasForm::InicializaControl()
{
  int j;
  for(m_EnUso = 0; m_EnUso < m_totarea; m_EnUso++) {
    switch(m_EnUso) {
      case 0:
      m_area[m_EnUso].m_barra = IDC_CBAR1;
      for(j = 0; j < m_area[m_EnUso].m_totren; j++) {
        switch(j) {
          case 0:
          m_area[m_EnUso].m_ven[j].m_control[0] = IDC_ID_ATLETA;
          m_area[m_EnUso].m_ven[j].m_control[1] = IDC_NOMBRES;
          m_area[m_EnUso].m_ven[j].m_control[2] = IDC_A_PATERNO;
          m_area[m_EnUso].m_ven[j].m_control[3] = IDC_A_MATERNO;
          m_area[m_EnUso].m_ven[j].m_alta = 0;
          m_area[m_EnUso].m_ven[j].m_modif = 0;
          m_area[m_EnUso].m_ven[j].m_total = 4;
          break;
        }
      }
      break;
      case 1:
      m_area[m_EnUso].m_barra = IDC_CBAR2;
      for(j = 0; j < m_area[m_EnUso].m_totren; j++) {
        switch(j) {
          case 0:
          m_area[m_EnUso].m_ven[j].m_control[0] = IDC_NUM_EVAL;
          m_area[m_EnUso].m_ven[j].m_control[1] = IDC_FAYO_EVAL;
          m_area[m_EnUso].m_ven[j].m_control[2] = IDC_FMES_EVAL;
          m_area[m_EnUso].m_ven[j].m_control[3] = IDC_FDIA_EVAL;
          m_area[m_EnUso].m_ven[j].m_control[4] = IDC_ID_INSTRUCTOR;
          m_area[m_EnUso].m_ven[j].m_control[5] = IDC_NOMBRE_INST;
          m_area[m_EnUso].m_ven[j].m_control[6] = IDC_PESO;
          m_area[m_EnUso].m_ven[j].m_control[7] = IDC_ESTATURA;
```

Figura IV.8 Fragmento del archivo SCG14F01.CPP Asignación de número de control

En la figura IV.8 se muestra parte del archivo SCG14F01.CPP donde se define el número de control que le corresponderá a cada textbox y botón en la pantalla.

```
switch(pCampo->m_indcon) {
  case 0: ———————————————————————— # de Control
  switch(pCampo->m_mensaje) {
    case 1: ———————————————————————— # de evento 2 = enter 3 = picklist
    case 2:
    if(pCampo->m_onChange) {
      valida_id_atleta(pCampo);
    }
    break;
    case 3:
    CAtletasDial dlg;
    UpdateData();
    m_p3Set->m_strFilter.Empty();
    m_p3Set->m_strSort= (CString)"id_atleta";
    m_p3Set->Open();
    if(m_p3Set->IsEOF() || m_p3Set->IsBOF()) {
      Mensajes(1,(CString) "No Existen Registros en el Archivo Maestro de Atletas" );
      m_p3Set->Close();
      break;
    }
    dlg.m_btn = m_btn;
    dlg.m_dlg = m_dlg;
    dlg.m_edit = m_edit;
    dlg.m_lbox = m_lbox;
    dlg.m_mbox = m_mbox;
    dlg.m_scr = m_scr;
    dlg.m_sta = m_sta;
    dlg.rbtn=rbtn;
    dlg.gbtn=gbtn;
    dlg.bbtn=bbtn;;
    dlg.rdlg=rdlg;
    dlg.gdlg=gdlg;
    dlg.bdlg=bdlg;
```

Figura IV.9 Fragmento del archivo SCG14F01.CPP

En la figura IV.9 se muestra el segmento del archivo SCG14F01.CPP donde se disparan en base al número de control la función a ejecutar. En este caso al presionar Enter en el control 0 (en visual C++ los arreglos comienzan en cero), lo cual equivaldría al case 2, en el textbox de la clave del atleta, disparara la función valida_id_atleta donde el parámetro pCampo será el valor que se encuentra en dicho textbox. Y si se le da doble click en dicho textbox equivaldría al case 3 que llamaría a la clase CAtletasDial y presentará en pantalla el cuadro de dialogo Catálogo de Atletas.

A continuación se muestran las figuras IV.10 y IV.11 con la definición de la clase CMainFrame.

```
// scg14frm.h : interface of the CMainFrame class
/////////////////////////////////////////////////////////////////////////////
class CMainFrame : public CFrameWnd
{
protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)
// Attributes
public:
// Operations
public:
// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;
// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnAppExit();
    afx_msg void OnHelpPantalla();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
/////////////////////////////////////////////////////////////////////////////
```

Figura IV.10 Archivo SCG14FRM.H

```
/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction
CMainFrame::CMainFrame()
{
}

CMainFrame::~CMainFrame()
{
}
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    SIZE boton;
    SIZE imagen;
    boton.cx = 35;
    boton.cy = 34;
    imagen.cx = 25;
    imagen.cy = 24;
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;
    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE("Failed to create status bar");
        return -1;
    }
    m_wndStatusBar.SetPaneInfo(0,0,SBPS_STRETCH,0);
    m_wndStatusBar.SetPaneInfo(1,0,SBPS_NORMAL,60);
    if (!m_wndToolBar.Create(this) ||
        !m_wndToolBar.LoadBitmap(IDR_MAINFRAME) ||
        !m_wndToolBar.SetButtons(buttons,
          sizeof(buttons)/sizeof(UINT)))
    {
        TRACE("Failed to create toolbar");
```

Figura IV.11 Fragmento del archivo SCG14FRM.CPP

Esta clase es derivada de la clase CFrameWnd. La cual maneja la estructura de la ventana, es decir, la barra de botones, los controles, las cajas de texto, etc.

A continuación se muestran las figuras IV.12 y IV.13 con la definición de la clase CAtletasDial.

```
// Scg14p01.h : header file
/////////////////////////////////////////////////////////////////////////////
// CAtletasDial dialog
class CAtletasDial : public CDialog
{
// Construction
public:
    CAtletasDial(CWnd* pParent = NULL); // standard constructor
// Dialog Data
    //{{AFX_DATA(CAtletasDial)
    enum { IDD = IDD_ATLETAS };
        // NOTE: the ClassWizard will add data members here
    //}}AFX_DATA
    CAtletasSet* m_pSet;
    CEvalua_atletasSet* m_pMSet;
    CString m_valor;
    CString m_valor1;
    CString m_valor2;
    CString m_valor3;
    CFont*   m_font;
    int     m_accion;
    long    indice;
    HBRUSH  m_btn;
    HBRUSH  m_dlg;
    HBRUSH  m_edit;
    HBRUSH  m_lbox;
    HBRUSH  m_mbox;
    HBRUSH  m_scr;
    HBRUSH  m_sta;
    BYTE    rbtn;
    BYTE    gbtn;
    BYTE    bbtn;
    BYTE    rdlg;
```

Figura IV.12 Fragmento del Archivo SCG14P01.H

```
// Scg14p01.cpp : implementation file
#include "stdafx.h"
#include "Scg14.h"
#include "Scg14s01.h"
#include "Scg14p01.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CAtletasDial dialog
CAtletasDial::CAtletasDial(CWnd* pParent /*=NULL*/)
    : CDialog(CAtletasDial::IDD, pParent)
{
    //{{AFX_DATA_INIT(CAtletasDial)
        // NOTE: the ClassWizard will add member initialization her
    //}}AFX_DATA_INIT
  m_accion = 0;
  m_valor = "";
  m_valor1 = "";
  m_valor2 = "";
  m_valor3 = "";
}
void CAtletasDial::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAtletasDial)
        // NOTE: the ClassWizard will add DDX and DDV calls here
    //}}AFX_DATA_MAP
}
BEGIN_MESSAGE_MAP(CAtletasDial, CDialog)
    //{{AFX_MSG_MAP(CAtletasDial)
    ON_LBN_DBLCLK(IDC_LIST1, OnDblclkList1)
    ON_WM_CTLCOLOR()
```

Figura IV.13 Fragmento del archivo SCG14P01.CPP

La clase CAtletasDial es derivada de la clase CDialog. La clase CDialog proporciona una interfaz para administrar cuadros de diálogo. En este caso el cuadro de dialogo que presentara la clave, nombre(s), y apellido(s) de (e) l(os) Atleta(s). Que incluirá, entre otras, a las funciones DoDataExchange, OnCancel, OnOk, etc.

Siguiendo con la definición de las clases, se muestran a continuación las figuras IV.14 y IV.15 con la definición de la clase CEvalua_AtletasSet

```
//Scg14s01.h:interface of the CEvalua_atletasSet class evalua_atletas
/////////////////////////////////////////////////////////////////////////////
class CEvalua_atletasSet : public CRecordset
{
DECLARE_DYNAMIC(CEvalua_atletasSet)
public:
    CEvalua_atletasSet(CDatabase* pDatabase = NULL);
// Field/Param Data
    //{{AFX_FIELD(CEvalua_atletasSet, CRecordset)
    long        m_id_atleta;
    CString     m_nombres;
    CString     m_a_paterno;
    CString     m_a_materno;
    //}}AFX_FIELD
    long        m_id_atletaParam;
// Implementation
protected:
    virtual CString GetDefaultSQL();
    virtual void DoFieldExchange(CFieldExchange* pFX);
};
/////////////////////////////////////////////////////////////////////////////
class CDet_evaluacionesSet : public CRecordset
{
DECLARE_DYNAMIC(CDet_evaluacionesSet)
public:
    CDet_evaluacionesSet(CDatabase* pDatabase = NULL);
// Field/Param Data
    //{{AFX_FIELD(CDet_evaluacionesSet, CRecordset)
    long        m_id_atleta;
    int         m_num_eval;
    int         m_fayo_eval;
    BYTE        m_fmes_eval;
    BYTE        m_fdia_eval;
    BYTE        m_id_instructor;
```

Figura IV.14 Fragmento del Archivo SCG14S01.H

```
// Scg14s01.cpp : implementation file
#include "stdafx.h"
#include "Scg14.h"
#include "Scg14s01.h"
#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
/////////////////////////////////////////////////////////////////////////////
// CEvalua_atletasSet
IMPLEMENT_DYNAMIC(CEvalua_atletasSet, CRecordset)
CEvalua_atletasSet::CEvalua_atletasSet(CDatabase *pdb)
        : CRecordset(pdb)
{
    //{{AFX_FIELD_INIT(CEvalua_atletasSet)
    m_id_atleta = 0;
    m_nombres = "";
    m_a_paterno = "";
    m_a_materno = "";
    m_nFields = 4;
    //}}AFX_FIELD_INIT
    m_nParams = 1;
    m_id_atletaParam = 0;
}
CString CEvalua_atletasSet::GetDefaultSQL()
{
    return "evalua_atletas";
}
void CEvalua_atletasSet::DoFieldExchange(CFieldExchange* pFX)
{
    //{{AFX_FIELD_MAP(CEvalua_atletasSet)
    pFX->SetFieldType(CFieldExchange::outputColumn);
    RFX_Long(pFX, "id_atleta", m_id_atleta);
    RFX_Text(pFX, "nombres", m_nombres);
```

Figura IV.15 Fragmento del archivo SCG14S01.CPP

Esta clase es derivada de la clase CRecordset. La clase CRecordset representa un conjunto de registros de una determinada tabla resultado de realizar una consulta sobre ella (un SELECT de SQL), es decir, siempre se asocia a una tabla de una base de datos. Tiene como restricción la imposibilidad de realizar consultas compuestas (registros de más de una tabla) o las clásicas uniones de las bases de datos relacionales (JOINS).

IV.2 Depuración del Sistema

Debido a que las cosas están siempre en constante cambio, cuando ya se cree tener el programa o el sistema terminado siempre surgen cosas que no se tenían previstas u otras que cambiaron de último minuto. Por lo tanto siempre, o en un gran número de ocasiones se tienen ajustes. Y este sistema no podía ser la excepción.

En esta sección se mostrarán algunos ajustes que se le realizaron al programa SCG14 por situaciones que no se tenían previstas en un principio y algunas que surgieron al momento de programar el sistema.

Después de realizar estos ajustes, se procedió a analizar la manera de resolver la tarea de determinar el somatotipo y la composición corporal de los atletas, en base a las medidas registradas en la pantalla. Dicho procedimiento se realiza manualmente con la ayuda del formato mostrado en la figura IV.16.

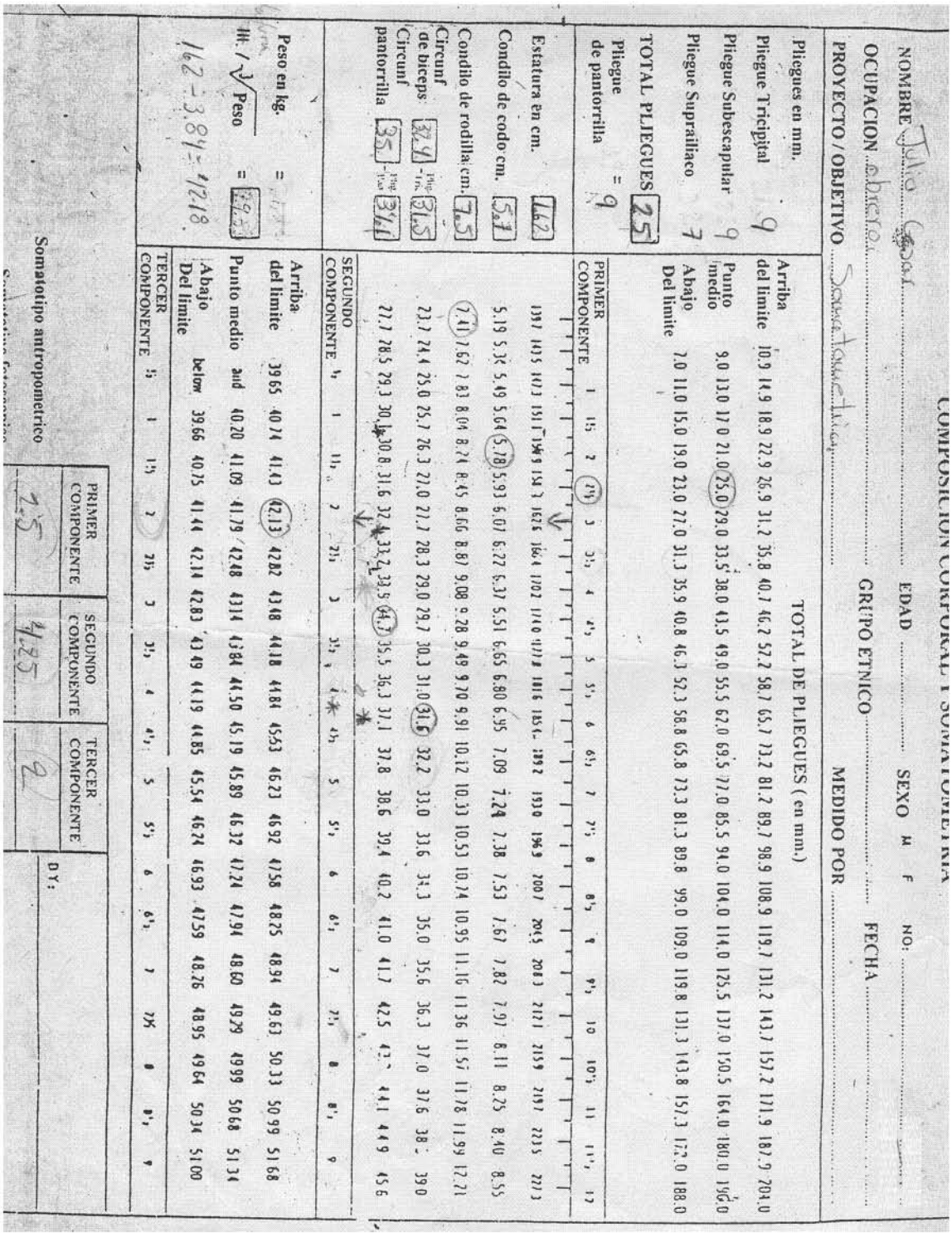


Figura IV.16 Formato de Composición Corporal y Somatometría.

Y mediante las fórmulas mostradas en la ffigura IV.17.

$$S\_C = \left[(Estatura\ )^{0.725} \times (PESO\ )^{0.425} \times (0.007184\ ) \right]$$

$$P\_O = \left(\left((Estatura\ )^2 \times C\_C \times C\_R \times 400 \right)^{0.712} \right) \times 3.02$$

P_V = 24% DEL P_T_HOMBRE S Y 21% DEL P_T_MUJERE S

$$P\_G = \left(\left[(S\_P) + (S\_C) \right] \times 0.15 \right) + 5.8$$

$$P\_M = P\_T - \left[P\_O + P\_V + P\_G \right]$$

Figura IV.17 Formulas para el cálculo de la composición corporal

Donde:

S_C = Superficie Corporal.

P_O = Peso Óseo.

C_C = Condilo de Codo.

C_R = Condilo de Rodilla.

P_V = Peso Visceral.

P_T_HOMBRES = Peso total en hombres.

P_T_MUJERES = Peso total en mujeres.

P_G = Peso Graso.

S_P = Sumatoria de los Pliegues.

P_M = Peso Muscular.

P_T = Peso Total.

Deficit Muscular = Primero se determina que % del peso total corresponde a peso muscular y a este porcentaje se le resta, en el caso de los hombres 50 y en el caso de las mujeres 45, el resultado de esta operación es el % del déficit muscular, entonces lo que se tiene que hacer es transformarlo a kilogramos mediante una regla de tres.

Nota: Los ***condilos*** se deben transformar de centímetros a metros.

Ahora en lo concerniente a la somatometría se tiene que:

Para calcular el primer componente se realiza el siguiente procedimiento.

Se suman los pliegues tricipital, subescapular y supraliaco, los cuales se encuentran en milímetros. Después se busca en el cuadro total de pliegues que esta en milímetros, el intervalo

que incluya el valor más cercano al total de pliegues obtenido. Al encontrarse el valor más cercano se toma el valor de componente que corresponde a ese intervalo.

El procedimiento para calcular el segundo componente es el siguiente:

Se toma el valor de la estatura en centímetros. Y se busca en el renglón de estaturas el valor más cercano.

Se toma el valor del condilo de codo que debe estar en centímetros. Y se busca en el renglón de condilo de codos el valor más cercano.

Se toma el valor del condilo de rodilla que debe estar en centímetros. Y se busca en el renglón de condilo de rodillas el valor más cercano.

Se toma el valor de la circunferencia de bíceps que esta en centímetros y se le resta el pliegue tricipital, el cual se tiene que transformar a centímetros, puesto que, esta capturado en milímetros. Se busca en el renglón correspondiente el valor más cercano al resultado de esta resta.

Se toma el valor de la circunferencia de pantorrilla que esta en centímetros y se le resta el pliegue de pantorrilla, el cual se tiene que transformar a centímetros, puesto que, esta capturado en milímetros. Se busca en el renglón correspondiente el valor más cercano al resultado de esta resta.

Se toman las posiciones donde se localizaron los valores más cercanos como puntos de referencia. Se determina la distancia en posiciones entre los distintos puntos de referencia y el punto en el que se localizo la estatura. Y se toma el punto más lejano.

Se calcula la distancia entre los puntos correspondientes a condilo de codos, condilo de rodillas, circunferencia de bíceps y circunferencia de pantorrilla. Con respecto a este punto más lejano. Promediamos las distancias, este promedio se le resta a la distancia del punto más lejano con respecto a estatura. Ahora se revisa a que lado de la posición de estaturas se encuentra el punto más lejano, es decir, derecha o izquierda.

En el renglón del segundo componente se toma el número 4 como punto de equilibrio, y avanzamos una distancia igual al resultado de la resta entre la distancia del punto más lejano y el promedio de distancias. En la misma dirección en la que se encontraba el punto mas lejano con respecto a la posición de estaturas.

La posición que se obtiene es el valor del segundo componente.

Para obtener el valor del tercer componente se realiza lo siguiente:

Se obtiene el peso en kilogramos y se divide la estatura entre la raíz cúbica del peso. Se busca en la tabla de valores el intervalo donde se encuentre el valor más cercano al resultado antes obtenido y se toma el valor de componente asignado a ese intervalo.

El somatotipo será el correspondiente al mayor valor de los componentes, es decir, Si el mayor componente es el primero el atleta es ectomorfo, si el mayor componente es el segundo el atleta es mesomorfo y si el mayor componente es el tercero el atleta es endomorfo.

Estas tareas se resolvieron de la siguiente manera:

La función valida_estatura ( ) se encarga de:

Al teclear el valor de la estatura se calcula la relación estatura peso, es decir, la estatura entre la raíz cúbica del peso.

La función valida_subescapular ( ) se encarga de:

Al teclear el valor del pliegue subescapular se calcula la sumatoria de los pliegues (subescapular, tricipital y supraliaco).

La función valida_tricipital ( ) se encarga de:

Al teclear el valor del pliegue tricipital se calcula la sumatoria de los pliegues (subescapular, tricipital y supraliaco).

La función valida_supraliaco ( ) se encarga de:

Al teclear el valor del pliegue supraliaco se calcula la sumatoria de los pliegues (subescapular, tricipital y supraliaco).

La función valida_condilo_rodilla ( ) se encarga de:

Al teclear el valor del condilo de rodilla calcula todos los pesos (óseo, visceral, graso, muscular y el déficit muscular).

La función valida_c_brazo ( ) se encarga de:

Al teclear el valor de la circunferencia del brazo se calcula la circunferencia del brazo – el pliegue tricipital.

La función valida_c_pantorrilla ( ) se encarga de:

Al teclear el valor de la circunferencia de la pantorrilla, se calcula la circunferencia de la pantorrilla – el pliegue de pantorrilla.

La función OnButton 1 ( ). Se dispara al dar un clic en el botón de 1er componente su función consiste en realizar una búsqueda en la tabla de pliegues que se creo al igual que las tablas estaturas, condilo_codos, condilo_rodillas, circunfe_biceps, circunfe_panto y estatura_peso. Para evitar el uso de matrices en las cuales almacenar los valores contenidos en el documento antes mostrado y facilitar la búsqueda de los valores que son necesarios para las distintas operaciones que realizara el programa.

La función OnButton 2 ( ). Se dispara al dar un clic en el botón de 2ndo componente su función consiste en realizar búsquedas en las tablas de estaturas, condilo_codos, condilo_rodillas, circunfe_biceps y circunfe_panto. Para determinar las posiciones e implementar todo el juego de condiciones (punto mas lejano, distancias entre los puntos de referencia, etc.) para determinar el segundo componente.

La función OnButton 3 ( ). Se dispara al dar un clic en el botón de 3er componente su función consiste en realizar la búsqueda en la tabla de estatura_peso. Para buscar el valor mas cercano para determinar el tercer componente. Además de determinar cual de los tres componentes es el mayor y de esta manera obtener el somatotipo correspondiente al atleta.

Con esto se resolvieron las tareas de realizar los cálculos e implementar las distintas condiciones antes mencionadas.

De manera muy similar se construyeron, los programas restantes del presente sistema. Por esta razón, espero que la explicación sobre la construcción del presente programa haya sido lo suficientemente clara.

Con ayuda del software denominado CRYSTAL REPORTS® se desarrollaron distintos listados. Los cuales cuentan con gráficas en las que se puede apreciar el avance, estancamiento o retroceso que tiene el atleta en su acondicionamiento físico, a lo largo de su estancia en el gimnasio.

En seguida se explica de una manera muy breve el procedimiento que se siguió para la construcción de los programas SCG20, SCG21, SCG22, SCG23, SCG24, SCG25, SCG26 y SCG27 cuya tarea consiste en generar los listados mencionados anteriormente.

Además del SCG26 cuya tarea es generar los diplomas que se les entregan a las personas que toman algún o algunos curso(s).

Lo primero es abrir el 32-bit Crystal Report Designer como lo muestra la figura IV.18 en la primer pantalla aparecen las opciones: nuevo reporte, abrir reporte y cancelar.

Figura IV.18 Primer pantalla de crystal reports

Al dar un click en la opción de nuevo reporte el asistente pregunta que tipo de reporte va a ser como lo muestra la figura IV.19. En este sistema se empleo la opción de Standard.

Figura IV.19 Segunda pantalla de Crystal Reports galeria de reportes.

Como se muestra en la figura IV.20 el asistente pide que se le de el origen de los datos en este caso se utilizo la opción Datafile. Pidiendo después de esto la ruta de acceso a la base de datos. Y las tablas a incluir.

Figura IV.20 Tercer pantalla de Crystal Reports ruta de acceso para la base de datos.

Al darle las tablas que se emplearán en el reporte, aparece una cuarta pantalla la cual se muestra en la figura IV.21 donde el asistente pide los campos a mostrar en el mismo.

Figura IV.21 Cuarta pantalla de Crystal Reports campos a incluir.

Al darle los campos a mostrar en el reporte, aparece una quinta pantalla la cual se muestra en la figura IV.22 donde el asistente pide el campo a partir del cual se van a ordenar los datos.

Figura IV.22 Quinta pantalla de Crystal Reports Orden de los campos.

Ya con el orden que se le darán a los datos, sigue otra pantalla la cual se muestra en la figura IV.23 donde el asistente pide los campos a sumarizar y la operación a realizar.

Figura IV.23 Sexta pantalla de Crystal Reports totales a mostrar.

Al llegar a la pestaña Graph se inicia el asistente para creación de graficas en principio de cuentas aparece la pantalla que se muestra en la figura IV.24 donde se pide que elija el tipo de grafica que se desea utilizar.

Figura IV.24 Séptima pantalla de Crystal Reports asistente para la creación de graficas.

Al llegar a la pestaña Data la cual se muestra en la figura IV.25 donde el asistente pide los campos a mostrar en el reporte, el corte, el lugar en el que se colocara la grafica además de preguntar si será una por reporte o si dependerá de algún campo.

Figura IV.25 Octava pantalla de Crystal Reports asistente para la creación de graficas.

Al llegar a la pestaña Axes la cual se muestra en la figura IV.26 donde se determinan los rangos de datos, si se presentaran o no las líneas de división, el número de divisiones, etc.

Figura IV.26 Novena pantalla de Crystal Reports asistente para la creación de graficas.

Al llegar a la pestaña Options la cual se muestra en la figura IV.27 donde vienen entre otras cosas pide que se especifique si será una grafica a color o en blanco y negro, si se mostrara la etiqueta de los campos el valor o nada, sin olvidar el tamaño y tipo de los marcadores además del tamaño de la grafica y si se mostrara o no la leyenda y de mostrarse en que lugar se localizara.

Figura IV.27 Décima pantalla de Crystal Reports asistente para la creación de graficas.

Para finalizar con la gráfica aparece una pantalla, la cual se muestra en la figura IV.28 donde se piden datos como el título que llevara la misma, el subtítulo, la nota al pie, además de los títulos de grupo y de datos.

Figura IV.28 Décimo primera pantalla de Crystal Reports asistente para la creación de graficas.

Se termina la pestaña Graph y sigue la pestaña Select, la cual se muestra en la figura IV.29 donde se pueden incluir filtros para los datos, al incluir una regla de selección y ejecutar el programa se pueden modificar estos filtros.

Figura IV.29 Décimo segunda pantalla de Crystal Reports definición de filtros.

En la pestaña de Style la cual se muestra en la figura IV.30 donde se le puede dar un título al listado, darle un estilo o agregar el logo de la compañía. Para ver la vista previa del reporte, se le puede dar un clic en el botón de preview report.

Figura IV.30 Décimo tercera pantalla de Crystal Reports definición de estilo.

Al terminar de definir las características del reporte, están las pantallas de vista previa y diseño las cuales se muestran en las figuras IV.31 y IV.32 respectivamente, en las cuales como su nombre indica se pueden hacer ajustes al diseño y ver como lucirían en tiempo de ejecución, para corregir los detalles que se presenten.

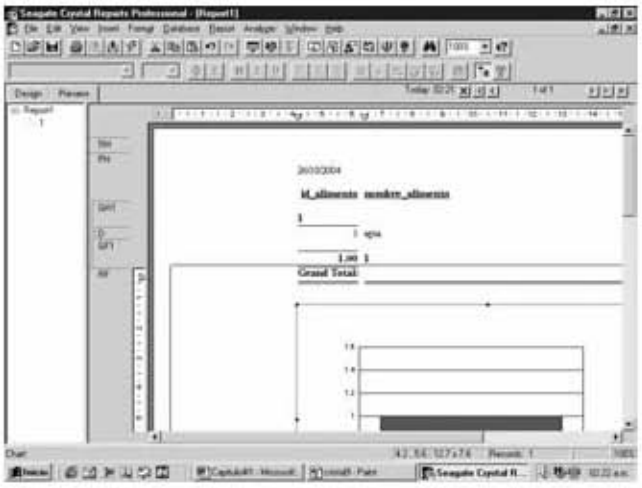


Figura IV.31 pantalla preview de Crystal Reports.

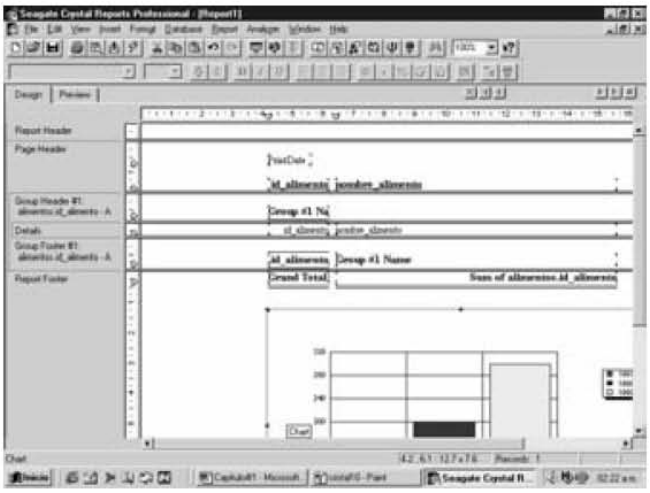


Figura IV.32 pantalla design de Crystal Reports.

A continuación se muestran las figuras IV.33, IV.34, IV.35, IV.36, IV.37, IV.38, IV.39 y IV.40, donde se presentan los listados que se realizarón para este sistema.

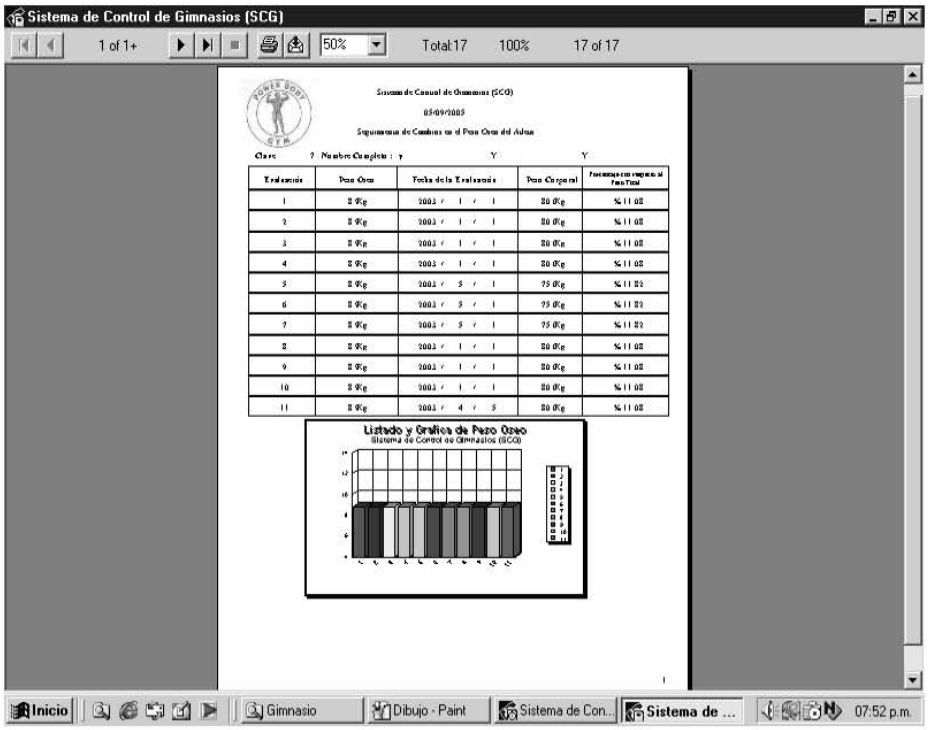


Figura IV.33 Listado para verificar el aumento o disminución en el peso oseo del atleta

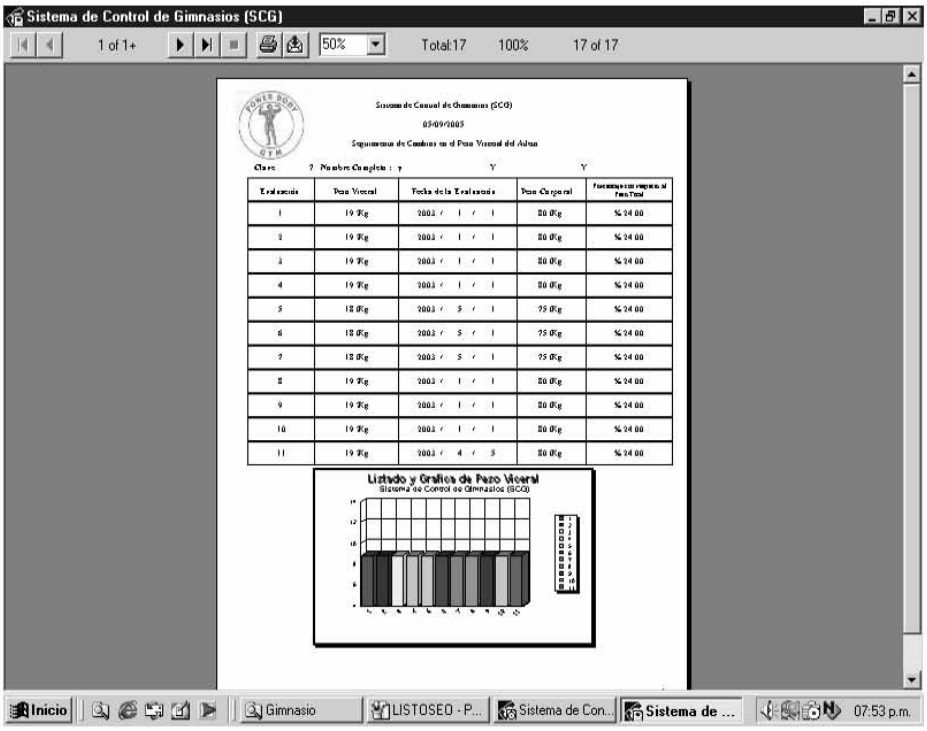


Figura IV.34 Listado para verificar el aumento o disminución en el peso viceral del atleta

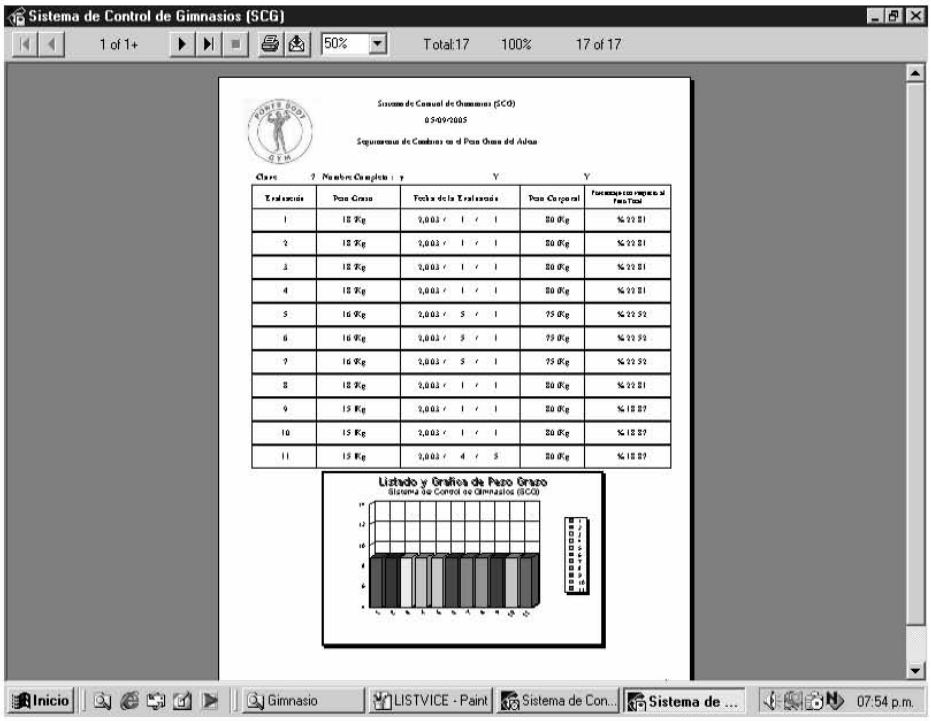


Figura IV.35 Listado para verificar el aumento o disminución en el peso graso del atleta

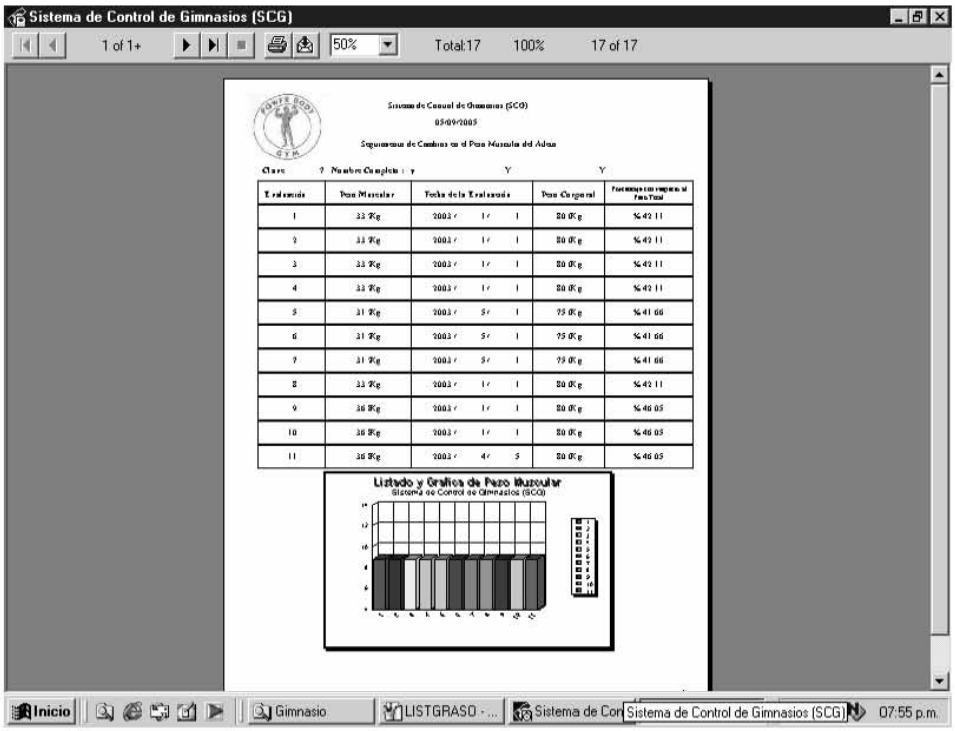


Figura IV.36 Listado para verificar el aumento o disminución en el peso muscular del atleta

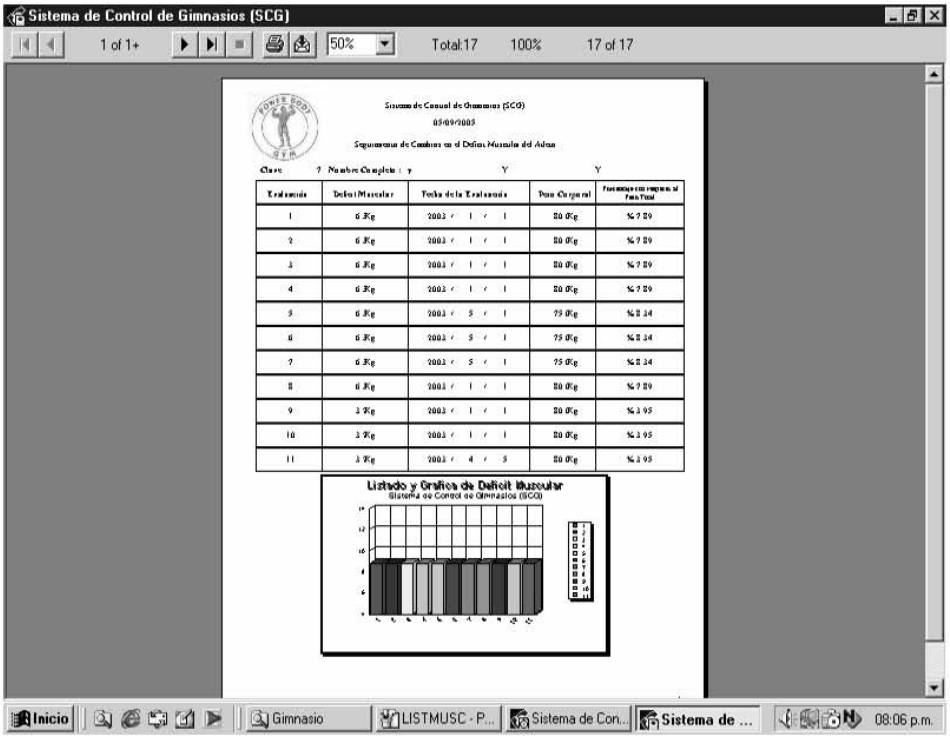


Figura IV.37 Listado para verificar el aumento o disminución en el deficit musical del atleta

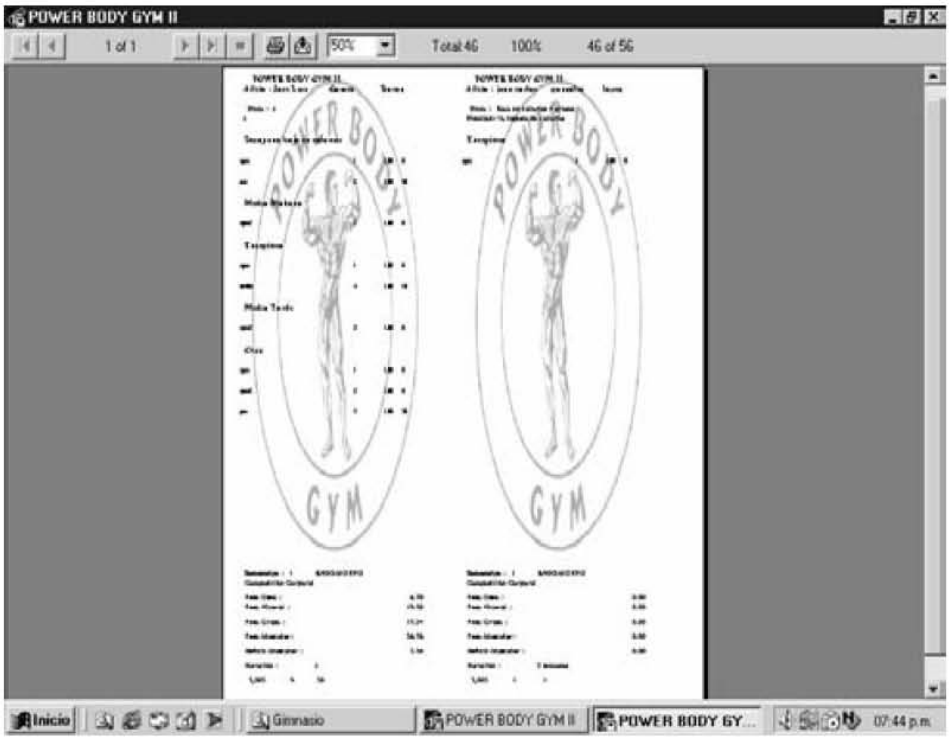


Figura IV.38 Listado para ver los resultados del atleta y la dieta que le fue asignada.

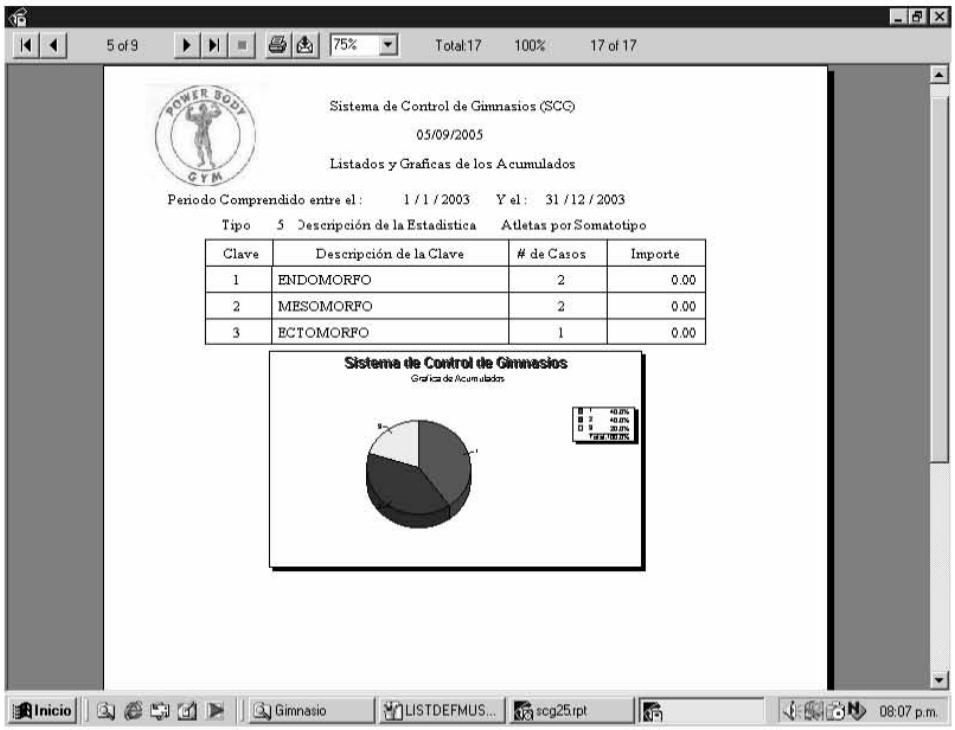


Figura IV.39 Listado para mostrar gráficamente los resultados del programa SCG19 acumulados.

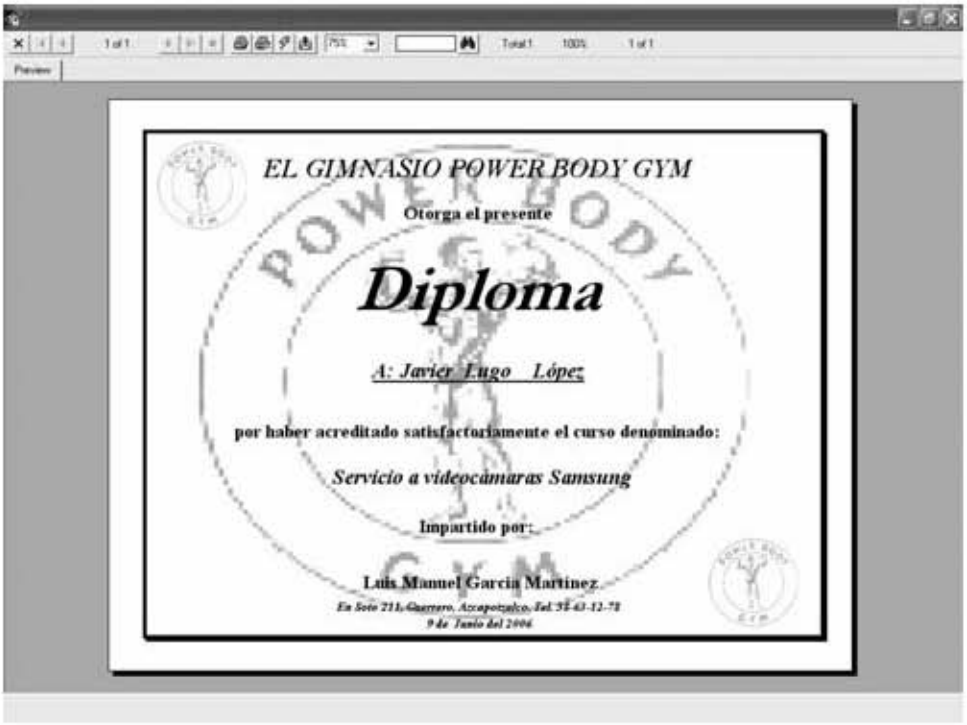


Figura IV.40 Programa SCG26 que genera los diplomas.

Con esto se puede dar por terminado el desarrollo del Sistema de Control de Gimnasios (SCG). A continuación se presenta la manera de accesar al programa SCG14 Evaluación de Atletas.

Primero al dar dobleclick al icono del programa en el escritorio de la PC, aparecerá la pantalla que se muestra en la figura IV.41.

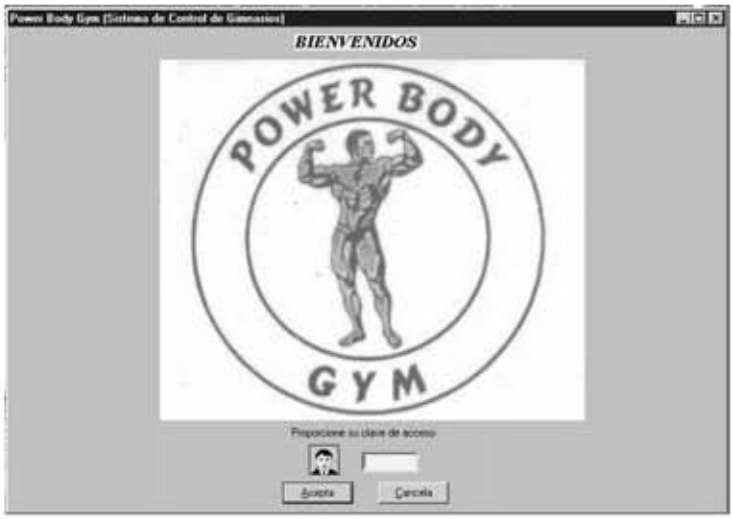


Figura IV.41 Primer pantalla del sistema.

En esta pantalla se teclea la clave de acceso que se le dio al usuario. Cada clave tendrá acceso a distintos programas, dependiendo del papel que desempeñe ese usuario dentro del gimnasio. Al teclear una clave que sea correcta aparecerá la pantalla del menú principal del sistema la cual se presenta a continuación en la figura IV.42.

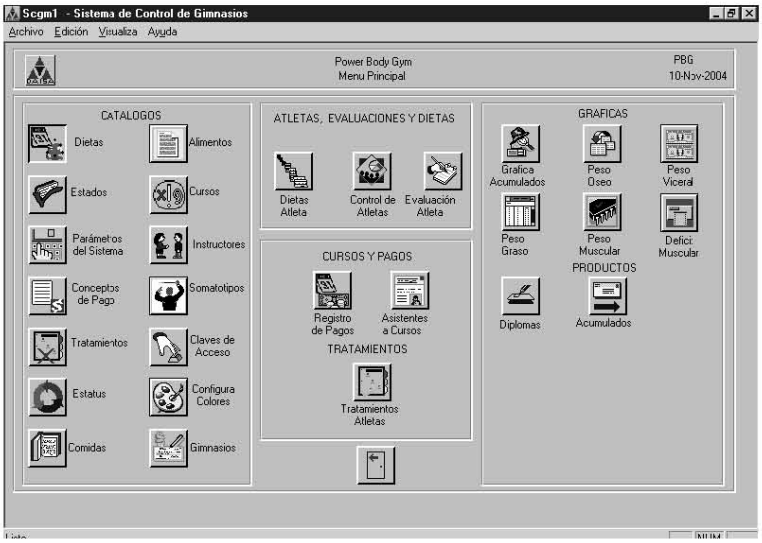


Figura IV.42 Menú principal del sistema.

Desde aquí si se tiene una clave con acceso completo, se puede entrar a cualquiera de los programas. En este caso si se le da un clic en el icono de Evaluación Atletas se entrara al programa SCG14 cuya pantalla se presenta en la figura IV.43 del cual se ha descrito su construcción en este trabajo.

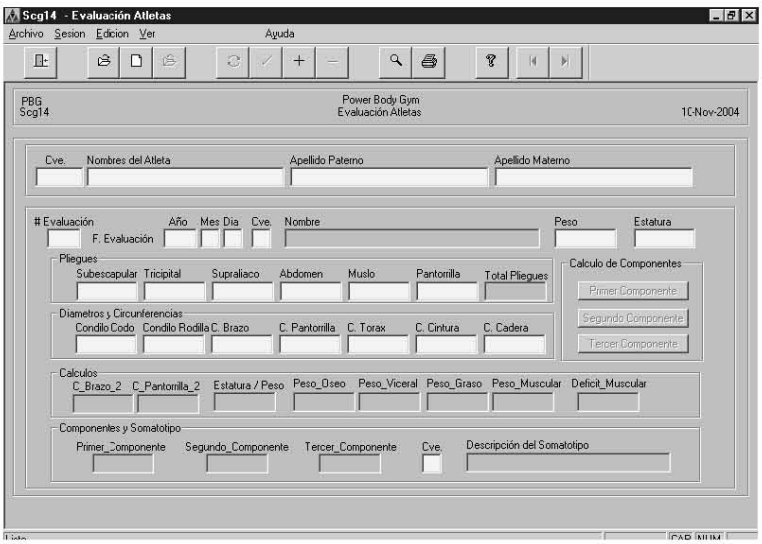


Figura IV.43 Pantalla del programa SCG14.

Otra ruta posible era entrar al menú principal y elegir el icono de Control de Atletas y estando en el programa de Atletas dar un clic en el botón de Captura de Evaluaciones.

Con esto se puede decir que se concluyeron el 3er. Y 4to Hitos técnicos: Programación y Pruebas Orientadas a Objetos en lo que se refiere al programa SCG14.

Este mismo procedimiento se siguió en la construcción de todos y cada uno de los programas que forman parte de este sistema.

Y en el siguiente capitulo se muestran las conclusiones a las que se llegaron durante la realización de este trabajo.

Conclusiones

Lo que con mucho trabajo se adquiere, más se ama.

Aristóteles

Finalmente se puede dar por concluido el Sistema de Control de Gimnasios (SCG), en lo que a la construcción del software se refiere. Por supuesto queda en marcha el mantenimiento ya que, como es de esperarse, con el paso del tiempo pueden surgir nuevas necesidades que obliguen hacer cambios, para explotar adecuadamente el sistema.

Al ir desarrollando paso a paso este sistema siguiendo la metodología orientada a objetos se han presentado de manera clara las ventajas de la misma tales como:

1. Dado que los objetos de software que se construyen están basados en el mundo real, los programas quedan expresados en los términos del problema que se está resolviendo. Logrando con esto, que sean más sencillos de comprender aún cuando se le hayan realizado modificaciones al código.
2. Lo natural del lenguaje ayuda a que otros desarrolladores entiendan el código sin problemas facilitando modificaciones posteriores durante el mantenimiento.
3. La naturaleza modular de los objetos y la herencia de paradigmas anteriores permiten realizar cambios al interior de estos sin que se afecte al programa, a menos que este cambio afecte su comportamiento externo, ya que los objetos aíslan el conocimiento y la responsabilidad al entorno donde se desenvuelven.

4. La facilidad para reutilizar fragmentos de código que ya se han probado demostrando funcionar adecuadamente. De esta manera se ahorra tiempo en el desarrollo del sistema, logrando con esto crear un software más económico para el usuario final.

5. La flexibilidad a los cambios, ya que se pueden añadir funciones adecuando solo unos cuantos objetos (los que se vean afectados por dicha función), sin que se tengan que hacer modificaciones a todo el programa.

6. El dividir el problema en objetos permite repartir el trabajo entre varias personas, al estar trabajando todas ellas en paralelo se logra reducir el tiempo de desarrollo dejando un mejor margen de tiempo para la fase de pruebas.

En lo que se refiere a la hipótesis planteada en lo que fue la introducción se puede decir lo siguiente:

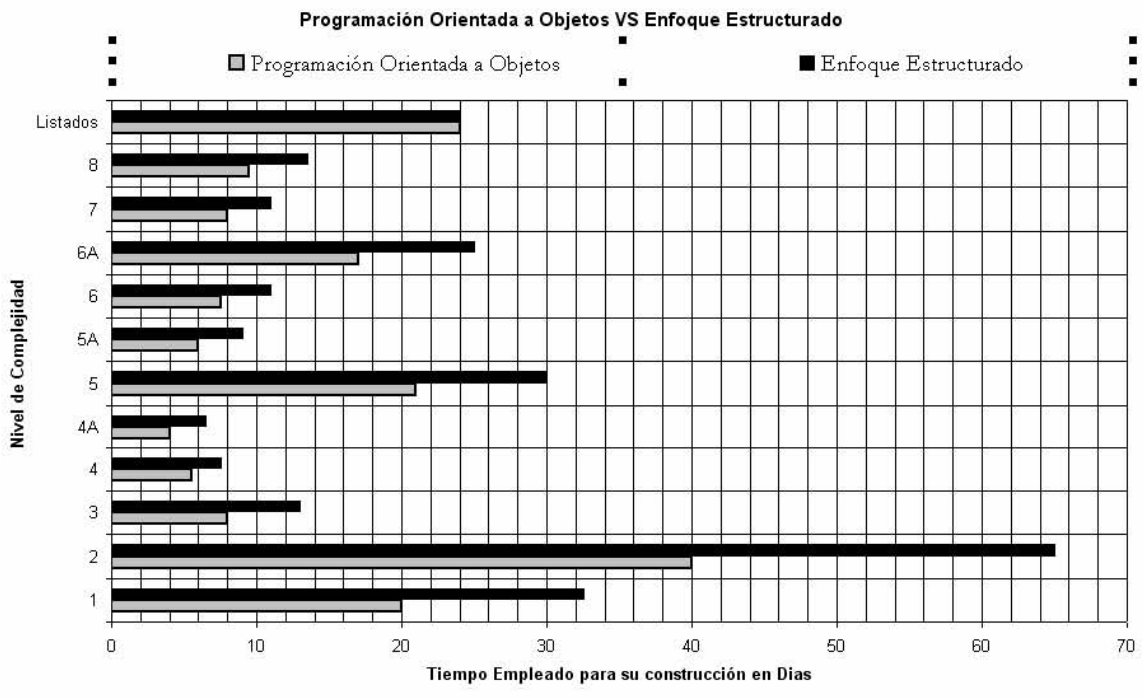


Figura V.1 Cuadro comparativo entre los tiempos empleados utilizando programación OO y los tiempos contemplados usando programación Estructurada

En la figura V.1 están los programas clasificados por nivel de complejidad esto en base a sus características (tablas consultadas, número de campos mostrados en pantalla, entre otras características). También se muestra el tiempo que se tiene contemplado se emplearía en la construcción de todos los programas de cada uno de estos niveles en el enfoque estructurado. Así como el tiempo que se llevo en la construcción de estos mediante el uso del enfoque orientado a objetos.

Como se puede observar, el tiempo empleado en la construcción de los programas que conforman este sistema mediante el Enfoque Orientado a Objetos, se llevo en promedio un 45.45% menos de días. A diferencia del tiempo que se tiene contemplado tardaríamos en realizar este mismo sistema mediante el Enfoque Estructurado.

Esta aseveración se apoya principalmente en el hecho, de que, si se utilizara el Enfoque Estructurado, se tendrían que construir módulos que controlen las diversas tareas que se realizan en la interfaz grafica, entre las que se incluyen: el control de las ventanas, botones, cuadros de texto y demás objetos que forman parte de la barra de herramientas que se localiza en la parte superior de las pantallas.

Así como algunas funciones especiales como son los pick list, la exportación de datos, etc. Para cada uno de los 29 programas que conforman el sistema, aumentando de manera considerable el tiempo necesario para realizar el análisis, diseño y desarrollo, además de dispararse la cantidad de código que será preciso generar, causando con esto que aumente el tiempo que se va a emplear en la construcción de este sistema y se eleve el costo del mismo.

Sin embargo con el Enfoque Orientado a Objetos se tiene la gran ventaja de que con la construcción de una sola biblioteca de funciones, se realiza el control de todas las tareas arriba mencionadas, en cada uno de los programas sin la necesidad de hacer grandes cambios o generar mucho código. Esto gracias al encapsulamiento, la herencia, el polimorfismo y demás características de este paradigma, que se han mencionado a lo largo de este trabajo.

A continuación se presentan los diagramas de Gantt, que muestran el tiempo estimado en el Enfoque Estructurado figura V.2 y el tiempo empleado en el Enfoque Orientado a Objetos figura V.3 para la construcción de este sistema haciendo la misma clasificación de programas por nivel de dificultad.

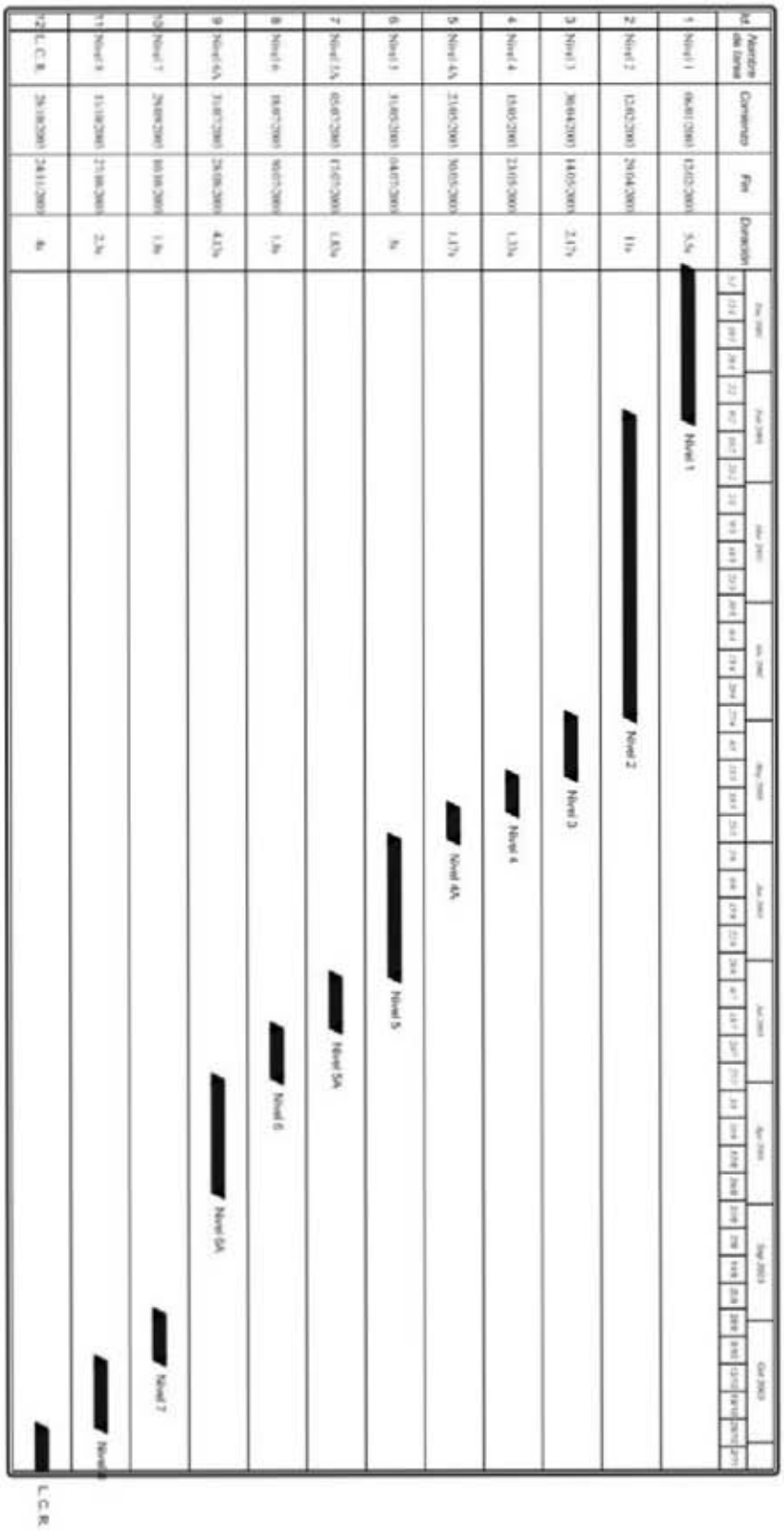


Figura V.2 Diagrama de Gantt para el Enfoque Estructurado

Diagrama de Gantt Planeación del Proyecto bajo el Paradigma Orientado a Objetos

Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Nivel 1	06/01/2003	28/01/2003	3.33s
2	Nivel 2	29/01/2003	15/03/2003	6.67s
3	Nivel 3	17/03/2003	25/03/2003	1.33s
4	Nivel 4	26/03/2003	31/03/2003	.83s
5	Nivel 4A	31/03/2003	04/04/2003	.83s
6	Nivel 5	05/04/2003	29/04/2003	3.5s
7	Nivel 5A	30/04/2003	06/05/2003	1s
8	Nivel 6	07/05/2003	15/05/2003	1.3s
9	Nivel 6A	16/05/2003	04/06/2003	2.8s
10	Nivel 7	05/06/2003	13/06/2003	1.3s
11	Nivel 8	14/06/2003	25/06/2003	1.63s
12	L. C. R.	26/06/2003	23/07/2003	4s

Figura V.3 Diagrama de Gantt para el Enfoque Orientado a Objetos.

En cuanto a los demás objetivos planteados para el Sistema de Control de Gimnasios se tiene lo siguiente:

No se pudo reducir el tiempo que se empleaba en la realización de la evaluación física de los atletas, por que ésta se realiza en forma manual mediante una cinta métrica, un *Plicómetro o compás de pliegues cutáneos* y un *paquímetro* también denominado *compás de pequeños diámetros* o en su defecto un *calibrador Vernier* también conocido como "pie de rey". Esto en lo que a recolección de datos se refiere (medidas de los pliegues, condilos y circunferencias).

En cuanto al cálculo de los distintos pesos (óseo, visceral, graso y muscular). Además de la determinación del déficit muscular, se consiguió un ahorro significativo en tiempo, ya que anteriormente los cálculos se realizaban con la ayuda de una calculadora lo cual tardaba alrededor de 15 minutos por atleta y actualmente el sistema los va calculando conforme son tecleados los valores necesarios para cada cálculo, obteniendo con esto un ahorro de tiempo de mas del 100%.

En lo que toca a la determinación del somatotipo anteriormente se tardaba de 15 a 20 minutos por persona, en evaluar los distintos factores que intervienen en la determinación de los tres distintos componentes, actualmente el sistema solo tarda unos cuantos segundos (lo que se tarda el usuario en dar un clic a cada uno de los botones correspondientes a cada componente), obteniendo con esto un ahorro de tiempo de mas del 100%.

En la elaboración y asignación de las dietas se alcanzo una reducción de aproximadamente 15 minutos en el tiempo empleado para este fin. Ya que anteriormente se tardaban alrededor de 40 minutos y actualmente con ayuda de el sistema se tardan aproximadamente 25 minutos, debido a que el sistema además de tener la composición de cada alimento, suma automáticamente la cantidad de calorías de los alimentos conforme se van eligiendo en la pantalla de pick list. Ahorrando con esto el tiempo que empleaba el instructor en revisar cuantas calorías tiene cada alimento, obteniendo con esto un ahorro de tiempo del 37.5%.

En cuanto al objetivo general de este trabajo que era demostrar que un sistema podía mejorar el control del gimnasio se tuvieron los siguientes resultados:

Se facilito el registro y control de los cursos, servicios, tratamientos, pagos, etc.

También se redujo la captura al máximo, agilizando el procedimiento de inscripción de los atletas a los diferentes servicios del gimnasio.

Se crearon listados con los distintos resultados obtenidos en lo que a peso óseo, visceral, graso, muscular y déficit muscular se refiere, mostrando el avance o retroceso que tiene un atleta mensualmente.

Entre los ya mencionados listados, también se encuentra el SCG27 en el cual se muestran los resultados obtenidos por el atleta en la evaluación, así como la dieta que debe seguir.

Por ultimo se tienen dos listados mas, uno que muestra gráficamente los resultados obtenidos por el programa de acumulados (SCG19). Y otro el (SCG26) que genera los diplomas que se les entregan a las personas que toman uno o varios cursos.

Al tener los formatos impresos, ya no es necesario depender de la buena caligrafía del instructor, para que el atleta comprenda sus resultados y conozca la dieta a seguir junto con las indicaciones extras, si es que existiese alguna. Como complemento a los puntos anteriores, se disminuyeron de manera considerable la pérdida de información y los errores cometidos.

Se puede decir que al término del presente trabajo se tiene un sistema que cumple las expectativas del usuario lo cual se puede apreciar especialmente en los siguientes puntos:

- El manejo de los catálogos permite que el sistema no dependa por completo de los datos. Por ejemplo, si en lugar de incluir un catálogo de estatus hubiera declarado los estatus como dos variables A = activo y B = baja y, surgiera la necesidad de un nuevo estatus digamos baja temporal, tendrían que buscar a alguien para hacer el ajuste al código del programa. En cambio con el catálogo de estatus tienen la capacidad de dar de alta todos los estatus que consideren necesarios o dar de baja los estatus que ya no se emplean.
- Tiene la capacidad de manejar claves de acceso, donde cada clave va a tener determinado permiso para cada programa, o puede no tener acceso a todos los programas.
- Tiene la posibilidad de cambiar los colores de la ventana de diálogo, barras de autoscroll, texto de fondo, etc.
- Con el manejo de los picklist se puede recuperar la información guardada en los catálogos. Por ejemplo, si no manejara el picklist de atletas, se tendría que teclear el nombre del atleta siempre que fuera necesario lo cual puede ocasionar errores en la captura que comúnmente son llamados “errores de dedazo”. Sin embargo con la ayuda del picklist de atletas, solo es necesario que se capture una vez el nombre del atleta en dicho catálogo, y de este modo cuando en un programa es necesario introducir el nombre del atleta, solo será necesario dar doble clic en el campo nombre del atleta y escoger de la lista que aparece, el nombre que se desea y dar un clic en aceptar. De esta

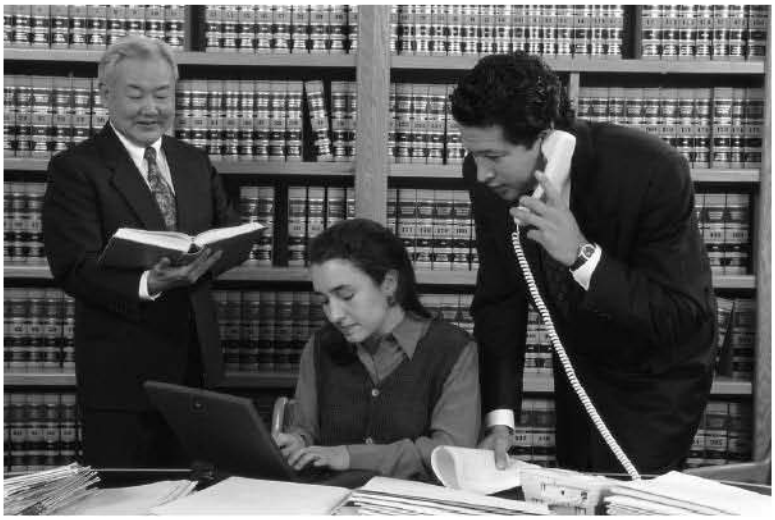
manera automáticamente bajarán el nombre y apellidos del atleta que se escogieron de la pantalla de captura.

- Válida los datos que son tecleados contra los catálogos que previamente se capturaron, esto con el fin de evitar inconsistencia en los datos que se ingresan al sistema y que generen, con esto errores, en el funcionamiento del sistema.

En general, se puede decir que al terminar este trabajo se tiene un sistema de gran calidad, en el cual se hizo todo lo posible para cumplir con todos y cada uno de los objetivos planteados.

Cabe señalar que ya se ha capacitado a los usuarios, para que conozcan el Sistema de Control de Gimnasios y estén familiarizados con él, y se efectuaron pruebas de cada uno de los procesos.

Se tiene planeado implantar el software en el gimnasio Power Body Gym en su reinauguración a partir del mes de Julio del 2006.



Glosario

La verdadera sabiduría está en reconocer la propia ignorancia.

Sócrates

Acromion: El acromion (el punto mas alto del hombro) esta formado por la parte exterior del omoplato extendiéndose sobre la articulación del hombro.

Calibrador Vernier (Pie de rey): Consiste en una regla fija de 12 cm. con precisión de un milímetro, sobre la cual se desplaza otra regla móvil o reglilla (Vernier). La reglilla graduada del Vernier divide 9 Mm. en 10 partes iguales de manera que pueden efectuarse lecturas con una precisión de un décimo de mm.[1]

Cresta Iliaca: Se localiza fácilmente en la posición de "manos en las caderas"

Condilo: Eminencia redondeada, pero no esférica, de la extremidad articular de un hueso.

[1] http://www.raulybarra.com/notijoya/archivosnotijoya3/3vernier.htm

Ectomorfo: Presentando un predominio de formas lineales y frágiles, así como una mayor superficie en relación a la masa corporal. Los tejidos que predominan son los derivados de la capa ectodérmica. Y poseen un alto índice ponderal (relación entre estatura y raíz cúbica del peso).[2]

Endomorfo: El término se origina del endoderma, que en el embrión origina el tubo digestivo y sus sistemas auxiliares (masa visceral). Indica predominio del sistema vegetativo y tendencia a la obesidad. Los endomorfos se caracterizan por un bajo peso específico, razón por la cual flotan fácilmente en el agua. Su masa es flácida y sus formas redondeadas.

Escapula: Omóplato.

Hito: Hecho importante que constituye un punto de referencia.

Mesomorfo: Se refiere al predominio en la economía orgánica de los tejidos que derivan de la capa mesodérmica embrionaria: huesos, músculos y tejido conjuntivo. Por presentar mayor masa músculo esquelética poseen un peso específico mayor que los endomorfos.

Olécranon: Apófisis del cúbito que forma la prominencia del codo.

Paquímetro o compás de pequeños diámetros: Es un compás de corredera graduado, de profundidad en sus ramas de 50 Mm., con capacidad de medida de 0 a 259 mm. Sirve para medir los diámetros óseos. La precisión es de 1 Mm.

Paradigma: Ejemplo que sirve de norma.

Plicómetro o Compás de pliegues cutáneos: También llamado espesímetro. Mide el espesor del tejido adiposo en determinados puntos de la superficie corporal. Su característica básica es la presión constante de 10 gr. /cm2 en cualquier abertura. La precisión debe de ser de 0.1 mm. Los márgenes de medida oscilan entre 0 y 48 mm.

---

[2] Las definiciones de Ectomorfo, Endomorfo, Mesomorfo, Paquímetro y Plicómetro se obtuvieron en http://www.efdeportes.com/efd84/somato.htm

Pliegue: Angulo que forma la piel en las articulaciones.

Pliegue Subescapular: Se mide debajo del ángulo inferior de la escapula, en diagonal siguiendo la línea natural de la piel en un ángulo de 45° con la columna vertebral.

Pliegue Suprailiaco: se mide diagonalmente a tres cm., aproximadamente sobre la cresta iliaca.

Pliegue Tricipital: Se mide en el punto medio entre acromion y olécranon, en cara posterior del brazo, teniendo la precaución de no incluir el músculo en la medición.

Somatotipo: El somatotipo es utilizado para estimar la forma corporal y su composición, principalmente en atletas.

Referencias

[DEL03] Diccionario Enciclopédico Larousse Ilustrado, 9ª edición, Larousse, 2003.

[BER93] Berard, E. V., Essays on Object-Oriented Software Engineering, Addison-Wesley, 1993.

[BOO91] Booch, G. Object-Oriented Design, Benjammings Cummings, 1991.

[COA91] Coad, P. y E. Yourdon, Object Oriented Analysis, 2a edición, Prentice Hall, 1991.

[JAC92] Jacobson, I., Object Oriented Software Engineering, Addison Wesley, 1992.

[RUM91] Rumbaugh et al., Object Oriented Modeling and Design, Prentice Hall, 1991.

[ROG00] Roger S. Pressman Ingeniería del Software Un Enfoque Practico, Mc Graw Hill, 4ª edición, México 2000.

[TERRY] Ferry Halpin, Information Modeling and Relational Data Bases for Conceptual Analysis to Logical Design, Morgan Kaufmann, ISBN 1-55860-672-6

[BRU92] Bruce, T. A. Designing Quality Databases With IDEF1X Information Models, Dorset House, New York, 1992.

[COD70] Codd E. A Relational Model of Data for Large Shared Data Banks, Communications of the ACM, vol. 13, Num. 6 pp. 377-387, 1970.

[AMP00] Amparo López Gaona Introducción a la base de datos, Serie: Notas de Clase, Vínculos Matemáticos #7, 200, Taller de Publicaciones del Departamento de Matemáticas de la Facultad de Ciencias, UNAM.