UNIVERSIDAD NACIONAL AUTONOMA
DE MEXICO

FACULTAD DE INGENIERIA

AUTOMATIZACION DE UN PROCESO DE EVALUACION DE PRUEBAS PARA VALORACION DE LA CAPACIDAD FISICA USANDO ORIENTACION A OBJETOS

TESIS

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

PRESENTA

FRANCISCO VALDES SOUTO

DIRECTOR DE TESIS: ING. DOMINGO PALAO MUÑOZ

MEXICO, D. F. 1999

PAGINACION DISCONTINUA

A mi Padre y a mi Madre, por haberme dejado la mejor de las herencias, pero sobre todo por haberme enseñado el camino con ejemplos.

A mis hermanos Hugo, Alejandro y Martha, por haberme dado su apoyo, amistad y cariño incondicional.

A Nancy por haberme brindado tu amor, darme tu apoyo, paciencia y enseñarme a compartir, gracias por existir.

A Jorge Valeriano Assem, por brindarme su amistad y dejarme aprender de él.

A la Universidad Nacional Autónoma de México, por darme la oportunidad de crecer.

ÍNDICE

INTRODUCCIÓN

Como en la gran mayoría de las actividades humanas, la tecnología tiene en el deporte una importancia que va en aumento en la medida que se desarrollan nuevos conocimientos con sus consiguientes aplicaciones.

En el ámbito deportivo existen dos campos de influencia de la tecnología perfectamente diferenciados: el de las innovaciones en materia de productos deportivos y el proceso de control del entrenamiento.

En el primero cotidianamente se conocen novedades de las principales marcas que compiten en el mercado de calzado, ropa instrumentos deportivos, etc. Ofreciendo infinidad de modelos con múltiples variantes para cada disciplina deportiva con el objetivo de ofrecer a los deportistas un mayor confort y seguridad en la práctica del deporte al que se dirija el producto.

En el segundo campo donde la aplicación de la tecnología en el deporte es significativa, el proceso y control del entrenamiento, ha venido evolucionando significativamente a lo largo de las últimas décadas. La influencia de la tecnología en este campo ha determinado que el proceso de desarrollo de un deportista vaya asimilando los procedimientos del desarrollo tecnológico; En función de la finalidad se estructura un proceso para modificar y controlar los resultados buscados.

Todo sistema de instrucción deportiva debe disponer de tres medios entre otros , y son los siguientes:

1.- Condiciones materiales, instalaciones y equipo de entrenamiento.
2.- Medios y métodos de entrenamiento.
3.- Medios electrónicos de recopilación de información sobre cambios en el estado del organismo de los competidores, sobre tendencias en el desarrollo de las disciplinas deportivas, sobre los principios de construcción de la forma deportiva.

Todo lo anterior nos permite entender el entrenamiento deportivo como un proceso tecnológico cuidadosamente planificado y controlado, de acuerdo con los objetivos establecidos. Para que exista un control o una correcta valoración del entrenamiento, todos estos elementos del sistema deportivo deben estar conectados e integrados informativamente para optimizar las decisiones de la dirección deportiva.

Dentro del ámbito deportivo es bien sabido que la preparación física es la base para practicar cualquier deporte, sin embargo muy pocas gentes tienen los elementos necesarios para poder argumentar el nivel de preparación física que tiene un grupo de deportistas.

Esto es debido a que por lo general la mayoría de los entrenadores son empíricos, les llamamos empíricos porque no tienen una preparación en entrenamiento deportivo, sin embargo existen entrenadores que no son empíricos que tampoco son capaces de poder hacerlo.

Para poder realizar una valoración del nivel de capacidad física de un grupo de deportistas es necesario aplicar un conjunto de pruebas que nos permita evaluar los elementos de la capacidad física que son: velocidad, resistencia, flexibilidad y fuerza, una vez teniendo los resultados se debe de hacer un análisis estadístico para poder tener una valoración objetiva además de poder tomar decisiones correctas, fundamentadas y vinculadas a un contexto, esto es porque no es lo mismo decir una persona levanta un cierto peso, a decir que él con peso que levantó se encuentra ubicado en una posición determinada con respecto a sus compañeros.

Para poder realizar un análisis estadístico se necesita tener gente encargada de ello porque involucra mucho tiempo, además del conocimiento requerido para desarrollarlo. Ésta es otra de las razones por la cual mucha gente no realiza el análisis estadístico debido a que no saben como hacerlo y si supieran tomaría mucho tiempo y ese tiempo se incrementaría conforme se incrementara el número de deportistas.

Si se quisiera hacer un análisis de los mismos datos pero a un subconjunto de gentes solamente se debe de hacer todo el procedimiento de nuevo para el subconjunto seleccionado, involucrando más tiempo todavía.

Obviamente para realizar las pruebas al grupo de personas seleccionado, se deben de tomar en cuenta cuestiones como que las condiciones sean lo más semejantes, que las personas encargadas de realizarlas y medirlas sean las mismas y además que sean objetivas.

Una vez que se ha ubicado el procedimiento para poder argumentar el nivel de preparación física, mediante este trabajo se pretende "crear un sistema que permita automatizar un análisis estadístico para poder evaluar unas pruebas de capacidad física generales, y dar la posibilidad de que un mayor número de personas encargadas de preparar físicamente a equipos de algún deporte tengan la posibilidad de tener resultados claros y concisos de la preparación física de sus equipos, sin necesidad de tener una preparación en otras áreas como las computacionales o matemáticas, que como ya se dijo es muchas veces el caso. Además de reducir en lo posible el tiempo que se lleva el realizar este proceso de forma manual".

El planteamiento expuesto anteriormente tiene algunas limitaciones debido a que si no, se podría abundar en varios aspectos por ejemplo en la teoría estadística o el uso específico para algún deporte. Las limitaciones que el problema debe de satisfacer son las siguientes:

- Debe de automatizar un proceso estadístico ya definido.
- El conjunto de pruebas de capacidad física sobre las que se va a hacer el análisis deben de estar definidas.
- Debe de ser de fácil uso, interfaz Windows.
- Debe ser un sistema "stand alone", es decir, en una sola máquina, debido a que no todas las gentes encargadas de la preparación física cuentan en sus instituciones con una infraestructura suficiente para hacer un sistema cliente servidor.
- Debe de darse la posibilidad de realizar el análisis sobre subconjuntos de la misma información de acuerdo a tres condiciones para tener cuando menos tres niveles de abstracción.

El presente trabajo se constituye de seis capítulos y tres apéndices, a través de los cuales se le da solución al problema planteado.

En el Capítulo 1, se exponen los antecedentes necesarios para el desarrollo de sistemas de computación y se hace una revisión al ciclo de vida del software, también se expone la evolución de la abstracción hasta llegar al objeto que es el concepto base de la programación orientada a objetos. Esto es con la finalidad de ir preparando el camino para posteriormente describir detalladamente la programación orientada a objetos, y mencionar algunos aspectos del desarrollo del software.

En el Capítulo 2, se hace una breve comparación de la programación orientada a objetos y de la estructurada. Aquí se presentan ya detalladamente los conceptos de la programación orientada a objetos tratando de dar una base teórica sólida para poder poner en práctica este tipo de programación .

En el Capítulo 3, se aborda la metodología de Ivar Jacobson, describiendo cada uno de los pasos que se deben de seguir para realizar un sistema utilizando esta metodología. Esta es una metodología orientada a objetos.

En el Capítulo 4, se dan un conjunto de recomendaciones para elegir un lenguaje de programación, en el cual se desarrollará un sistema orientado a objetos.

En el Capítulo 5, se dan consideraciones para el desarrollo de la interfaz de usuario y se explica el ciclo del desarrollo de la interfaz.

En el Capítulo 6, se aborda la solución del problema planteado utilizando lo descrito en capítulos anteriores y sobre todo siguiendo la metodología descrita en el Capítulo 3, aquí se realiza el análisis, diseño y construcción del sistema, tratando de cumplir con todos las necesidades planteadas.

En el Apéndice A, se muestran los resultados obtenidos del análisis de las pruebas realizado manualmente y los obtenidos mediante el sistema, para que se pueda hacer la comparación.

En el Apéndice B, se muestra parte del código con el cual se generaron los discos de instalación y/o CD.

En el Apéndice C, se muestra el manual de usuario del sistema.

1. ANTECEDENTES

1.1 DE SISTEMAS ESTRUCTURADOS A SISTEMAS ORIENTADOS A OBJETOS

La realización de tareas que para los seres humanos son poco menos que triviales, se convierten en tareas muy complejas cuando se decide implementarlas en una computadora. Ejemplos de esto son el reconocimiento de voz e imágenes, comprensión de instrucciones habladas, traducción de un lenguaje a otro y otras.

La complejidad de los procesos que existen en el mundo real se puede observar cuando un analista de sistemas pide a su cliente que explique lo que desea o que cuente en que consiste el trabajo que pretende realizar en la computadora. Generalmente el usuario de algún sistema, ve a éste con un nivel de abstracción muy alto, que se quiere decir con esto, que un usuario ve a su sistema como "el inventario", "facturación", "la nómina", es decir, difícilmente se percata de la complejidad que entraña una actividad por sencilla que parezca.

Todos los sistemas, como ya se mencionó, son entidades muy complejas en su naturaleza misma por pequeños que sean. Sin embargo, es labor del especialista en sistemas, crear la sensación de sencillez. Esta es la tendencia en cualquier rama de la tecnología, todo se va reduciendo a sencillas cajas negras.

Durante el proceso de desarrollo de la programación, se han creado métodos que permiten que el programador pueda manejar la complejidad creciente misma del desarrollo y de los mayores requerimientos de los sistemas. En cada paso de este camino, se tomaron los mejores elementos de los métodos previos y se siguió avanzando.

Hoy en día, hay muchos proyectos que están próximos o en el punto donde la aproximación de la programación estructurada ya no funciona. Además de que la creciente demanda de software exige a los desarrolladores la creación de software en muy poco tiempo, por esta razón se vuelve indispensable la reutilización del software, haciendo necesarios métodos para reutilizar el software y que lo hagan lo más flexible que se pueda. Con esto ya se tiene camino ganado ya que no es necesario empezar de cero cada vez que se crea una aplicación. De esta forma, para resolver estos problemas se creó el "Desarrollo Orientado a Objetos" (DOO).

La programación orientada a objetos (POO) toma las mejores ideas de la programación estructurada y las combina con nuevos conceptos que crean una nueva visión de la tarea de la programación. La programación orientada a objetos permite descomponer fácilmente un problema en subgrupos de partes relacionadas, traduciendo estos subgrupos en unidades llamadas objetos.

Es verdad que programar una computadora es una de las tareas más difíciles de realizar para la raza humana ya que se requiere talento, creatividad, inteligencia, lógica, habilidad para manejar abstracciones y experiencia, además de que las herramientas estén disponibles. La programación orientada a objetos es una nueva técnica que permite incrementar la productividad y mejorar la confiabilidad de los sistemas. La programación orientada a objetos es más que una simple colección de nuevos lenguajes de programación, es una nueva manera de pensar acerca de lo que significa computación, acerca de cómo podemos estructurar la información dentro de una computadora.

UN NUEVO PARADIGMA

Paradigma viene del latín paradigma y del griego paradeigma, originalmente significa un ejemplo ilustrativo, el historiador Thomas Kuhn[1] expande la definición a : un conjunto de teorías, estándares y métodos que juntos representan una manera de organizar conocimiento; esto es, una forma de ver el mundo, *"también podemos enfatizar que el lenguaje que empleamos para hablar influye directamente en la forma en que vemos el mundo"*[2], en este sentido la programación orientada a objetos es un nuevo paradigma. La visión de orientación a objetos nos fuerza a reconsiderar nuestra forma de pensar y desarrollar la computación.

Las aplicaciones son complejas porque modelan la complejidad del mundo real. Actualmente existen aplicaciones muy complicadas para que un sólo individuo las implemente, esto tomaría mucho tiempo. Además la aplicación puede estar mucho tiempo activa y acumulará una variedad de adecuaciones a las necesidades, de esta forma lo que debe de ser arreglado va siendo más complicado.

El problema empeora cuando se debe de adicionar una pieza a un sistema, porque esto implica que una adición potencial de interacciones: cada pieza nueva debe interactuar con cada otra pieza del sistema, de esta forma el problema crece exponencialmente con cada parche adicional. Algunas de estas interacciones no deberían de ocurrir, pero el problema es que no se podría saber con seguridad cuáles ocurrirían y si son necesarias. Cada problema arreglado es capaz de introducir numerosos problemas similares en algunas de las partes relacionadas.

[1] Thomas S. Kuhn ,THE STRUCTURE OF SCIENTIFIC REVOLUTIONS, 2ª de., University of Chicago Press, Chicago IL, 1970, citado en Timothy Budd, AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING, edición con correcciones, Addison-Wesley Publishing Company, Oregon State University, abril 1991, pág. 2

[2] Timothy Budd, AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING, edición con correcciones, Addison-Wesley Publishing Company, Oregon State University, abril 1991,pág.2

El desarrollo orientado a objetos pretende manejar la complejidad relativa a los problemas del mundo real haciendo una abstracción de conocimiento y encapsulándolo en objetos. Encontrar o crear esos objetos es un problema de estructuración de actividades y conocimiento.

1.2. HISTORIA DE LA ABSTRACCIÓN

LA ABSTRACCION COMO UN PROCESO MENTAL NATURAL

La gente típicamente entiende el mundo real construyendo modelos mentales de porciones de él; un modelo mental es una vista simplificada de cómo trabaja algo y tu interactuas con ese algo. En esencia, este proceso de construcción de modelos es el mismo que el diseño del software, el diseño de software produce un modelo que puede ser manipulado por una computadora.

Pero el modelo mental desarrollado debe de ser muy simple de tal forma que pueda ser usado. Por ejemplo, consideremos un mapa de un territorio, para que sea útil el mapa debe de ser más simple que el modelo del territorio, si incluyera cada detalle debería de ser del tamaño del territorio, y no serviría para su propósito.

Un mapa nos es útil porque abstrae sólo las características que queremos modelar de un territorio, así, tenemos varios mapas para diferentes propósitos, mapa de caminos, topográfico etc.

Como un mapa puede ser significante más pequeño que su territorio, e incluir cuidadosamente información seleccionada, así los modelos mentales abstraen de un sistema esas características requeridas para nuestro entendimiento, ignorando características irrelevantes. Este proceso de abstracción es psicológicamente necesario y natural; la abstracción es crucial para entender el mundo, esencial para el funcionamiento de la mente humana y es muy poderosa para luchar contra la complejidad.

MECANISMOS DE ABSTRACCIÓN

Como ya se dijo los programadores han luchado con el problema de la complejidad de los problemas a resolver a lo largo del tiempo en la historia de la computación. A continuación se revisarán la variedad de mecanismos que los programadores han utilizado para controlar la complejidad, haciendo hincapié en algunos de ellos. Las técnicas de orientación a objetos han sido vistas como una salida de una progresión histórica.

En los primeros días de la computación, los programadores enviaban instrucciones binarias a una computadora acomodando "switches" en el panel. El lenguaje ensamblador creó mnemónicos que son abstracciones diseñadas para eliminarle la necesidad a la gente de tener que recordar la secuencia de bits de los cuales las instrucciones estaban compuestas.

El siguiente uso de la abstracción se aplica en la habilidad para crear instrucciones definidas por el usuario. El conjunto de instrucciones de una máquina dada puede ser extendido por la agrupación de secuencias de las sentencias básicas en macroinstrucciones y nombrándolas. Un conjunto de instrucciones realizado, puede ser invocado por una sola macroinstrucción. Una macroinstrucción hace que la máquina haga muchas cosas.

Los lenguajes de programación de alto nivel permiten a los programadores alejarse de las arquitecturas específicas de una máquina dada. Ciertas secuencias de instrucciones fueron reconocidas con un uso universal, y los programas fueron escritos en términos de ellas solamente. Cada instrucción puede invocar una diversidad de instrucciones máquina, dependiendo de en que máquina el programa sea compilado. Esta abstracción permitió a los programadores escribir software para un propósito genérico sin preocuparse en que tipo de máquina correría el programa.

Secuencias de sentencias de lenguajes de alto nivel pueden ser agrupadas en procedimientos e invocadas por una sentencia.

PROCEDIMIENTOS

Los procedimientos y funciones fueron unos de los primeros procesos de abstracción utilizados en los lenguajes de programación. Los procedimientos permiten ejecutar tareas repetidamente, o ejecutarlas con pequeñas variaciones, reunir las tareas en un sólo lugar para reutilizarlas en lugar de que el código sea duplicado varias veces. El procedimiento también da la primera posibilidad de ocultamiento de información. Un programador puede escribir un procedimiento o un conjunto de procedimientos que pueden ser usados por otros programadores que no necesitarán saber los detalles exactos de la implantación, sólo necesitan la interfaz. Pero los procedimientos no son la solución a todos los problemas, en particular no son un mecanismo efectivo para el ocultamiento de la información, y solamente resuelven parcialmente el problema de que varios programadores hagan uso de los mismos nombres.

Para ilustrar este problema, podemos considerar a un programador que escribe un conjunto de rutinas para implementar un simple "stack". Desarrolla 4 rutinas llamadas: "init", "push", "pop" y "top". Podemos saber que los datos contenidos en el "stack" no deben ser locales a ninguna de las rutinas, sino que deben compartirse. Por lo tanto se tendría en una variable global el "stack". Pero esto implica que no se puede limitar el acceso a ella, además de que aunque sea una variable global no se utiliza en el programa sino solamente en esas rutinas que lo manejan. De esta forma otros programadores que fueran a usar las rutinas deberán declarar

una variable global para un nombre impuesto por la persona que desarrollo las rutinas, además de que no podrán declararse variables con el nombre del "stack" así como de las rutinas ya que pasan a ser palabras reservadas.

MÓDULOS

Un módulo provee la habilidad de dividir en dos partes un "name space". Un "name space" puede ser traducido como un espacio nombrado y se refiere a un proceso que se llama de alguna manera. La parte pública puede ser accesada desde cualquier punto fuera del módulo, sin embargo la parte privada es accesible dentro del módulo. Variables y procedimientos pueden ser definidos en esta parte.

David Parnas[3] quien popularizó la noción de los módulos, describe los siguientes dos principios para su uso.

1. Uno debe dar al usuario la información necesaria para usar el módulo correctamente, nada más.

2. Uno debe dar al que implementa el módulo toda la información necesaria para completar el módulo y nada más.

Esto se puede expresar de la siguiente manera "Si no se necesita conocer alguna información, no se debe tener acceso a ella". A esto se le llama ocultamiento de la información.

Los módulos resuelven una parte pero no todos los problemas de los desarrolladores de software. Por ejemplo los módulos permitirían ocultar los detalles de la implantación del "stack", pero ¿Qué sucede si se quiere tener más de un "stack"?. Esto es el problema ya que los módulos por si mismos proporcionan un método efectivo de ocultamiento de información pero no permiten el desarrollo de instanciamiento, que es la habilidad de hacer múltiples capas de áreas de datos es este caso, código.

[3] David L. Parnas ,ON THE CRITERIA TO BE USED IN DECOMPOSING SYSTEMS INTO MODULES, Comunication of the ACM, 15(12):1059-1062, 1972, citado en Timothy Budd, AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING,edición con correcciones, Addison-Wesley Publishing Company, Oregon State University,abril 1991, pág. 12

TIPOS DE DATOS ABSTRACTOS

Un tipo de dato abstracto es un tipo de dato definido por el usuario que puede ser manipulado de manera similar a los tipos de datos definidos por el sistema. Como éstos últimos, un tipo de dato abstracto corresponde a un conjunto (posiblemente infinito en tamaño) de datos válidos y a un número de operaciones que pueden desarrollarse con esos valores. Los usuarios pueden crear variables con valores que estén en el rango de valores permitidos y pueden operar en esos valores usando operaciones definidas.

Para nuestro ejemplo se podría definir el "stack" como un tipo de dato abstracto, y las operaciones del "stack" como las únicas operaciones permitidas para ser desarrolladas en instancias del "stack".

Para construir un tipo de dato abstracto debemos ser capaces de:

- Exportar una definición del tipo.
- Hacer disponibles un conjunto de operaciones que puedan ser usadas para manipular instancias del tipo.
- Proteger los datos asociados al tipo, es decir, sólo podrán ser operados por las rutinas proporcionadas.
- Hacer múltiples instancias del tipo.

OBJETOS

Se dice que la definición de un objeto es un tipo de dato abstracto. Las técnicas de programación orientada a objetos agregan nuevas ideas al concepto de tipo de dato abstracto, la más importante es el paso de mensajes o estímulos. La actividad inicia por una solicitud hecha a un objeto específico, no por una función usando un tipo de dato específico para ser invocada. En los puntos de vista convencionales la principal importancia radica en la operación, mientras en la programación orientada a objetos radica en el valor. Esto se puede ver de forma más clara en nuestro ejemplo con una pregunta ¿Se llama a la rutina "push" con un "stack" y un valor? o ¿Se le pide al "stack" hacer un "push" con un valor en sí mismo?.

Implícito en el paso de mensajes está la idea de que la interpretación de un mensaje puede variar con diferentes objetos. Esto es, que el comportamiento y respuesta que el mensaje produce

va a depender del objeto que recibe el mensaje. Así "push" puede significar una cosa pensando en el "stack" y otra muy diferente para un controlador de un brazo mecánico.

Finalmente la programación orientada a objetos suma los conceptos de herencia y polimorfismo. La herencia permite a los diferentes tipos de datos capturar el mismo código reduciendo así el tamaño de código y la funcionalidad. El polimorfismo permite que este código compartido se adecue a circunstancias específicas de cada tipo de dato. El énfasis en la independencia de componentes individuales permite un proceso incremental de desarrollo, en el cual son diseñadas unidades individuales de software.

1.3. CICLO DE VIDA DEL SOFTWARE

De acuerdo a la definición dada de paradigma en el capítulo primero podemos entender que es una forma de ver el mundo, así, el paradigma del ciclo de vida del software, es una forma de cómo se va a desarrollar el software durante su vida. En este sentido existen varios enfoques válidos o paradigmas que observan o ven de diferente forma el ciclo de vida del software.

El desarrollo de sistemas es visto usualmente como un proceso lento, el cual puede tomar varios años desde el principio hasta el final. Históricamente, en la mayoría de los proyectos relacionados con computadoras, los requerimientos para un sistema son especificados como un todo.

Es la regla, en lugar de la excepción, que los requerimientos del sistema tanto de información como técnicos no son conocidos completamente al principio del proyecto. Se sabe que el sistema crece progresivamente. Cuando la primera versión del sistema esta en operación, aparecen nuevos requerimientos y lo anterior debe de cambiar. De esta manera el sistema como un todo no puede ser desarrollado completamente ya que la especificación de requerimientos va a cambiar constantemente durante el periodo de desarrollo, el cual puede ser de varios años para sistemas grandes.

En muchos casos es mejor desarrollar el sistema paso a paso, empezando con pocas de sus funciones medulares. Como es correcto esta dirección nos va a llevar a un claro y mejor entendimiento de la funcionalidad del sistema, pudiendo adicionarse nuevas funciones. De esta manera, el sistema es alargado incrementalmente hasta que el nivel deseado es alcanzado (esto es, que el producto final esté disponible, normalmente el primer producto se libera como se muestra en la figura 1.1), una estrategia incremental también provee una rápida retroalimentación durante el proceso de desarrollo. En la práctica el desarrollo incremental de sistemas significa que podemos dividir el sistema en partes, correspondientes a los servicios solicitados por los clientes.

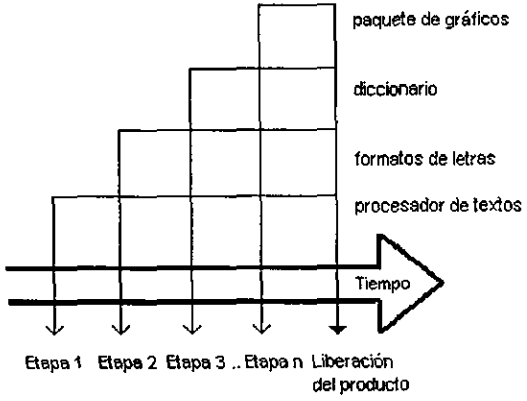


Figura 1.1. Desarrollo incremental

Como se puede observar cada nueva etapa extiende el sistema con nueva funcionalidad para el producto final que reúne todo lo del sistema deseado. Además todas las liberaciones siguientes del sistema son desarrolladas como series de etapas incrementales.

El desarrollo de sistemas puede ser visto como un proceso para producir descripciones de modelos. Esto es cierto para todos los niveles - análisis, diseño, implantación y prueba. En este contexto el código fuente es visto como una descripción que puede ser entendida por programadores y también por el proceso de producción (un compilador y ligador). Las descripciones presentan modelos de diferentes grados de detalle. Los primeros modelos son bastante abstractos, enfocándose en cualidades externas del sistema, considerando que los modelos posteriores serán más detallados e instructivos en el sentido que describirán como será construido el sistema.

En esencia, el desarrollo de sistemas consiste en tres fases que sigue una después de otra; análisis, construcción y pruebas (ver la figura 1.2). En el análisis se desarrolla una especificación orientada a la aplicación para especificar que ofrecerá el sistema a sus usuarios. En esta temprana etapa, cuando los cambios relativamente no son costosos, el objetivo es encontrar también una buena estructura para el sistema, es decir una estructura que sea robusta permitiendo cambios y la cual será dividida en unidades claras, comprensibles e indivisibles que puedan ser ordenadas. Esta especificación, la llamamos "modelo de análisis", especifica el comportamiento funcional del sistema bajo circunstancias prácticamente ideales y sin tomar en cuenta el ambiente particular de implantación.

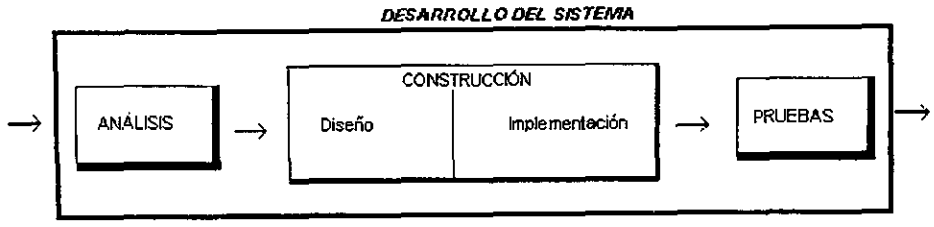


Figura 1.2. Fases del desarrollo de sistemas

Durante la construcción, las condiciones idealizadas del análisis van a ser reemplazadas gradualmente por requerimientos del ambiente de implantación elegido. Esta fase define cómo va a ser realizado el modelo de análisis orientado a la aplicación utilizando componentes como software del sistema, sistemas manejadores de base de datos e interfaces de usuarios. Las actividades de la fase de construcción constan de diseño e implantación. Las actividades de diseño formalizan el modelo de análisis en términos del ambiente de implantación y especifican los bloques de construcción identificados. Los programas separados (bloques) identificados en el diseño son entonces codificados (esto es, implementados).

En la prueba, el sistema es checado para asegurar que todos los paquetes de servicios en el modelo de análisis han sido correctamente implementados y que el desempeño del sistema cumple con los requerimientos.

¿Qué es un buen sistema?. Esta pregunta puede responderse parcialmente desde un punto de vista externo del sistema y parcialmente desde un punto de vista interno del sistema. El punto de vista externo es de quien usa de alguna forma el sistema. Ellos quieren que el sistema de resultados correctos rápido, que sea confiable, fácil de aprender y usar además de ser efectivo. El punto de vista interno es dado por los que desarrollaron el sistema y los que tienen que mantenerlo. Ellos quieren que el sistema sea fácil de modificar y extender, fácil de entender, que contenga partes reutilizables, fácil de probar, que sea compatible con otros sistemas, portable y poderoso.

Con el objeto de diseñar un buen sistema, diferentes métodos de desarrollo de sistemas han sido propuestos tanto para describir un proyecto para el desarrollo de una primera versión, como para tener una vista global entera del ciclo de vida del sistema.

El desarrollo de sistemas usualmente contiene las mismas fases (aunque puede dárseles diferentes nombres en los diferentes métodos), y el desarrollo siempre ocurre incrementalmente sobre ellas. Este escenario de desarrollo es cierto independiente del método usado. Podemos caracterizar el desarrollo de sistemas como turbulento al principio, pero estable subsecuentemente. El método de desarrollo usado debe, por lo tanto, hacer que el proceso de desarrollo sea estable tan pronto como sea posible. La idea es que se debe de trabajar en el análisis tanto tiempo como sea suficiente para entender el sistema en su totalidad, pero no tanto como para considerar los detalles que modificaremos sobre el diseño. Esto muchas veces implica relativamente que una gran parte del trabajo desarrollado se haga en la fase de análisis. Una división típica de tiempos para los proyectos es mostrada en la siguiente figura.

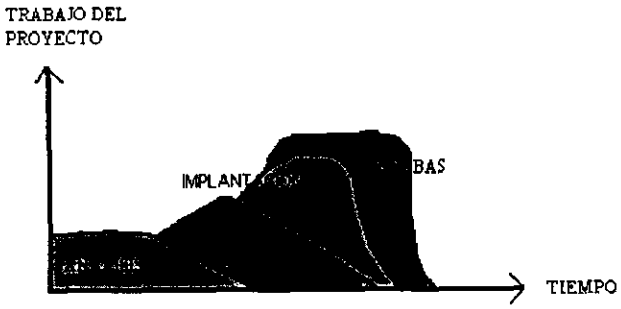


Figura 1.3. Distribución de tiempos en un proyecto

Inicialmente un pequeño grupo de gente desarrolla el análisis y luego el diseño. Estas actividades son trabajadas iterativamente. Como se estabiliza la estructura del sistema, más gente puede ser involucrada en la implantación y las pruebas. Sin embargo las actividades de análisis y diseño pueden ser realizadas aún cuando las pruebas inicien. En esta etapa es muy importante que los cambios que se hagan en los modelos de análisis y diseño se incorporen.

A continuación se describen algunos métodos de desarrollo de sistemas:

MODELO DE CASCADA

"*El paradigma del ciclo de vida clásico para la ingeniería del software. Algunas veces llamado "modelo en cascada". El paradigma del ciclo de vida del software exige un enfoque sistemático y secuencial para el desarrollo del software*"[4]. Este comienza con la definición del sistema, y sigue a través del análisis de los requisitos del sistema, el diseño, la codificación la prueba y el mantenimiento. A continuación se muestra un esquema de este paradigma.

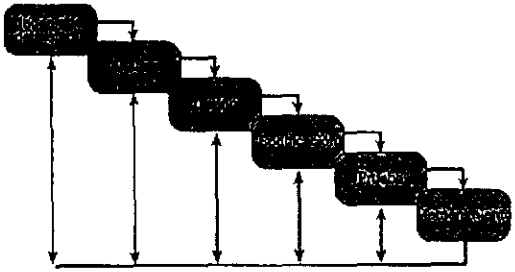


Figura 1.4. Modelo de cascada

[4] Pressman S. Roger, INGENIERÍA DE SOFTWARE UN ENFOQUE PRÁCTICO, 3a edición, McGraw-Hill, España, 1993, pág. 26

Este paradigma es el más usado y antiguo, sin embargo, tiene algunos detalles por los cuales ha sido criticado, sobre todo en su aplicabilidad.

Los proyectos reales muy rara vez siguen el flujo del modelo, ya que siempre hay iteraciones que crean problemas en su aplicación.

Este modelo necesita que todos los requisitos del sistema sean descritos explícitamente al principio, y esto es sumamente difícil para el cliente. Además de que en muchos proyectos al principio existen muchas incertidumbres.

Es necesario también que el cliente tenga paciencia ya que para tener una versión operativa del sistema, se tienen que llegar a etapas finales. Esto tiene como problema de que una falla que se detecte en una versión operativa puede tener consecuencias desastrosas.

CREACIÓN DE PROTOTIPOS

En muchas ocasiones es difícil determinar cómo se supone que debe de trabajar un sistema ya que el cliente no identifica los requisitos detallados sino define los objetivos generales, o el programador no esta seguro de la eficiencia de un algoritmo, de la interacción hombre-máquina, etc,. Es de mucha ayuda en estos casos desarrollar un prototipo. Un prototipo resalta ciertas propiedades del sistema deseado, otras partes pueden ser desatendidas, y sólo se necesitan que sean dadas en forma esquemática. El prototipo permite a los desarrolladores experimentar con un número diferente de opciones designadas. Así el prototipo sirve como complemento de un desarrollo incremental de sistemas.

La creación de prototipos facilita al programador la creación de un modelo de software a construir. El modelo puede tomar cualquiera de las siguientes formas:

- Un prototipo en papel, o en computadora que describa la interacción hombre - máquina, de tal forma que facilite al cliente entender como se va a llevar a cabo la interacción.

- Un prototipo que implemente alguna(s) partes de alguna(s) función(es) del sistema requerido.

- Un prototipo que será un programa que ya existe, que ejecute parte o toda la función deseada, pero que se deba de mejorar en otras características.

Una ventaja específica del desarrollo de prototipos es que puede servir como medio de comunicación entre el desarrollador y el cliente, ya que es mucho más fácil expresar un punto de vista sobre algo que puede ser demostrado y usado aunque sea parcialmente, que expresar una opinión de las especificaciones.

El desarrollo de prototipos es una técnica usual para comprender una aplicación. Además si el prototipo es diseñado cuidadosamente (utilizando el ciclo normal de desarrollo, análisis, construcción, pruebas), puede ser de utilidad en el sistema final.

Como en los otros métodos, la construcción de prototipos empieza con la definición de requerimientos, aquí se identifican las áreas donde se requiere una mayor definición, después se produce un "diseño rápido". *"El diseño rápido se enfoca sobre la representación de los aspectos del software visibles al usuario (por ejemplo, métodos de entrada y formatos de salida)"*[5]. El diseño rápido nos va a llevar a la construcción del prototipo que deberá ser evaluado por el cliente y se utilizará para detallar y mejorar los requisitos del software a desarrollar. En este proceso se producen dos cosas importantes: se mejora el prototipo de tal forma que cumpla con las necesidades del cliente y se logra un mayor entendimiento de lo que hay que hacer.

La siguiente figura muestra la secuencia en el paradigma de construcción de prototipos.

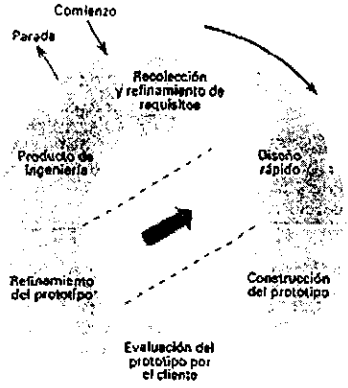


Figura 1.5. Paradigma de construcción de prototipos

En el paradigma de la construcción de prototipos se presentan algunos problemas.

- El cliente ve funcionando una versión de su sistema, como no sabe que no se ha tomado en cuenta nada de calidad ni de mantenimiento a largo plazo, (ya que es un prototipo rápido) cuando se le dice que se debe de reconstruir, él pide que se hagan las mejoras correspondientes para que sea un producto final.

- El programador muchas veces toma en cuenta ciertos compromisos para la implantación de los prototipos como lenguajes, algoritmos, etc., que no son los ideales, sólo se utilizaron para el prototipo. Después de algún tiempo el programador se familiariza y lo que hizo en el prototipo forma parte integral del sistema.

[5] Pressman S. Roger, INGENIERÍA DE SOFTWARE UN ENFOQUE PRÁCTICO, 3a edición, McGraw-Hill, España, 1993, pág. 28

Aunque el desarrollo de prototipos presenta problemas, es un buen método, sólo que sí es necesario definir las condiciones antes de empezar, esto es quedar de acuerdo cliente y programador que el desarrollo de prototipos sólo se utilizará como un mecanismo de definición de requisitos y no como producto final, por lo tanto tiene que ser descartado, al menos en parte y buscar la calidad y el mantenimiento a largo plazo.

Finalmente podemos concluir que el desarrollo de prototipos es un excelente método de trabajo, teniendo en mente que difiere de los otros modelos en que el objetivo del prototipo no es crear un producto, sino, enfatizar y demostrar ciertas propiedades del sistema deseado.

MODELO DE ESPIRAL

La gente inevitablemente crece, envejece y muere, pero el software puede ser rejuvenecido. En lugar de ver la vida del software como un proceso de una línea con una entrada, varias fases y una salida, se utilizará la palabra "ciclo".

En este modelo, el ciclo de vida del software empieza con la especificación de requerimientos. Después el software crece en el análisis, diseño, es implementado y luego probado. Si pasa las pruebas, el software se gradúa a nivel de producto. Una vez hecho esto, adquiere usuarios y los usuarios encuentran errores ("bugs"), esto debe de ser mantenido. Cuando los usuarios piden más, entonces debe de ser extendido. Crear y desarrollar versiones implica un progreso una vez más a través de las fases especificación, diseño, implantación y prueba.

Considerando el promedio de los proyectos de software; esto es, la vida del software hasta la graduación, la programación orientada a procedimientos o tradicional o procedural se enfoca en cómo implementar siguiendo una espiral como se muestra a continuación.

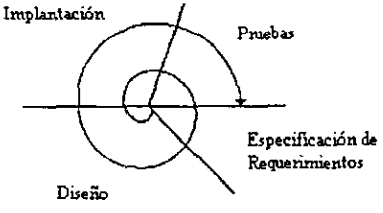


Figura 1.6. Ciclo de vida del software tradicional

Tradicionalmente una mayor parte de la vida del proyecto es gastada en la implantación.

Por otro lado el desarrollo orientado a objetos tiende a un software robusto que pueda ser fácilmente reutilizado, refinado, probado, mantenido y extendido, de este modo se asegura una madurez productiva.

El software es reutilizado cuando es usado como una parte de otro software para el cual ha sido diseñado. El software es refinado cuando es usado como las bases para la definición de otro software. El software es probado cuando su comportamiento está determinado o no para cumplir la especificación del software. El software es mantenido cuando son encontrados errores y corregidos. Por último el software es extendido cuando una nueva funcionalidad es adicionada a un programa existente.

La espiral de un software orientado a objetos debe verse como se muestra en la figura siguiente:

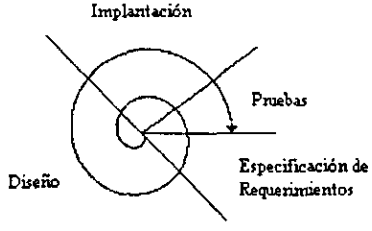


Figura 1.6. Ciclo de vida del software orientado a objetos

Esta espiral claramente muestra que una gran parte del tiempo es gastado en la fase de diseño. Una fracción más pequeña en implantación y prueba.

Una gran parte del tiempo es gastada en el diseño debido a que el software es diseñado para su fácil reutilización, mantenimiento y modificación.

Un gran esfuerzo es requerido para diseñar código reutilizable, pero es recompensado al final. El tiempo gastado en hacer un cuidadoso diseño permite el entendimiento del problema más profundamente. La implantación se realiza más rápido debido a que se ha entendido a fondo la distribución del problema. Por lo tanto, el tiempo total requerido para un ciclo puede no cambiar o verse disminuido. Y debido a que el software ha sido diseñado desde el principio teniendo en mente el mantenimiento, extensión, y reutilización, el tiempo requerido para cualquier cosa que se quiera realizar se reduce radicalmente.

2. EL DESARROLLO ORIENTADO A OBJETOS

La orientación a objetos es una técnica para el modelado de sistemas. Usando la orientación a objetos como una base, modelaremos los sistemas como un número de objetos que interactúan entre sí. Un modelo que es diseñado usando tecnologia orientada a objetos es fácil de entender, y puede estar directamente relacionado con la realidad. Además, con este método de diseño sólo va a existir una pequeña brecha semántica entre la realidad y el modelo.

El interés en el método de orientación a objetos ha crecido rápidamente en los últimos años. Esto es principalmente por el hecho que ha mostrado muy buenas cualidades. Las principales cualidades del diseño de sistemas con el método orientado a objetos son las siguientes:

- *Entendimiento*. El entendimiento del sistema es sencillo, así como la brecha entre el analista y el usuario, análisis y diseño.
- *Modularidad*. Las modificaciones del modelo tienden a ser locales, muchas de ellas resultan de un elemento individual, que es representado por un único objeto.

La programación estructurada[6] y la programación orientada a objetos, son distintas porque representan formas distintas de ver un mismo conocimiento. Sus enfoques cambian en cuanto a la forma de como se abstraen los datos que se representarán en forma de un programa.

La programación estructurada abstrae las formas de control más comunes de los programas, está orientada a los procedimientos, es decir, ¿cómo? se solucionará un problema, no contempla el manejo que recibirán los datos.

La programación orientada a objetos, por otra parte, no se preocupa por el ¿cómo se implantará un módulo?, sino ¿sobre quién se efectuará un procedimiento?, es decir el ¿qué? hará cierto módulo. La programación orientada a objetos representa una forma totalmente distinta de ver la computación, podemos decir que sus principios están cimentados en la programación estructurada, pero con unos refinamientos y características que la hacen diferente.

Podemos decir que la programación orientada a objetos y la programación estructurada sólo son formas distintas de ver el diseño e implantación de un programa. Pero ambas son importantes ya que para descomponer un problema es necesario descomponerlo estructuralmente para poder enfatizar el ordenamiento de eventos, pero también es necesario saber quienes serán los agentes que de una u otra forma serán causa de acción donde operarán los eventos.

[6] Ernesto Peñaloza Romero, APUNTES, FUNDAMENTOS DE PROGRAMACIÓN, 2a edición, UNAM, ENEP Aragón, 1996, capítulo I

2.1 COMPONENTES DEL DESARROLLO ORIENTADO A OBJETOS

2.1.1. OBJETO

El desarrollo orientado a objetos, como ya se mencionó, se pregunta primero acerca del objetivo del programa, esto es, el ¿Qué? no el ¿Cómo?, como se hace en la programación estructurada. El objetivo es encontrar los objetos y sus conexiones; para hacer eso es necesario ver que operaciones se necesitan desarrollar y que información resulta de esos objetos. El diseño orientado a objetos descompone un sistema en entidades que conocen como jugar sus roles dentro del sistema; ellos saben como ser ellos mismos.

El primer y más importante concepto es el de objeto. ¿Qué es un objeto?. La palabra objeto es usada en casi todos los contextos. Existen varias definiciones de objeto:

"Un objeto es caracterizado por un número de operaciones y un estado que recuerda el efecto de esas operaciones."[7]

"Un objeto es una entidad que tiene una función especifica de la cual es responsable, y la información necesaria para cumplir con esa tarea. Un objeto posee estados, conducta e identidad."[8]

"Un objeto es una entidad lógica que contiene datos y código que manipula esos datos."[9]

"Es una entidad representada por propiedades, valores de las propiedades y comportamientos."[10]

"Un objeto tiene estado, comportamiento e identidad; la estructura y comportamiento de objetos similares están definidos en su clase común; los términos instancia y objeto son intercambiables."[11]

"Un objeto representa un elemento, unidad o entidad individual e identificable, ya sea real o abstracta, con un papel bien definido en el dominio del problema"[12]

Nosotros diremos que un objeto es una entidad capaz de almacenar información que nos proporciona un estado y que ofrece un conjunto de operaciones o comportamiento para examinar o afectar dicho estado.

---

[7] Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998, pág. 44

[8] Ernesto Peñaloza Romero, APUNTES, FUNDAMENTOS DE PROGRAMACIÓN, 2a edición, UNAM, ENEP Aragón, 1996, pág. 227

[9] Herbert Schildt, TURBO C/C++ 3.1 MANUAL DE REFERENCIA, traducido de la 2a edición en inglés, McGraw-Hill, Inc., España, 1994, pág. 665

[10] BUILDING OBJECT-ORIENTED APPLICATIONS IN POWER BUILDER STUDENT GUIDE, Sybase Inc, Powersoft, nov 1996, pág 2-2.

[11] Grady Booch, ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS CON APLICACIONES, 2ª edición en español, Addison-Wesley/Diaz de Santos 1996, pág. 97

[12] Smith M. and Tockey, S, AN INTEGRATED APPROACH TO SOFTWARE REQUIEREMENTS DEFINITION USING OBJECTS, Seattle WA, Boeing Commercial Airplane Support Division, pág. 132

Un modelo orientado a objetos consiste en un número de objetos; éstos son partes claramente delimitadas del sistema modelado. Los objetos usualmente corresponden a entidades de la vida real, como una factura, un carro, una casa, etc.

Podemos decir que un objeto tiene dos partes, una *pública* y una *privada.*

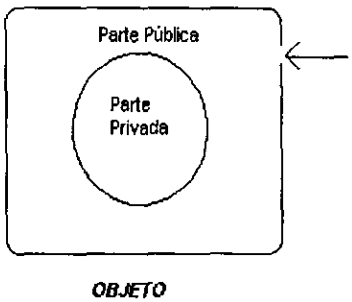


Figura 2.1. Esquema de un objeto con sus partes pública y privada

En general se habla de un comportamiento cuando una entidad reacciona ante un estímulo de una manera determinada. Un objeto responderá ante el mundo exterior comportándose de una forma determinada que será función del estado del objeto y del mensaje que reciba de ese mundo exterior. De esta forma la conducta de un objeto estará determinada por el conjunto de funciones que conforman dicho objeto. A estas funciones también se les denomina métodos del objeto.

Los atributos de un objeto son variables las cuales nos indican el estado del objeto en un cierto momento. Estos atributos sólo son manejados por el mismo objeto directamente.

Cada objeto dentro de un sistema debe de tener una responsabilidad bien establecida, y su conducta estará basada en las especificaciones que determinan su responsabilidad. Además el objeto deberá tener toda la información necesaria para cumplir con dichas responsabilidades.

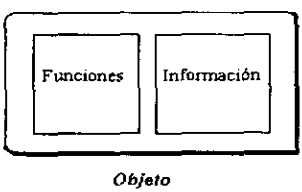


Figura 2.2. Funciones e información en un objeto

ENCAPSULAMIENTO

Un objeto sabe cómo ser el mismo, pero no sabe cómo ser ningún otro, algunos datos en el sistema residen en unos objetos y otros en otros objetos, por supuesto, algunos de los objetos pueden desarrollar unas funciones y otros objetos algunas otras. El diseño en orientado a objetos estructura responsabilidades preguntando ¿Qué puede hacer este objeto?, ¿Qué sabe este objeto?.

Un objeto debe de poseer una forma interna de comunicación y una externa, es decir un objeto posee una interfaz pública con la cual se puede comunicar con otros objetos, de esta forma un objeto que no posea las funciones e información para cumplir con su tarea puede comunicarse con otros que le ayudarán a cumplir su función. Por otra parte el objeto posee una información privada que oculta al resto de los objetos y que sólo a él le es de utilidad para cumplir con su responsabilidad.

A esto se le llama encapsulamiento, esto es, sólo mostrar la información que se requiere para el uso del objeto. El encapsulamiento maneja la complejidad relativa a los problemas del mundo real distribuyendo esa complejidad a objetos individuales y pertinentes. El encapsulamiento nos permite tratar al objeto como una entidad independiente y única de más sencilla manipulación (una caja negra).

A continuación se mencionan algunas definiciones de encapsulamiento:

"Toda la información en un sistema orientado a objetos es almacenado en objetos y solamente puede ser manipulado cuando se les ordena a los objetos desarrollar operaciones. El comportamiento y la información son encapsuladas en el objeto"[13]

"La acción de agrupar en un sólo objeto datos y operaciones que afecten estos datos es conocida como encapsulación."[14]

"Agrupar a las funciones e información en una entidad conocida como objeto es encapsular."[15]

"El proceso de introducir en el mismo comportamiento los elementos de una abstracción que constituyen su estructura y comportamiento; el encapsulameinto sirve para separar el

[13] Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998, pág. 49

[14] R. Wirfs-Brock / B. Wilkerson / L. Wiener, DESIGNING OBJECT-ORIENTED SOFTWARE, 4th printing , Prentice-Hall, Englewood, NJ,1990, pág. 6

[15] Ernesto Peñaloza Romero, APUNTES, FUNDAMENTOS DE PROGRAMACIÓN, 2a edición, UNAM, ENEP Aragón, 1996, pág. 227

interfaz contractual de una abstracción y su implantación.[16]

El conocimiento encapsulado en un objeto puede ser oculto para una vista externa. Como consecuencia, el conocimiento encapsulado dentro de un objeto parece diferente de afuera de ese objeto que de adentro. El objeto tiene una cara pública que presenta a las otras entidades dentro del sistema. La cara pública consiste en "que" puede hacer y decir. De esta forma otras entidades conocen sólo como pueden interactuar con él.

Los objetos también tienen su lado privado. Este es el "como" hacen esas cosas, y lo puede hacer en cualquier forma requerida. Como desarrolla las operaciones o computa la información no es concerniente a las otras partes del sistema. Este principio es conocido como ocultamiento de información. Utilizándolo, los objetos son libres de cambiar sus lados privados sin afectar el resto del sistema.

"Es un concepto fundamental en la orientado a objetos. Formalmente es el proceso de esconder todos los detalles de las propiedades y comportamientos de un objeto, que no se requiere para su uso."[17]

"Proceso de ocultar todos los secretos de un objeto que no contribuyen a sus características esenciales; típicamente, la estructura de un objeto está oculta, así como la implantación de sus métodos."[18]

Toda la información en un sistema orientado a objetos es almacenada en sus objetos y sólo puede ser manipulada cuando los objetos sean ordenados a desarrollar operaciones.

Los objetos soportan entonces el concepto de ocultamiento de información. Para usar un objeto no necesitamos saber como es representada la información o comportamiento internamente. Sólo necesitamos saber que operaciones ofrece.

Una de las propiedades importantes del encapsulamiento es que cada clase protege sus datos (atributos) de que sean cambiadas por otros objetos.

En la programación tradicional es deseable que un módulo se comunique con los demás sólo con los datos estrictamente necesarios, esto también existe en la programación orientada a objetos, con esto se logra una mayor flexibilidad en los componentes del sistema. En la programación tradicional un módulo debía ser accesado a través de una lista de parámetros, en la programación orientada a objetos se accesa a través de un mensaje, que es una forma alterna de enviar parámetros.

[16] Grady Booch, ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS CON APLICACIONES, 2ª edición en español, Addison-Wesley/Diaz de Santos 1996, pág. 564

[17] BUILDING OBJECT-ORIENTED APPLICATIONS IN POWER BUILDER STUDENT GUIDE, Sybase Inc, Powersoft, nov 1996, pág 7-2.

[18] Grady Booch, ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS CON APLICACIONES, 2ª edición en español, Addison-Wesley/Diaz de Santos 1996, pág. 567

Dentro de los beneficios de al encapsulamiento podemos mencionar:

- Previene cambios inesperados a los valores de las propiedades durante el proceso.
- Los cambios pueden ser hechos al proceso interno sin afectar los resultados, tanto como la interfaz no sea cambiada.
- Usar los objetos no requiere conocimiento de los métodos internos.

IDENTIDAD DE UN OBJETO

Agrupar a las funciones y la información en una entidad conocida como objeto es encapsular la identidad del objeto dentro de límites bien establecidos. Estos límites le dan la identidad necesaria para ser distinto de cualquier otro objeto.

Cada objeto es distinto de todos los demás objetos existentes en un momento dado el estado de sus variables, los mensajes recibidos y la manera como reaccionará a un mensaje le da su propia identidad.

"La identidad es aquella propiedad de un objeto que lo distingue de todos los demás objetos"[19]

"La naturaleza de un objeto que lo distingue de los demás"[20]

COMUNICACIÓN ENTRE OBJETOS

Los objetos del modelo tienen relación con uno u otro objeto. Estas relaciones pueden ser de dos tipos: *relaciones estáticas*, es decir, relaciones que existen durante un largo periodo de tiempo, lo que significa que dos objetos saben de la existencia del otro. Un objeto puede estar compuesto por otros objetos y esto está descrito con estas relaciones. Y *relaciones dinámicas*, es decir, relaciones por las cuales dos objetos se comunican entre sí.

La dinámica en un modelo orientado a objetos es creada a través de las relaciones dinámicas, esto quiere decir que los objetos mandan estímulo(s) a otro(s) objeto(s). Aquí estimulo significa el evento de un objeto comunicándose con otro.

[19] Khoshafian, S. and Copeland, G., OBJECT IDENTITY, SIGPLAN Notices, vol. 21(11), pág. 406, noviembre 1996 citado por Grady Booch, ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS CON APLICACIONES, 2ª edición en español, Addison-Wesley/Diaz de Santos 1996, pág. 106

[20] Grady Booch, ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS CON APLICACIONES, 2ª edición en español, Addison-Wesley/Diaz de Santos 1996, pág. 565

Los mensajes o estímulos son el vehículo mediante el cual serán iniciadas las actividades en la orientación a objetos.

"En programación orientada a objetos la acción es iniciada por la transmisión de un mensaje a un agente (objeto) responsable de la acción. El mensaje recibe la solicitud para una acción, y es acompañado por información adicional (argumentos) necesarios para llevar a cabo la solicitud. El receptor es el agente al cual es enviado el mensaje. Si el receptor acepta el mensaje, acepta también la responsabilidad de llevar a cabo la acción indicada. En respuesta a un mensaje, el receptor va a desarrollar un método para satisfacer la solicitud."[21]

El mensaje está compuesto de la siguiente forma:

mensaje : (destino, operación, argumentos)

donde el destino es el objeto al cual va dirigido, la operación es la operación que realizará el objeto receptor, los argumentos son opcionales dependiendo si los necesita o no la operación.

Un mensaje y una llamada a un procedimiento tienen en común que en los dos hay implícita una solicitud de una acción y son un conjunto de pasos bien definidos que se iniciarán de acuerdo a la solicitud. Sin embargo hay dos diferencias importantes, la primera es que en el mensaje hay un receptor designado para el mensaje; el receptor es un agente al cual el mensaje es enviado. En una llamada a un procedimiento no existe un receptor designado. La segunda diferencia es la interpretación del mensaje (esto es el método usado para responder al mensaje) es dependiente del receptor y puede variar con diferentes receptores.

Es importante que el comportamiento está estructurado en términos de responsabilidades. Una solicitud de acción indica sólo el deseo de realizar algo, el receptor del mensaje tiene la responsabilidad de llevarlo a cabo utilizando la técnica que mejor le parezca sin intervención externa.

2.1.2 CLASE

Es lógico pensar que pueden existir entre objetos algunos que tengan un comportamiento similar. Para describir todos los objetos que tienen un comportamiento y estructuras de información similares, se puede definir una clase que represente a esos objetos o que asocia características comunes de esos objetos. Por lo tanto podemos decir que una clase es una definición, una plantilla o un molde que permite la creación de nuevos objetos y es, por lo tanto,

[21] Timothy Budd, AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING, edición con correcciones, Addison-Wesley Publishing Company, Oregon State University, 1991,pág. 4

una descripción de características comunes de ciertos objetos. Los objetos que pertenecen a una clase tienen una plantilla en común.

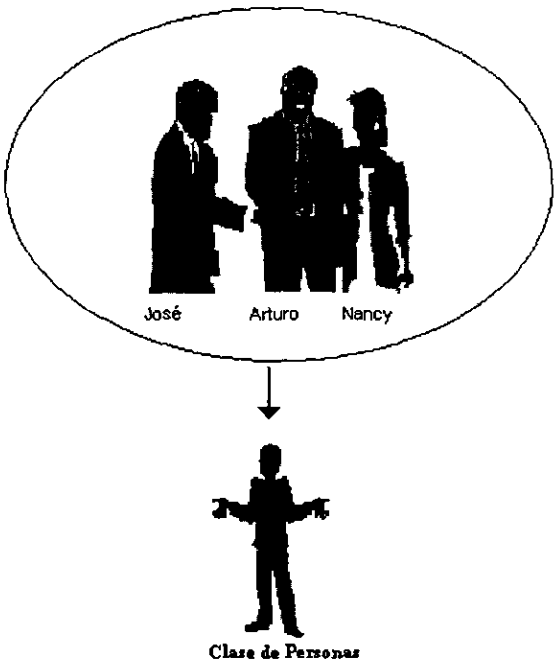


Figura 2.3. Clase

A continuación se enuncian algunas definiciones de clase citadas por varios autores:

"*Una clase representa una plantilla de ciertos objetos y describe cómo esos objetos son estructurados internamente. Los objetos de la misma clase tienen la misma definición para sus operaciones y para sus estructuras de información.*"[22]

"*Es una definición o caracterización de algo, es como una estructura, cada objeto que pueda ser caracterizado por esta definición es llamado instancia de la clase. Puede haber múltiples instancias de una clase. Cada instancia u ocurrencia tiene el mismo conjunto básico de características o propiedades, pero los valores pueden ser diferentes.*"[23]

[22] Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998, pág. 50

[23] BUILDING OBJECT-ORIENTED APPLICATIONS IN POWER BUILDER STUDENT GUIDE, Sybase Inc, Powersoft, nov 1996, pág 2-2.

"Una clase es un conjunto de objetos que comparten una estructura común y un comportamiento común"[24]

En sistemas orientados a objetos, cada objeto pertenece a una clase. Un objeto que pertenece a cierta clase es llamado instancia de la clase. Por lo tanto usaremos objeto e instancia como sinónimos.

"Una instancia es un objeto creado de una clase. La clase describe la estructura de la instancia (comportamiento e información), mientras el estado actual de la instancia es definido por las operaciones desarrolladas en la instancia."[25]

El comportamiento de la instancia y su estructura de información son, entonces, definidas por su clase. Cada instancia tiene una única identidad.

Una instancia de una clase es creada a tiempo de ejecución. En ese momento, es reservado espacio en memoria para sus propiedades y valores, y los métodos del objeto quedan disponibles. Al proceso de crear una instancia de la definición de su clase se le llama instanciamiento.

Todos los objetos son instancias de una clase. El método invocado por un objeto en respuesta a un mensaje es determinado por su clase. Todos los objetos de una clase dada usan el mismo método en respuesta a un mismo mensaje.

DESCUBRIENDO LAS CLASES

Una de las primeras decisiones que deben de ser hechas cuando se crean aplicaciones orientadas a objetos es la selección de las clases. Las clases pueden tener diferentes tipos de responsabilidades. Esto da lugar a diferentes tipos de clases, las siguientes categorías cubren la mayoría de los tipos de clases:

CLASE MANEJADORA DE DATOS, O DE ESTADO. Estas clases tienen como principal responsabilidad el mantenimiento de datos o información de estado de alguna forma. Estas clases muchas veces son reconocidas con los nombres en una descripción de un problema y son usualmente bloques fundamentales en el diseño.

CLASE FUENTES DE DATOS. Estas clases son las que generan datos o aceptan datos y luego los procesan como una clase que desarrolla salidas a disco o archivos. Este tipo de clases no detienen o almacenan datos por un periodo de tiempo pero generan una demanda de ellos o los procesan cuando se tienen que vaciar en un dispositivo de almacenamiento de datos.

[24] Grady Booch, ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS CON APLICACIONES, 2ª edición en español, Addison-Wesley/Diaz de Santos 1996, pág. 120

[25] Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998, pág. 50

CLASES DE VISTAS O DE OBSERVADORES. Una parte importante de la mayoría de las aplicaciones es el desplegado de la información en un dispositivo de salida como una terminal. Es bueno mantener el comportamiento del desplegado en clases separadas de las que mantienen y/o manejan los datos.

VISIBILIDAD Y DEPENDENCIA

Se sabe que la naturaleza de la interconectividad de software creado de la forma tradicional es el mayor obstáculo en la realización de componentes de software reutilizables. Esto es bien conocido y se han creado varios maneras de caracterizar la naturaleza de las conexiones y reglas para evitar interconexiones dañinas

Las interconexiones las podemos expresar en términos de *visibilidad* y *dependencia.* El término visibilidad es usado para describir una caracterización de los nombres, un nombre es mediante el cual los objetos o identificadores son accesados. Un objeto es visible en cierto contexto si el nombre del objeto es legal y denota al objeto. Un término frecuentemente usado para descubrir la visibilidad es el alcance ("*scope*") de una variable.

La visibilidad está relacionada con la interconectividad en el sentido de que es posible controlar y reducir la visibilidad de los nombres de un identificador, de esta forma se puede de manera más sencilla caracterizar como está siendo utilizado el identificador.

La visibilidad es una forma de protección y ocultamiento de datos y métodos ya que se pueden tener datos y métodos que sean privados o públicos, esto se refiere a que si son privados sólo podrán ser accesados a través de métodos asociados con la clase. Y si son públicos pueden ser accesados por cualquier objeto.

Es recomendable que los datos siempre sean privados ya que se garantiza que sólo se podrán modificar por métodos asociados con la clase y crear una interfaz pública con los métodos necesarios y suficientes para poder trabajar con una clase. De esta forma se pueden identificar finalmente posibles errores además de que se limita la interconectividad.

El concepto de dependencia es usado para relacionar una porción de software de un sistema con otro. Si un sistema de software (como una clase o un módulo) no puede existir de ninguna manera sin otro sistema, entonces el primer sistema es dependiente del segundo. Una clase hija es siempre dependiente de la clase padre.

ACOPLAMIENTO Y COHESIÓN

Con respecto a la dependencia existen dos conceptos relevantes estos son acoplamiento y cohesión, estos conceptos fueron introducidos por Stevens, Constantine y Myers[26] para evaluar el uso efectivo de los módulos, pero son válidos para los objetos.

El acoplamiento describe las relaciones entre módulos y la cohesión describe las relaciones dentro del módulo. Una reducción de interconexiones entre módulos es lograda mediante la reducción del acoplamiento. Por otro lado, los módulos que están bien diseñados, deben de tener un propósito y todos los elementos deben ser asociados para realizar una sola tarea. Así en un buen diseño, los elementos dentro de un módulo deben de tener una cohesión interna.

VARIEDADES DE ACOPLAMIENTO

El acoplamiento entre módulos puede surgir por diferentes razones algunas de las cuales son más deseables que otras, a continuación se enuncian algunos tipos de acoplamiento en orden de menos a más deseables.

ACOPLAMIENTO INTERNO DE DATOS. Este tipo ocurre cuando un módulo se le permite o puede modificar los valores de los datos locales (variables de instancia) en otro módulo. Esta organización de la actividad hace complicado el entendimiento y razonamiento de los programas y debe ser evitado lo más posible.

ACOPLAMIENTO GLOBAL DE DATOS. Este ocurre cuando dos o más módulos están ligados porque comparten las estructuras de datos globales. Esta situación frecuentemente complica el entendimiento de los módulos al analizarlos separadamente pero muchas veces no se puede evitar. En un sistema de OO una alternativa para este tipo de acoplamiento es hacer una nueva clase la cual se encarga del manejo de valores de datos de tal forma que todo el acceso a los valores globales es de esta manera (esta aproximación es similar al uso de funciones de acceso para tener acceso a los datos locales en un objeto). La técnica reduce el acoplamiento global de datos a acoplamiento paramétrico que es más fácil de entender y manejar.

ACOPLAMIENTO DE SECUENCIA O DE CONTROL. Este ocurre cuando un módulo debe llevar a cabo ciertas acciones en cierto orden pero el orden es controlado por otro módulo. Por ejemplo un sistema de base de datos puede ir a través de etapas en orden, llevar a cabo una

[26] W. Stevens, G. Myers and L. Constantine STRUCTURED DESIGN, IBM Systems Journal, 13(2), 1074; Reprinted in Edward Yourdon (De) Classics in Software Enginnering, Prentice-Hall, Englewood Cliffs, NJ, 1979 citado en Timothy Budd, AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING, edición con correcciones, Addison-Wesley Publishing Company, Oregon State University, abril 1991 ,pág. 216

inicialización, leer registros, actualizar registros, borrar registros, y generar reportes; sin embargo, cada etapa puede ser invocada por una rutina diferente y la secuencia de las llamadas dependerá del código en cada diferente módulo. La presencia del acoplamiento de control indica que el diseñador de un módulo estuvo siguiendo un nivel de abstracción más bajo del necesario (cada uno de los pasos a seguir en vez de un proceso de base de datos). Aún así cuando este tipo de acoplamiento no se puede evitar, es prudente que el mismo módulo se asegure que se han realizado los pasos correctos y en el orden correcto con un manejo apropiado de las llamadas a los procesos.

ACOPLAMIENTO PARAMÉTRICO. Este ocurre cuando un módulo debe invocar servicios y rutinas de otro y la única relación que hay es el número y el tipo de parámetros enviados y el valor regresado. Esta forma de acoplamiento es común, fácil de ver y fácil de identificar. Es la mejor opción.

ACOPLAMIENTO DE SUBCLASES. Este es particular para la orientación a objetos. Describe la relación de una clase con su clase padre o superclase. A través de la acción de la herencia, una instancia de una clase hija puede ser tratada como si fuera una instancia de la clase padre

VARIEDADES DE COHESIÓN

La cohesión interna de un módulo (o clase) es una medida del grado de ligado de varios elementos en el módulo (o clase). Se describen a continuación algunas formas de cohesión en el mismo sentido que las de acoplamiento de menos a más deseables.

COHESIÓN COINCIDENTAL. Ocurre cuando elementos de un módulo son agrupados sin aparente razón. Este agrupamiento es muchas veces resultado de modularizar un programa de una sola parte, arbitrariamente produciendo varios módulos pequeños. Es usualmente un signo de un diseño pobre. En un sistema orientado a objetos se dice que la cohesión coincidental ocurre cuando una clase consiste en métodos que no están relacionados.

COHESIÓN TEMPORAL. Este tipo de cohesión ocurre cuando los elementos son ligados debido a que serán utilizados aproximadamente al mismo tiempo. Un módulo que desarrolla la inicialización de un programa es un típico ejemplo. Aquí un mejor diseño probablemente podría distribuir las diferentes actividades de inicialización sobre los módulos que tengan el comportamiento subsecuente más cercano.

COHESIÓN DE COMUNICACIÓN. Ocurre cuando elementos de un módulo, métodos en una clase son agrupados porque accesan al mismo o mismos dispositivos de entrada y/o salida o mismos datos. El módulo o clase está actuando como administrador de datos o dispositivos.

COHESIÓN SECUENCIAL. Ocurre cuando elementos en un módulo son ligados por la necesidad de ser activados en un orden secuencial particular. La cohesión secuencial muchas veces ocurre como resultado de un intento de evitar el acoplamiento secuencial. Otra vez un mejor diseño usualmente puede encontrar si el nivel de abstracciones es el correcto. (Por supuesto si es necesario para desarrollar acciones en cierto orden, esta secuencialidad debe ser expresada en algún nivel de abstracción, pero el principio importante es esconder esta necesidad lo más que se pueda a los otros niveles de abstracción).

COHESIÓN LÓGICA. Ocurre cuando hay una lógica conexión entre los elementos de un módulo, pero no una conexión en términos de cualquier dato o control. Una librería de funciones matemáticas (seno, coseno,..) exhibe o tiene una cohesión de este tipo si cada una de sus funciones son implementadas separadamente sin referencia de algunas de las otras.

COHESIÓN DE FUNCIÓN. Es un deseable tipo de ligamiento en el cual los elementos de un módulo o los métodos en una clase están relacionados para el desarrollo de una función.

COHESIÓN DE DATOS. Esta ocurre cuando en un módulo se define internamente un conjunto de valores de datos, y exporta rutinas que manipulan la estructura de datos. La cohesión de datos ocurre cuando un módulo es usado para implementar una abstracción.

"También se puede muchas veces estimar el grado de cohesión de un módulo escribiendo un pequeño enunciado del propósito del módulo y examinando este enunciado. Las siguientes pruebas son sugeridas por Constantine.

1. Si la sentencia que describe el propósito de un módulo es una sentencia compuesta, conteniendo una coma o conteniendo más de un verbo, el módulo probablemente está desarrollando más de una función. Entonces probablemente tenga una liga de comunicación o secuencial.

2. Si la sentencia contiene palabras relacionadas con el tiempo como "primero", "siguiente", "entonces", "después", "cuando" o "inicial", entonces el módulo probablemente tenga ligas secuenciales o temporales. Un ejemplo puede ser "espera a que se introduzca un valor y entonces pone los demás valores".

3. Si el predicado de la sentencia no contiene un objeto específico seguido del verbo, el módulo está probablemente ligado lógicamente. Por ejemplo "Edita todos los datos" tiene un ligado lógico, "edita los datos fuente" puede tener un ligado funcional.

4. Si la sentencia contiene palabras como "inicializar", "borrar", el módulo probablemente tenga ligado temporal."[27]

Esta forma de evaluar no es muy precisa, pero sin embargo nos puede proporcionar una idea sobre lo que estamos haciendo.

Los conceptos de acoplamiento y cohesión tienen su máxima vigencia cuando se trata con objetos ya que estos sólo se pueden comunicar entre sí por medio de mensajes. Cada objeto o servidor ocultará a su cliente la manera como realiza su trabajo mostrando sólo una interfaz pública.

[27] Timothy Budd, AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING, edición con correcciones, Addison-Wesley Publishing Company, Oregon State University, abril 1991,pág. 218,219

2.1.3. HERENCIA

Podemos iniciar diciendo que la herencia es una útil herramienta de implantación pero no una técnica de desarrollo.

Existen dos relaciones que son fundamentales en el diseño orientado a objetos. Estas relaciones son conocidas como la relación "es un" o "es una" y la relación "tiene un" o "tiene una" (o parte de).

La relación "es un" esta dada entre dos conceptos cuando el primero es una instancia especializada del segundo. Esto es, para todos los procesos prácticos el comportamiento y los datos asociados con la idea más específica forman un superconjunto del comportamiento y datos asociados de una idea más abstracta. Para probar la relación "es un" consideremos la afirmación X "es un" Y, donde X y Y son conceptos conocidos. Si la afirmación es correcta, esto es, si concuerda con nuestro conocimiento, entonces se dice que X y Y tienen una relación "es un". Por ejemplo un carro "es un" vehículo, así, los hechos generales sobre los vehículos son aplicables a los carros.

La relación "tiene un" se aplica cuando el segundo concepto es componente del primero. Por ejemplo un carro "tiene un" motor, está claro que un carro nunca "es un" motor, tampoco que un motor "es un" carro, sin embargo, un carro "es un" vehículo que a su vez "es un" transporte.

Resumiendo podemos decir que la relación "es un" define jerarquías entre clases y subclases mientras que la relación "tiene un" define los datos que deben ser mantenidos por una clase.

Después de explicar las reglas anteriores. A continuación se mencionan algunas definiciones de herencia dadas por algunos autores para tratar de dejar en claro lo que es la herencia.

"Es la relación entre dos o más clases en la cual una definición de una clase incorpora todo o parte de la definición de otra clase debido a su propia definición."[28]

"Por herencia se entiende la propiedad en la cual las instancias de una subclase o clase hija pueden accesar a los métodos (comportamiento) asociados con una clase superior o superclase o clase padre. La herencia es transmitida, es decir una clase puede heredar características de superclases muchos niveles arriba.

[28] BUILDING OBJECT-ORIENTED APPLICATIONS IN POWER BUILDER STUDENT GUIDE, Sybase Inc, Powersoft, nov 1996, pág 3-2.

"Herencia significa que el comportamiento y los datos asociados con las clases hijas son siempre una extensión de las propiedades asociadas a una o unas superclases. Una subclase debe tener todas las propiedades de la clase superior y de otras subclases si es el caso. Por otro lado una clase hija es una forma más especializada o restringida de la clase padre, esto es, en cierta forma una contracción de la clase padre. Esta tensión entre extensión y contracción es una fuente de mucho poder de la técnica de OO, pero al mismo tiempo hay mucha confusión al usarla."[29]

" La herencia es el proceso por el cual un objeto puede adquirir las propiedades de otro objeto. Esto es importante porque permite manejar el concepto jerarquía de clases."[30]

Cuando describimos nuestras clases, es notable rápidamente que varias clases tienen características comunes (comportamiento y estructuras de información). Estas similitudes pueden ser compartidas entre las clases extrayéndolas y colocándolas en clases separadas. De esta forma las características comunes pueden ser compartidas por varias clases. Al recolectar las características comunes en una clase específica, dejamos que las clases originales se hereden de dicha clase; entonces sólo necesitamos describir las características que son específicas a las clases originales.

Si la clase "X" se hereda de la clase "Y", entonces las operaciones y la estructura de información descritas en la clase "Y" van a ser parte de la clase "X".

La única información restante es la que difiere entre las clases. La herencia es muy útil también para evitar redundancia, originando modelos pequeños que son fáciles de entender.

"Subclasificación: Es el proceso de crear nuevas clases descendientes resultando en una clasificación jerárquica"[31]

"Jerarquía de clases: Es un conjunto de clases relacionadas por herencia. Para crear una jerarquía de clases se empieza con una clase base, esta por lo general sólo contiene aquellas propiedades y métodos necesarios por todos los descendientes."[32]

Las clases que están abajo de otras clases en la herencia jerárquica son llamadas *descendientes* de la clase, las clases que están arriba se llaman *ancestros* o *antecesores*. Algunas veces los conceptos de *subclases* y *superclases* son utilizados como sinónimos.

Si una clase directamente hereda a otra, la llamamos *descendiente directo*, de esta forma la primer clase es el *ancestro directo* de la segunda clase. Un ancestro directo es llamado algunas veces también *padre* y al descendiente directo por supuesto *hijo*.

[29] Timothy Budd, AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING, edición con correcciones, Addison-Wesley Publishing Company, Oregon State University, abril 1991,pág. 85

[30] Herbert Schildt, TURBO C/C++ 3.1 MANUAL DE REFERENCIA, traducido de la 2a edición en inglés, McGraw-Hill, Inc., España, 1994, pág. 666

[31] BUILDING OBJECT-ORIENTED APPLICATIONS IN POWER BUILDER STUDENT GUIDE, Sybase Inc, Powersoft, nov 1996, pág 3-2.

[32] BUILDING OBJECT-ORIENTED APPLICATIONS IN POWER BUILDER STUDENT GUIDE, Sybase Inc, Powersoft, nov 1996, pág 3-8.

Existen tres tipos de clases en una jerarquía hereditaria:

- Clases base: Son originarias, o de nivel más alto de ancestros. Estas clases serán los ancestros comunes para todos los descendientes en una jerarquía de clases.

- Clases Abstractas: Son antecesores que adicionan funcionalidad a las clases base pero nunca se instancían. Ellas proveen un lugar para definir las propiedades y métodos comunes para las clases descendientes.

- Clases Concretas: Son las clases descendientes que deben ser instanciadas.

Las clases abstractas pueden tener:

- Valores de propiedades por *"defoult"*
- Eventos precodificados
- Comportamientos adicionales
- Propiedades adicionales

El instanciamiento de las clases concretas ocurre a tiempo de ejecución, en estas clases todos los comportamientos y atributos pueden ser heredados y extendidos.

ADICIONANDO REMPLAZANDO Y REFINANDO

Hasta aquí hemos supuesto que los métodos adicionados a una subclase y los heredados de una superclase son siempre distintos. Esto es que el conjunto de métodos y atributos definidos en la subclase no se encuentran en el conjunto de valores y métodos de la superclase. Para describir esto decimos que los métodos y valores son adicionados al protocolo de la superclase.

Una situación diferente ocurre cuando una subclase define un método con el mismo nombre que es usado para un método en la superclase, entonces el método en la subclase esconde o sobrescribe el método en la superclase esto es la sobrescritura. Cuando buscamos un método en una cierta situación, descubriremos el método en la subclase.

Cuando un mensaje es enviado a un objeto, la búsqueda del método siempre comienza examinando los métodos asociados con la clase del objeto. Si no se encuentra el método se busca en la superclase inmediata del objeto examinado, si no se vuelve a encontrar, se examina la superclase y así hasta que se encuentra el método o se reporta un error si se terminan las clases y no se encontró.

Un método definido en una subclase puede sobrescribir un método heredado con una de dos finalidades: el reemplazamiento del método es decir, remplaza el método de la superclase durante la ejecución, es decir, el código en la superclase nunca es ejecutado cuando se manipulan instancias de la subclase. El refinamiento del método, este significa que se conserva el comportamiento heredado ejecutándose el código de la superclase pero además es aumentado.

CUÁNDO SE PUEDEN O DEBEN HACER SUBCLASES

Es complicado cuando se está empezando a programar con la técnica de orientación a objetos, saber reconocer cuándo una clase debe ser una subclase de otra.

La regla fundamental para que una clase sea relacionada por herencia con otra, es que debe de existir una relación de funcionalidad entre las dos clases. Esto se describió con la regla "es un", "es una", esto es, si decimos que una subclase "es una" superclase, estamos diciendo que la funcionalidad y los datos que asociamos con la subclase forman un superconjunto de la funcionalidad y de los datos asociados con la clase padre. Sin embargo este sistema puede tener variaciones y a continuación se describen algunas situaciones comunes:

ESPECIALIZACIÓN. Por mucho el uso más común de la herencia y la subclasificación es para la especialización, la subclasificación por especialización es la más obvia y directa forma de la regla "es un", "es una", si se está considerando dos conceptos abstractos X y Y y la sentencia "X es un Y" hace que probablemente sea correcto hacer X una subclase de Y.

Por otro lado si la sentencia "X tiene un Y" tiene más sentido, se debe de considerar a Y como un atributo de X ya que nunca se cumplirá la regla "es un".

ESPECIFICACIÓN. Otro uso frecuente de la herencia es garantizar que las clases mantengan una interfaz común. Esto es que implementen los mismos métodos. La clase padre puede ser una combinación de operaciones implementadas y operaciones que son diferidas a las clases hijas. Muchas veces, no hay cambio de interfaz entre una subclase y una superclase pero la subclase debe de implementar el comportamiento descrito, pero no implementado en la clase padre.

Esto es actualmente un caso especial de subclasificación por especialización, excepto que las subclases no son un refinamiento de una clase ya existente, sino que se consideran realizaciones de una especificación abstracta incompleta.

La subclasificación por especificación puede ser reconocida cuando la superclase no implementa ningún comportamiento, pero describe el comportamiento que debería ser implementado en subclases.

GENERALIZACIÓN. Utilizar la herencia para realizar un subclasificación para una generalización es en cierta forma opuesto a la especialización. Aquí una subclase extiende el comportamiento de una superclase para crear una clase más general. Este tipo de subclasificación es muchas veces aplicables cuando se está construyendo en base a clases ya existentes que no se puedan modificar.

EXTENSIÓN. Cuando se hace una subclasificación por generalización se modifica o expande la funcionalidad existente de un objeto, la subclasificación por extensión incrementa totalmente nuevas habilidades.

La subclasificación por extensión puede ser distinguida de la subclasificación por generalización en que esta sobrescribe al menos un método del padre y la funcionalidad está ligada a la del padre, mientras, que la extensión sólo adiciona nuevos métodos a los del padre y la funcionalidad está menos ligada a los métodos existentes del padre.

LIMITACIÓN. La subclasificación por limitación es una forma más especializada de la especificación, donde el comportamiento de una subclase es menor o más restringido que el comportamiento de una superclase. Como la subclasificación por generalización, la subclasificación por limitación ocurre más frecuentemente cuando un programador construye en base a clases ya existentes que no pueden o no deben ser modificadas.

El programador puede hacer una subclase de una ya existente y modificar o sobrescribir los métodos no deseables de tal forma que produzcan un error si son usados. Esto es lo que caracteriza a la subclasificación por limitación, es decir, los métodos que han sido modificados o sobrescritos para que produzcan un error si son utilizados.

Hay que notar que la subclasificación por limitación es una explícita violación a la regla "*es un*", "*es una*" de la herencia por esto hay que usar esta técnica lo menos posible.

VARIACIÓN. La subclasificación por variación puede ser cuando dos o más clases tienen una implantación similar, pero no tienen a simple vista relación jerárquica. Por ejemplo El código necesario para controlar el "mouse" puede ser muy parecido al necesario para controlar el "joystick". Conceptualmente no existe ninguna razón particular para que uno sea subclase del otro. Entonces una de las dos clases puede ser seleccionada arbitrariamente como el padre con el código que sería heredado por la otra.

Muchas veces otra mejor alternativa es crear una clase con el código que usan los dos y heredar los dos de un antecesor común. Si se está trabajando con una base de clases existentes esto no se puede hacer.

COMBINACIÓN. Una situación común existe cuando una subclase representa una combinación de características de dos o más superclases es conocida como múltiple herencia y no es proporcionada por todos los lenguajes.

HERENCIA MÚLTIPLE

Cuando describimos una nueva clase, si queremos utilizar características de dos o más clases existentes, podemos heredar de esas clases. A esto se le llama herencia múltiple. Esto significa que una clase puede tener más de un ancestro directo.

La herencia múltiple es muchas veces justificada como una forma de proporcionar descripciones reutilizables. Si una descripción de una clase incluye comportamiento el cual parcialmente existe en otra clase, entonces este comportamiento puede ser extraído y de esta

forma podemos heredar comportamiento común de dos clases. Este concepto se ve más claro en un esquema, además de que para implementarlo el lenguaje debe de permitirlo. Ver la figura siguiente.

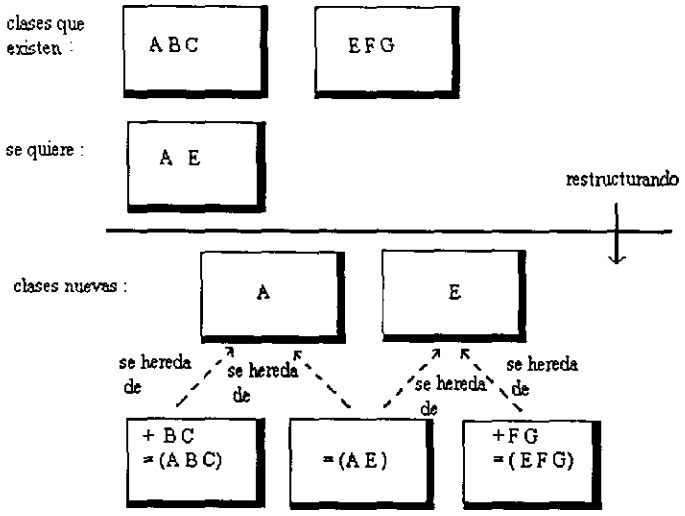


Figura 2.4. Herencia Múltiple

BENEFICIOS DE LA HERENCIA

Si se usa correctamente la herencia existen algunos beneficios. Aquí se describen algunos:

SOFTWARE REUTILIZABLE. Cuando el comportamiento es heredado de otra clase, el código que provee ese comportamiento no debe de ser sobrescrito. Esto puede parecer obvio pero tiene implicaciones importantes. Muchos programadores gastan un gran porcentaje de su tiempo escribiendo código que ya han escrito anteriormente. Usando programación orientada a objetos algún código puede ser escrito una vez y reutilizado.

Otros beneficios del código reutilizable es el incremento de confiabilidad ya que ese código que se probó para una clase es el mismo que se usará en una subclase. También contribuye a la más sencilla detección de errores.

CÓDIGO COMPARTIDO. Dentro de la programación orientada a objetos el compartimiento de código puede darse a varios niveles. En un nivel, algunos usuarios o proyectos separados pueden usar las mismas clases. Nos podemos referir a ellos como componentes de software. Otra forma de compartir código ocurre cuando dos o más clases diferentes desarrolladas por un programador como una parte de un proyecto son heredadas de una sola clase. Cuando esto sucede las dos o más diferentes tipos de objetos van a compartir el código que heredan. Este código tiene que ser escrito sólo una vez y sólo una vez contribuye al tamaño del programa.

CONSISTENCIA DE INTERFAZ. Cuando múltiples clases se heredan de la misma superclase, estamos seguros que el comportamiento que heredan va a ser el mismo en todas ellas. De esta forma es sencillo garantizar que las interfaces de objetos similares son de hecho similares y el usuario no presentará confusión de la colección de objetos que siempre es similar aunque el comportamiento sea diferente.

PROTOTIPO RÁPIDO. Cuando un sistema de software es construido utilizando componentes reutilizables, el tiempo de desarrollo puede concentrarse en entender la parte nueva del sistema. De esta forma los sistemas pueden de forma más rápida y fácil ir manteniendo un estilo de programación conocido como prototipo rápido o programación exploratoria. Un prototipo es producido de forma muy fácil, con este se experimenta de forma que el sistema definitivo está basado en la experiencia del primero.

OCULTAMIENTO DE LA INFORMACIÓN. Cuando un programador reutiliza un componente de software, sólo necesita conocer la naturaleza del componente y su interfaz, no es necesario para el programador tener detalle de la información concerniente a las técnicas usadas para desarrollar el componente. Esta es una forma de reducir la interconectividad entre los sistemas de software.

MODIFICACIÓN SIMPLE. A través de la herencia, también obtenemos otra ventaja. Si queremos modificar algunas características, es suficiente realizarlas en un lugar. De esta forma, si se hace esa modificación se va a heredar esa nueva definición. Así la herencia es muy útil para la fácil modificación de los modelos.

EL COSTO DE LA HERENCIA

Como se vio los beneficios de la herencia son bastantes, pero todo beneficio tiene un costo, de esta forma se considera el costo de la programación orientada a objetos y en particular del uso de la herencia.

VELOCIDAD DE EJECUCIÓN. Es claro que los métodos heredados en los que se debe de lidiar con varias subclases es más lento que el código especializado. Sin embargo la diferencia es muy pequeña y la mayoría de las veces es preferible. Por otro lado la reducción del tiempo de ejecución puede ser balanceada con el incremento en el desarrollo de software. Finalmente muchos programadores actualmente tienen una idea de ¿Cómo es utilizado el tiempo de ejecución?, esto es, que saben que parte de sus sistemas es crítica por lo tanto se podría optimizar o mejorar esas secciones.

TAMAÑO DEL PROGRAMA. Por lo general el uso de cualquier librería de software (clases ya desarrolladas) impone una multa o castigo sobre su uso en tamaño, sin embargo este gasto puede en algunos casos ser substancial, es cierto también que como el costo de la memoria decrece el tamaño del programa es menos importante.

COMPLEJIDAD DEL PROGRAMA. La técnica de programación orientada a objetos puede ser una solución al problema de la complejidad del software, sin embargo la herencia puede reemplazar un tipo de complejidad por otra. Entender el flujo de control de un programa que usa herencia puede ser muy complicado y puede requerir varios vistazos y análisis sobre el gráfico jerárquico.

2.1.4. POLIMORFISMO

El término polimorfismo viene del griego *poly = muchos, morphos = forma*. En lenguajes de programación un objeto polimórfico es una entidad como una variable o un argumento de una función, que permite almacenar valores de diferentes tipos durante la ejecución.

En el polimorfismo puro hay una función y un número de diferentes interpretaciones. El polimorfismo no puro ocurre cuando a una función se le pueden aplicar argumentos de diferentes tipos.

Instancias, creadas de las clases, nos proporcionan de manera conjunta un comportamiento dinámico que desearemos modelar. Es cuando estas instancias empiezan a comunicarse con los otros comportamientos del sistema que han sido desarrollados. Una instancia puede conocer de otras instancias que estímulos se van a enviar. Si una instancia manda un estímulo a otra instancia, pero no sabe a que clase pertenece la instancia receptora, decimos que tenemos polimorfismo.

Un estímulo puede ser interpretado en diferentes formas, dependiendo de la clase receptora. Muchas veces se dice que el polimorfismo significa que una operación puede ser implementada de diferentes formas en diferentes clases. Esto es actualmente sólo una consecuencia de lo que se ha dicho, y no, polimorfismo por sí mismo.

El polimorfismo es una característica muy importante para nuestros modelos. Es el receptor de un estímulo el que determina como debe de ser interpretado el estímulo, no el transmisor. El transmisor necesita saber sólo que otra instancia va a desarrollar un cierto comportamiento. Esta es una herramienta muy poderosa que nos permite desarrollar sistemas flexibles. De esta forma sólo debemos especificar que se debe de hacer no como se debe de hacer. De esta manera, a través de la delegación de qué debe ocurrir, se puede obtener un sistema flexible y resistente a los cambios. Si se requiere adicionar un objeto de una nueva clase, esta modificación va a afectar sólo al nuevo objeto, y no a los que les manden estímulos a él.

Se mencionan a continuación definiciones dadas por varios autores de polimorfismo.

"El polimorfismo, permite usar un nombre para varios propósitos relacionados pero ligeramente diferentes. La finalidad del polimorfismo es permitir el uso de un nombre para especificar una clase de acción general."[33]

"Polimorfismo significa que el emisor de un estimulo no necesita saber la clase de la instancia receptora. La instancia receptora puede pertenecer a una clase arbitraria."[34]

"Un concepto de teoría de tipos, de acuerdo con el cual un nombre (como una declaración de variable) puede denotar objetos de muchas clases diferentes que se relacionan mediante alguna superclase común; así todo objeto denotado por este nombre es capaz de responder a algún conjunto común de operaciones de diferentes modos"[35]

Polimorfismo significa que el transmisor de un estímulo no necesita saber la clase de la instancia receptora. El transmisor provee sólo una solicitud para un evento específico, mientras el receptor conoce cómo desarrollar este evento.

La liga del estímulo recibido con la operación apropiada a realizar es realizada ligando el estímulo con una operación. Si esta ligadura ocurre durante la compilación, se dice que es un *ligado estático*. La característica de polimorfismo hace imposible determinar a tiempo de compilación a que clase pertenece una instancia y por esto decidir que operación desarrollar. Esta información no va a estar disponible hasta el tiempo de ejecución, cuando esto por supuesto debe de ser conocido, esto se llama *ligado dinámico*. El ligado dinámico es muy flexible pero reduce el desempeño.

La ventaja con el ligado dinámico es el sistema flexible que se obtiene. Esta flexibilidad es de gran utilidad en sistemas que son regularmente modificados. Por no realizar un ligado antes de la ejecución, muchas de las modificaciones hechas no afectan al objeto transmisor.

El ligado estático, sin embargo es más seguro y eficiente ya que se pueden detectar errores a tiempo de compilación y esto evita errores a tiempo de ejecución. El ligado dinámico es una forma de implementar el polimorfismo.

Esto se ve más claro en un ejemplo. Suponiendo que se tiene una colección de clases que representan figuras geométricas, podemos definir un método para dibujar la forma en cada clase circulo, triángulo, cuadrado etc. Pero también podemos definir un método dibujar figura en la superclase forma geométrica, aunque no producirá ningún comportamiento útil ya que no tiene suficiente información para dibujar la cierta forma geométrica. Sin embargo la mera presencia del método en la superclase va a permitir asociar el concepto dibujar con la clase forma geométrica y no con las tres subclases separadamente.

[33] Herbert Schildt, TURBO C/C++ 3.1 MANUAL DE REFERENCIA, traducido de la 2a edición en inglés, McGraw-Hill, Inc., España, 1994, pág. 665

[34] Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998, pág. 55

[35] Grady Booch, ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS CON APLICACIONES, 2ª edición en español, Addison-Wesley/Diaz de Santos 1996, pág. 567

FORMAS DE POLIMORFISMO

SOBRECARGA (OVERLOADING)

Decimos que un nombre de una función es sobrecargado si hay dos o más cuerpos de funciones asociados con el mismo nombre. Nótese que la sobrecarga es una parte necesaria de la sobrescritura pero los términos no son idénticos y además puede existir sobrecarga sin sobrescritura.

En la sobrecarga es el nombre de la función el que es polimórfico, esto es tiene diferentes formas. Otra forma de pensar en la sobrecarga y el polimorfismo es que existe una función abstracta que toma varios tipos diferentes de argumentos, y el código a ejecutar depende de los argumentos dados, es decir, permiten que se comparta el nombre y se discriminan por el número y tipo de datos que se requieren. Este es llamado sobrecarga paramétrica.

La mayoría de los lenguajes de orientados a objetos permiten el estilo de sobrecarga, esto es la ocurrencia de métodos con nombres iguales en clases no relacionadas.

El mecanismo de sobrecarga es un requisito previo necesario para las otras formas de polimorfismo consideremos sobrescritura y métodos diferidos.

MÉTODOS DIFERIDOS

Un método diferido puede pensarse como una generalización de la sobrescritura, en ambos casos el comportamiento descrito en la superclase es modificado por una subclase. En un método diferido, sin embargo, el comportamiento en la superclase es esencialmente nulo, y toda la actividad útil es definida como parte de sobrescribir el método.

Una ventaja de los métodos diferidos es conceptual, en la cual permite al programador pensar en una actividad que se asocia a una abstracción, en un nivel más alto.

2.1.5. REUTILIZACIÓN DEL SOFTWARE

La encapsulación nos permite construir entidades independientes y dependientes de un comportamiento de cierta forma y conocimiento de cierta información. Dichas entidades pueden ser usadas en cada aplicación que pueda hacer uso de dicho comportamiento y conocimiento. Con un pensamiento cuidadoso es posible construir entidades que puedan ser útiles en muchas situaciones. Sin embargo, se requiere de un esfuerzo para diseñar entidades que sean generalmente útiles.

Hay tres tipos de tareas del diseño de software distinguidas por el producto del proceso. El diseño del software puede producir:

Componentes
"Frameworks"
Aplicaciones

COMPONENTES. Los componentes son entidades que pueden ser usadas en diferentes programas. El primer objetivo cuando se diseñan componentes es hacerlos generales, así pueden ser componentes de muchas aplicaciones. Para las aplicaciones que los usen deben de ser unas cajas negras. Los desarrolladores de aplicaciones que hagan uso de los componentes no necesitan conocer ni entender la implantación de esos componentes.

Los componentes son comúnmente "descubiertos" cuando los programadores encuentran piezas de códigos similares.

Cuando un programador toma el tiempo para sacar las características similares de varios elementos, juntarlas en uno creando una interfaz general y uniforme, nace un componente. Ultimamente los programadores tienden a abstraer la funcionalidad común cuando se designa una pieza de software antes de codificar piezas similares.

FRAMEWORKS. Son estructuras esqueléticas de programas que pueden ser utilizados para construir una aplicación completa.

El uso de "frameworks" permite que todas las aplicaciones que se desarrollan tengan un mismo esquema, es decir, que se vean y que se sientan igual, permitiendo a los programadores concentrase en los detalles de su aplicación.

El primer objetivo cuando diseñamos "frameworks" es hacerlos refinables. Las interfaces del resto de la aplicación deben de ser tan claras y precisas como sea posible. Los "frameworks" son cajas blancas que se puede hacer uso de ellos. Los desarrolladores deben de ser capaces de entender rápidamente la estructura del "framework", y cómo escribir código que pueda encajar en el "framework".

APLICACIONES. Las aplicaciones son programas completos. El principal objetivo cuando diseñamos aplicaciones es hacerlas fáciles de mantener. En este sentido el comportamiento de la aplicación puede conservarse apropiadamente y consistentemente durante su ciclo de vida. Las aplicaciones pueden extender ciertos componentes, o ampliar un "framework" de forma única

para la aplicación. Una aplicación es reutilizada principalmente cuando se va a extender su funcionamiento ya que se utiliza toda la aplicación como base y se agrega algo más.

Una aplicación debe de ser diseñada para que permita alteraciones frecuentes. La manera más racional de construir aplicaciones es juntar módulos cambiables de la aplicación, cada uno de los cuales es construido utilizando componentes.

Los módulos de la aplicación estarán bien definidos, de tal forma que si hay la necesidad de realizar un cambio, es muy probable que afecte sólo a un módulo y no a varios.

2.2. SISTEMAS ORIENTADOS A OBJETOS V.S. SISTEMAS ESTRUCTURADOS

Los métodos existentes para el desarrollo de sistemas pueden ser divididos básicamente en métodos *estructurados* y métodos *orientados a objetos*. Por métodos estructurados entendemos a los métodos que tratan a los datos y/o funciones más o menos separados. Los métodos orientados a objetos ven a los datos y las funciones altamente integrados.

Los sistemas estructurados entonces distinguen entre los datos y las funciones, en donde las funciones, en principio son activas y tienen comportamiento y los datos son unos almacenadores pasivos de información que son afectados por las funciones. El sistema es típicamente dividido en funciones mientras que los datos son enviados entre esas funciones.

Un sistema desarrollado utilizando métodos estructurados muchas veces llega a ser difícil de mantener. Un gran problema con métodos estructurados es que en principio todas las funciones deben de saber como son almacenados los datos, esto es la estructura de datos. En el caso que varios tipos de datos tienen diferentes formatos, implica que muchas veces se necesitarán cláusulas de condición que no tienen nada que ver con la funcionalidad pero relacionan a los diferentes tipos de datos. Entonces los programas llegan a ser difíciles de leer. De esta forma si se quiere cambiar la estructura de datos, debemos de modificar todas las funciones relacionadas con la estructura de datos. Los sistemas desarrollados con estos métodos son muy inestables ya que una pequeña modificación genera grandes consecuencias. En la misma forma la estabilidad del proceso de desarrollo no va a ser perdurable, ya que cualquier modificación va a crear consecuencias en otras áreas.

Otro problema con estos sistemas, es que la gente no piensan naturalmente en términos de lo que estructuran. La especificación de requerimientos es formada naturalmente en lenguaje humano. Esto muchas veces está descrito en términos de "qué" hará el sistema, que funcionalidad va a soportar el sistema y que elementos van a existir en el sistema. Esto es creado o transformado a términos "como" para una adecuación funcional cuando cambie el enfoque. En este sentido se crea una gran distancia semántica entre los puntos de vista internos y externos del sistema.

Esta es una de las razones por las cuales los sistemas estructurados son difíciles de modificar ya que son diseñados acerca del "como" un cierto comportamiento va a ser llevado a cabo (y esta es un área muy sensible a ser cambiada), y las modificaciones generan mayores consecuencias.

Los métodos orientados a objetos estructuran el sistema en los elementos que existen en el dominio del problema. Esto es muchas veces una manera más natural de describir el sistema, estos elementos son muy estables normalmente. Los cambios que llegan a ocurrir normalmente afectan a uno o a pocos elementos, lo que significa que los cambios en el sistema son locales usualmente.

El propósito del análisis orientado a objetos como el de los otros análisis, es obtener un entendimiento de la aplicación; un entendimiento dependiendo sólo de los requerimientos funcionales del sistema. La distancia semántica que puede existir entre los requerimientos y el interior del sistema es mínima ya que tanto como los requerimientos como el sistema están expresados en términos de que van a hacer. Debido a la vinculación de las partes exteriores e interiores del sistema descritas es posible que un sistema orientado a objetos se adapte más fácilmente a los cambios del mundo o del el problema modelado por el software.

La programación orientada a objetos promueve mecanismos para la reutilización como clases, objetos, herencia, instanciamiento, promoviendo a su vez el cambio de subrutinas personalizadas a componentes reutilizables. La reutilización de componentes propicia que el software pueda ser desarrollado más rápidamente, económicamente y con mayor confiabilidad.

El mantenimiento se va a reducir notablemente, ya que va a ser cada vez más específico y enfocado a los objetos de nueva creación. El tiempo promedio de corrección de errores se va a reducir notablemente ya que se sabe exactamente qué debe de hacer y qué no debe de hacer cada objeto.

Es posible debido a que están bien definidos los objetos y a su poca dependencia repartir el trabajo en varios grupos de personas.

La programación orientada a objetos tiene desventajas y sobre todo son en el sentido del aprendizaje ya que la curva de aprendizaje de estas metodologías es mayor, por lo tanto se requiere mayor capacitación. Además se requiere que se tenga algo de experiencia ya que muchas veces no porque la metodología tenga ventajas quiere decir que se sabrán aplicar, y por el contrario nos puede traer conflictos fuertes al aplicarlas sin un amplio conocimiento de las mismas.

3. METODOLOGÍA DE JACOBSON

3.1. ¿PORQUÉ ESTE MÉTODO?

Uno de los últimos enfoques dentro de la ingeniería de software ha sido la orientación a objetos, dada la aceptación que la orientación a objetos ha tenido, es natural que un gran número de profesionales se hayan dedicado a definir metodologías para el análisis y diseño de sistemas, sin embargo, como lo indica Mili[37] y colaboradores, no existe una metodología única que sea considerada como la mejor para el desarrollo de sistemas orientados a objetos.

Para llevar a cabo el desarrollo de este proyecto de tesis se eligió la metodología desarrollada por Ivar Jacobson[38] y colaboradores llamada Ingeniería de Software Orientada a Objetos (OOSE). Se eligió esta metodología debido a que incorpora varios conceptos para el desarrollo de sistemas orientados a objetos como los que se describen a continuación.

"Escenario" o "Caso de uso": Es una secuencia relacionada de transacciones que un usuario realizará en dialogo con el sistema, es decir, es una forma específica de usar el sistema.

Este es uno de los conceptos más fuertes de este método, porque nos puede dar un gran entendimiento sobre lo que se quiere del sistema y es fácil observar que definiendo todos los casos de uso se define toda la funcionalidad que debe de tener el sistema, de hecho todos los modelos de la metodología se generan a partir de los casos de uso del sistema.

A la gente que utiliza un determinado sistema a través de sus casos de uso se le denomina "Actor". Un actor representa un cierto papel que un usuario puede ejecutar.

De esta forma vemos que un actor debe de estar asociado a uno o varios casos de uso, entonces una vez definidos todos los casos de uso y los actores se tiene la base del sistema, como esto se define desde un principio se puede seguir la metodología de forma muy sencilla a través de sus diferentes modelos.

En el modelo de análisis, esta metodología utiliza tres esquemas para diferenciar tres tipos de objetos, haciendo los esquemas más entendibles y los objetos más específicos. La clasificación es muy lógica define tres tipos de objetos :

- de "interfaz": corresponden a los objetos que interactúan con el mundo exterior al sistema,
- de "entidad": éstos modelan la información que debe permanecer en el sistema durante un tiempo determinado,
- de "control" :éstos representan aquella funcionalidad que no está íntimamente ligada a ninguno de los otros dos tipos de objetos como control de secuencias.

[37] Mili H., Mili F. and Mili A.; 1995; *Reusing Software: Issues and Research Directions*; IEEE Transactions on Software Engineering ; June ; pp 528-561.

[38] Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998.

De esta forma utilizando estos tres tipos de objetos se puede obtener un sistema más definido en cuanto a funcionalidad y más adaptable debido a que las partes que más frecuentemente cambian son las de interfaz y de funcionalidad y estos cambios afectarían solamente a objetos individuales o grupos muy pequeños de objetos.

Ya en el modelo de diseño existe otro concepto que es el de "Diagramas de Interacción". Los diagramas de interacción son unos diagramas que describen la comunicación entre los bloques durante su ejecución. Esto nos da la ventaja de que se puede definir paso a paso y por cada escenario la comunicación entre los objetos y su comportamiento antes de su codificación, pero como serán codificados, entonces podemos adecuar las cosas de acuerdo a variables como el ambiente destino, etc.

Analizando los conceptos descritos anteriormente vemos que en conjunto estos conceptos y toda la metodología nos puede dar un sistema completo en el sentido de que se puede garantizar a través de los casos de uso y actores que todas las necesidades de los usuarios serán satisfechas, siempre y cuando se hayan identificado como casos de uso.

También podemos decir que se puede obtener un diseño bastante distribuido en cuanto a su manejo de información lo que lo hace fácil de mantener y/o extender.

Es una metodología que permitiría dentro de una organización definir y establecer actividades muy específicas y así tener un mayor control sobre las actividades y rendimiento de la gente, es decir, por ejemplo, podría tenerse un equipo de gentes que se dedicara al análisis y diseño, el cual quizá deba tener un cierto grado de preparación y escolaridad, además de otro grupo que se dedicara a la codificación únicamente, este grupo tomaría el análisis y diseño que realizara el grupo de análisis y diseño, debido a esto el grado de preparación y escolaridad que se necesitaría podría ser técnico.

3.2. DESARROLLO DE SISTEMAS Y METODOLOGÍA

Cuando se desarrollan sistemas, es importante conocer como interactúan los diferentes pasos del método y como se ajustan al proceso de desarrollo.

Una clave fundamental de un sistema de software es su estructura interna. Una buena estructura hace el sistema fácil de entender.

Comúnmente el desarrollo de un sistema que no es sencillo es una tarea compleja que no podemos realizar muy bien debido a que se debe de tener un manejo simultáneo de muchos requerimientos .

Lo que necesitamos hacer, es manejar la complejidad en una forma organizada. Esto se hará trabajando con diferentes modelos, cada uno enfocado en cierto aspecto del sistema: Introduciendo entonces la complejidad del sistema de forma gradual en cada modelo.

Cada modelo encierra una parte o aspecto del sistema a desarrollar. Estos modelos son generados por las actividades que se muestran en la siguiente figura.

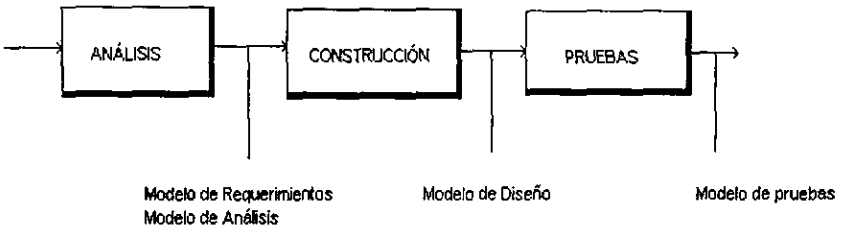


Figura 3.1. Diferentes modelos usados para manejar la complejidad

Uno de nuestros objetivos es encontrar un lenguaje de modelado, notación o técnica de modelado para cada uno de los modelos. Un conjunto de técnicas de modelado definen la metodología y ésta la arquitectura en la cual está basado el sistema. Otra forma de denotarlo es "*La arquitectura de un sistema es la denotación de un conjunto de técnicas de modelado*".[39]

Intuitivamente se puede pensar en la arquitectura como el conjunto de modelos que se pueden construir usando un método definido. La arquitectura entonces, define los tipos de modelos que pueden de ser construidos y las características que va a tener cada modelo.

Una técnica de modelado es normalmente descrita por medio de "sintaxis", "semántica" y "pragmática", por sintaxis entendemos al conjunto de objetos y la forma de utilizarlos para describir un modelo, por semántica que significa y pragmática son las reglas para utilizar la técnica de modelado.

Las técnicas de modelado son usadas para construir modelos, éstos deben de ser lo suficientemente poderosos para construir los sistemas en que estamos interesados. Las técnicas deben de ser fáciles de usar y contener pocos pero poderosos objetos de modelado para facilitar su fácil aprendizaje. Estos nos ayudarán a manejar la complejidad del sistema que vayamos a construir.

Entonces podemos decir que una arquitectura de un sistema es el resultado obtenido después de aplicar un método al sistema. Un método es un procedimiento planeado mediante el cual se aproxima a un objetivo paso a paso.

[39] Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998, pág. 118

3.3. METODOLOGÍA

El análisis orientado a objetos contiene, en algún orden las siguientes actividades.

- Localización de objetos
- Organización de objetos
- Descripción de interacción de objetos
- Definición de las operaciones de los objetos
- Definición interna de los objetos

LOCALIZACIÓN DE OBJETOS. Por medio del aprendizaje sobre lo relevante en el dominio de la aplicación, los objetos pueden ser encontrados. Es muchas veces el caso en que no hay problema para encontrar objetos; la dificultad es usualmente en escoger o seleccionar los objetos relevantes para el sistema. El objetivo es encontrar los objetos esenciales, ya que en la medida que sean esenciales probablemente siempre existirán y de esta forma obtendremos un sistema estable.

La mayoría de los métodos orientados a objetos hoy en dia, tienen un sólo tipo de objeto. De esta forma uno obtiene un modelo simple y general. Sin embargo hay razones para tener diferentes tipos de objetos, con un sólo tipo de objeto puede ser difícil ver la diferencia entre diferentes objetos. Teniendo varios tipos de objetos, uno puede obtener más rápido un panorama del sistema.

ORGANIZACIÓN DE OBJETOS. Hay varios criterios utilizados para la organización y clasificación de objetos y clases de objetos. Una clasificación empieza por considerar cuántas clases de objetos son similares entre sí. Esto es naturalmente la base de la jerarquía hereditaria. Una clase puede heredar a otra clase.

Otra clasificación puede ser hecha considerando qué objetos trabajan con que otros objetos o qué tanto un objeto es parte de otro. Una clasificación similar es ver que objetos son en alguna manera dependientes de otros y se pueden agrupar en subsistemas.

DESCRIPCIÓN DE INTERACCIÓN DE OBJETOS. De forma de obtener un esquema de cómo el objeto se adecua en el sistema, podemos describir diferentes escenarios en los cuales el objeto toma parte y se comunica con otros objetos. De esta forma, podemos describir completamente los alrededores del objeto y qué esperan los otros objetos de nuestro objeto. La interfaz del objeto puede ser decidida de estos escenarios. También consideraremos como ciertos objetos son parte de otros objetos.

DEFINICIÓN DE LAS OPERACIONES DE LOS OBJETOS. Las operaciones sobre los objetos vienen naturalmente cuando consideramos la interfaz de los objetos. Las operaciones pueden ser identificadas directamente de la aplicación, cuando consideramos qué podemos hacer con los elementos que modelamos.

DEFINICIÓN INTERNA DE LOS OBJETOS. Finalmente, el objeto va a ser definido internamente, esto incluye la definición de la información que cada objeto va a almacenar.

Durante el desarrollo creamos modelos del sistema que estamos creando. Para diseñar esos modelos trabajamos en un proceso de descripción de los procesos del sistema. Cada proceso toma uno o varios modelos y los transforma en otros modelos.

Figura 3.2. Esquema de un proceso

El modelo final deberá de ser una descripción del sistema completa y probada.

El proceso de análisis produce dos modelos; de la especificación de requerimientos se obtiene el "*modelo de requerimientos*", aquí se especifica toda la funcionalidad del sistema. Esto es hecho principalmente utilizando escenarios en el "*modelo de escenarios*", que es parte del modelo de requerimientos. El modelo de escenarios va a ser la base de los procesos de construcción y pruebas y controla una gran parte del desarrollo del sistema.

El modelo de requerimientos es la base de otro modelo creado para el proceso de análisis llamado "*modelo de análisis*". Este modelo es la base de la estructura del sistema, en ese momento especificamos todos los objetos lógicos que son incluidos en el sistema y como se relacionan y son agrupados.

Estos dos modelos son el resultado de el proceso de análisis y proporcionan la entrada para el proceso de construcción.

En el proceso de construcción se diseña e implementa el sistema (figura 3.3). Primeramente se realiza un diseño, esto resulta en un "*modelo de diseño*" en el que cada objeto va a ser completamente especificado. El subproceso de implantación implementa los objetos y resulta en el "*modelo de implantación*" que consiste en el código fuente.

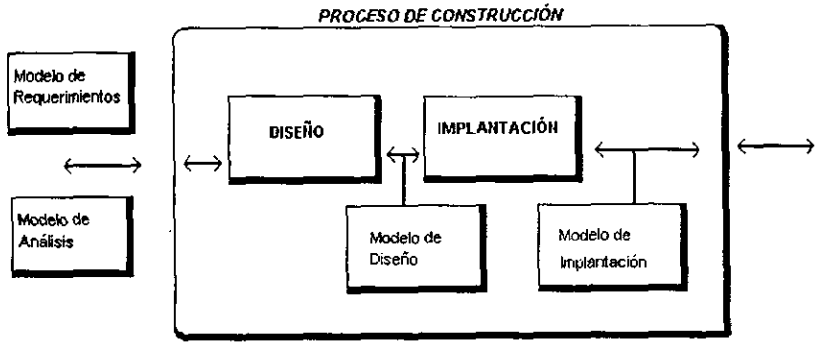


Figura 3.3.Proceso de construcción

Este modelo de implantación proporciona junto con los modelos de requerimientos y de análisis la entrada para el proceso de pruebas (figura 3.4). El proceso de pruebas prueba el modelo de implantación, parcialmente a partir del modelo de requerimientos y del modelo de diseño y produce un "*modelo de prueba* ". Este modelo es realmente el resultado de probar el modelo de implantación.

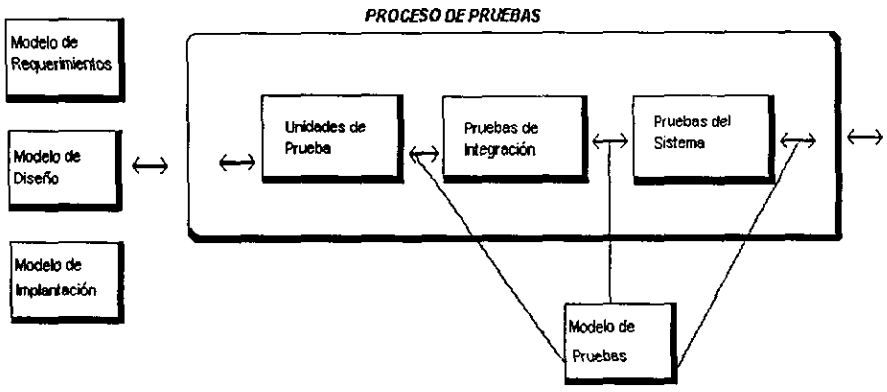


Figura 3.4. Proceso de Pruebas

Como se ha descrito antes una característica importante de estos modelos es la rastreabilidad. Es decir, se puede rastrear un objeto de un modelo a otro hacia delante y hacia atrás.

El trabajo para crear un modelo, es identificar y descubrir objetos en un cierto espacio de información y construir modelos utilizando estos objetos.

Hay varios criterios en varios métodos orientación a objetos. Usualmente, decimos que el objeto debe de tener una interpretación de la realidad. También decimos que los objetos deben de ser obvios, cosas tangibles, cosas que se puedan enfocar o señalar.

Cuando evaluamos cómo necesita el sistema manejar esos objetos podemos decidir cuando un objeto deberá ser incluido o dejado fuera del modelo que se creará. Un objeto puede ser perfectamente adecuado para un modelo y erróneo completamente para otro. Esto significa que un objeto debe ser colocado en un contexto para ver si es apropiado. Por lo tanto lo que es realmente de interés es cómo un objeto trabaja con otros objetos y bajo que condiciones.

¿Qué es entonces un buen objeto? El criterio más importante es que debe de ser robusto a las modificaciones y que nos ayude a entender el sistema. Cómo se sabe todos los sistemas que se crean serán modificados, debemos de crear un modelo de estructura robusto. Por lo tanto debemos analizar como afectarán las modificaciones al sistema.

Trabajando un largo tiempo con los primeros modelos, obtendremos un buen entendimiento del sistema. El proceso de desarrollo debe de ser diseñado para que resulte en una estructura robusta lo más pronto posible. Esto reduce el riesgo de tener que cambiar la estructura del sistema en etapas posteriores y esto nos fuerza a utilizar una estructura entendible.

3.3.1. ANÁLISIS

Cuando son identificados los requerimientos de un sistema empieza el desarrollo. La especificación de requerimientos se realizará de alguna manera describiendo, qué se quiere obtener del sistema. A partir de esta especificación desarrollamos el sistema, tratando de que cumpla con la funcionalidad y calidad requerida así como que sea lo más efectivo posible para poder ponerlo en operación.

A partir de aquí empieza el periodo más importante del sistema, ya que se hace una gran cantidad de mantenimiento y puede tomar mucho tiempo dependiendo de la complejidad del sistema.

Los modelos que son desarrollados durante el análisis son completamente orientados a la aplicación y ninguna consideración es hecha sobre el ambiente de implantación donde será realizado. El propósito es formular el problema y resolver problemas bajo condiciones ideales.

En el análisis, se desarrollan dos diferentes modelos: el modelo de requerimientos y el modelo de análisis. La base real es la especificación de requerimientos y la discusión con los usuarios prospectos. El primer modelo, el modelo de requerimientos debe de abarcar lo más posible para delimitar el sistema y definir su funcionalidad. Para este propósito desarrollamos un esquema conceptual del sistema usando objetos del dominio del problema y especificando también las descripciones de las interfaces. También describimos el sistema como un número de escenarios que son desarrollados por un número de actores.

Los actores constituyen el ambiente del sistema, y los escenarios es lo que pasa en el sistema.

El modelo de análisis da una configuración conceptual del sistema, consiste en objetos de entidad, de control y de interfaz. El propósito de este modelo es desarrollar una estructura robusta y extendible como base de una construcción. Cada uno de los tipos de objetos tiene su propia contribución especial para obtener esa robustez, y juntos ofrecer la total funcionalidad que ha sido especificada en el modelo de requerimientos.

3.3.1.1. MODELO DE REQUERIMIENTOS

Es necesario hacer una primera transformación, esta es, de la especificación de requerimientos al modelo de requerimientos. El modelo de requerimientos consiste en :

- Un modelo de escenarios
- Descripción de interfaz
- Un "modelo de dominio" del problema

Modelo de Escenarios

El modelo de escenarios utiliza actores y escenarios que esquemáticamente se especifican en la siguiente figura. Estos conceptos son simples y nos ayudan a definir qué existe fuera del sistema (actores) y que debe de ser desarrollado por el sistema (escenarios). Los actores entonces representan lo que interactúa con el sistema. Ellos representan todo lo que se necesita para intercambiar información con el sistema. Los actores no son como los otros objetos en el sentido de que sus acciones no son determinísticas. Se debe diferenciar entre actores y usuarios. Un usuario es la persona que usa el sistema, mientras que un actor representa un cierto rol que un usuario puede ejecutar. Podemos ver entonces al actor como una clase y un usuario como una instancia de esa clase. Esta instancia existe sólo cuando el usuario hace algo con el sistema. La misma persona puede aparecer como instancia de diferentes actores.

ACTOR

ESCENARIO

Figura 3.5. Representación de actores y escenarios

Como forma de identificar los escenarios que serán desarrollados en el sistema, vamos a identificar a los usuarios del sistema. Estro se logra por medio de los actores. Los actores modelan a los usuarios prospectos; el actor es un tipo de usuario o categoría, y cuando un usuario hace algo actúa como una ocurrencia de ese tipo. Una persona puede instanciar (tomar los papeles de) varios actores. Los actores entonces definen los roles que los usuarios pueden jugar.

Los actores modelan todo lo que se necesita para intercambiar información con el sistema. Los actores pueden modelar usuarios humanos pero también pueden modelar otros sistemas que se comunican con nuestro sistema, lo esencial es que los actores constituyen cualquier cosa que es externa al sistema que se está desarrollando.

Por esto se tiene que hacer una "delimitación del sistema", para definir dónde está el límite entre los actores y los escenarios.

Con respecto a los actores podemos pensar en ellos como en una clase que describe un comportamiento, y un usuario por otro lado puede ejecutar varios roles, es decir, servir como varios actores. A los actores que van a utilizar el sistema directamente los llamamos "actores primarios", Además de los actores primarios hay actores que supervisan y mantienen el sistema, a estos se les llamará "actores secundarios". Los actores secundarios existen sólo que los primarios usen el sistema.

Una forma de empezar a identificar a los actores es analizar hacia a quien se supone que va a ayudar el sistema.

Se hace la división entre actores primarios y actores secundarios debido a que queremos que la estructura del sistema sea decidida en términos de la funcionalidad principal. Los actores primarios gobiernan la estructura del sistema. Así cuando identificamos escenarios, empezamos con los actores primarios. De esta manera podemos garantizar que la arquitectura del sistema va a estar adaptada a los usuarios más importantes.

Después de haber decidido qué está fuera del sistema, podemos definir la funcionalidad dentro de él, esto se hace especificando los escenarios. Un escenario es una forma específica de utilizar el sistema desarrollando una parte de la funcionalidad. Cada escenario construye un ciclo completo de eventos iniciados por un actor y especifica (el escenario) la interacción que tiene lugar entre el actor y el sistema. La colección de escenarios especifican todas las formas existentes de utilizar el sistema.

La razón por la que hayamos identificado actores inicialmente es que ellos van a ser la mayor herramienta para encontrar escenarios, Cada actor va a desarrollar un cierto número de escenarios en el sistema, para cada ciclo completo de eventos iniciados por un actor, identificamos un escenario. Analizando cada actor y definiendo todo lo que ellos son capaces de hacer con el sistema, definiremos la funcionalidad completa del sistema.

Varios escenarios pueden empezar de forma similar, no siempre va a ser posible decidir qué escenario ha sido instanciado hasta que sea completado. En otras palabras el actor no demanda que se desarrolle un escenario, el sólo empieza un ciclo de eventos que finalmente resulta en un escenario completo.

Considerando cada actor, se decide qué escenarios se supone puede desarrollar dicho actor. Para identificar los escenarios, podemos leer la especificación de requerimientos desde la perspectiva del actor y discutirlos con los que serán los actores. Nos puede ayudar el hacer ciertas preguntas como :

¿Qué es la principal tarea de cada actor?
¿Tiene el actor que leer/escribir/cambiar alguna información del sistema?

¿Debe de informar el actor al sistema acerca de los cambios externos?
¿Desea el actor ser informado acerca de los cambios inesperados?

Cuando el esquema se estabiliza, cada escenario debe describirse detalladamente. Primero se describe el curso básico, que es el curso de eventos más importante, dando el mejor entendimiento del escenario. Variantes del curso básico de eventos, y errores que puedan ocurrir son descritos en cursos alternativos. Normalmente un escenario tiene un solo curso básico, pero varios cursos alternativos.

Debido a que los escenarios muchas veces se enfocan en una funcionalidad particular del sistema es posible analizar toda la funcionalidad del sistema de una manera incremental. De esta manera podemos desarrollar escenarios para diferentes áreas de funcionalidad independientemente y después juntarlas para completar el modelo de requerimientos.

La extensión especifica cómo una descripción de un escenario puede ser insertada o extender otra definición de otro escenario. Las extensiones a los escenarios pueden ser descritas en una forma muy sencilla y en particular los cambios y adiciones a la funcionalidad son hechos fácilmente.

Así se describen los principales escenarios totalmente independiente de cualquier funcionalidad extendida. Similarmente podemos adicionar nuevas extensiones sin cambiar las descripciones originales.

La asociación de una extensión es dibujada con una flecha punteada. La asociación de instancias como se mencionó es dibujada con flechas completas.

Para entender mejor la extensión podemos ver una situación típica donde tenemos un simple escenario login/logout, y describe todos los escenarios que se puedan realizar cuando un usuario se conecta, esto por medio de la extensión del escenario Login/Logout (ver figura 3.6).

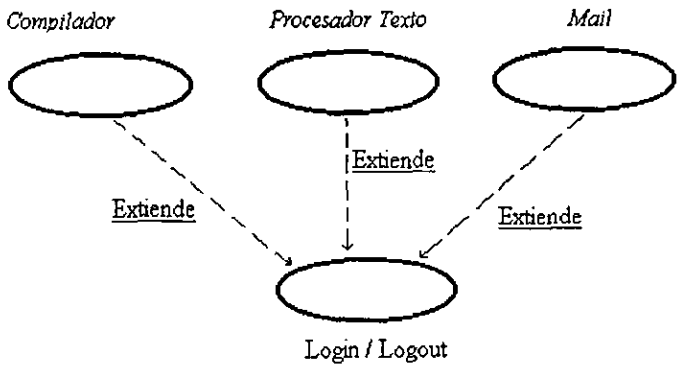


Figura 3.6. Extensión

Será simple agregar nuevas opciones sin tener que hacer cambios en los escenarios anteriores; los nuevos escenarios son simplemente definidos y usados como extensiones de Login/Logout.

La extensión es usada para modelar extensiones de otros escenarios completos. Aquí hay algunos ejemplos de cuando usar extensiones :

- Para modelar partes opcionales de escenarios
- Para modelar cursos complejos y alternativos que ocurren rara vez.
- Para modelar subcursos separados que sean ejecutados sólo en ciertos casos.
- Para modelar la situación donde varios escenarios diferentes pueden ser insertados en un escenario especial, como Login/logout o un sistema de opciones de menú.

Debemos ver la asociación de extensión como una interrupción en el escenario original el cual corre cuando el nuevo escenario es insertado. El escenario original no sabe cuándo ocurrirá o no una interrupción.

Para cada escenario que debe ser insertado en otro escenario, se almacena la posición en el escenario original donde es insertado el escenario de extensión. Esta posición es almacenada y es descrita en el escenario de extensión (no en el original). La posición es expresada como referencia de un lugar en la descripción del escenario original.

Cuando un aviso es insertado de esta manera sucede que el escenario original corre usualmente hasta el punto donde el escenario nuevo es insertado. En este punto, el nuevo curso es insertado. Después de que la extensión ha terminado, el curso original continua como si nada hubiera pasado.

El refinamiento es hecho principalmente para identificar partes comunes de los escenarios. De esta forma sólo tenemos que describir las partes comunes una vez en todos los escenarios que muestren el comportamiento. Cualquier cambio a esta parte automáticamente afectará a todos los escenarios que la compartan. A los escenarios que extraemos los llamamos "escenarios abstractos" y no pueden ser instanciados, pero son manejados para describir partes que son comunes a los otros escenarios. A los escenarios que realmente se instancian se llaman "escenarios concretos". Esto es similar al nombramiento de las variables abstractas y concretas.

Las descripciones de los escenarios abstractos son utilizadas en las descripciones de los escenarios concretos. Esto significa que cuando una instancia de los escenarios sigue la descripción de un escenario concreto, en cierto punto continuará siguiendo la definición de un escenario abstracto. Esta relación es como un tipo de herencia, sin embargo, no tiene la misma semántica que tiene la herencia en lenguajes orientados a objetos. Así que le daremos otro nombre "relación de escenarios".

Como en una asociación de clases es dibujada con flechas punteadas de la clase concreta a la abstracta, para los escenarios es usado también.

Normalmente comportamientos similares entre escenarios son identificados después de que se han descrito los escenarios, sin embargo algunas veces es posible identificarlos antes.

La reutilización entre escenarios no está limitada a un escenario abstracto. Varias partes que son comunes a otros escenarios pueden ser extraídas para un escenario. Un escenario específico puede entonces utilizar todos estos escenarios abstractos.

En la descomposición de escenarios, siempre es usado el curso completo. El curso no necesita ser una secuencia atómica, aunque muchas veces este es el caso. Sin embargo podemos tener una situación donde los escenarios abstractos pueden usarse entrelazándolos en un escenario concreto (ver figura 3.7). Del lado izquierdo hay un escenario abstracto con una secuencia A y B estas serán integradas con las del escenario abstracto C y D como se ilustra en la figura.

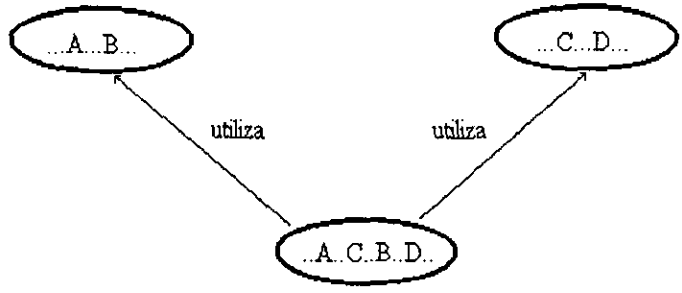


Figura 3.7. Descomposición de escenarios

Los escenarios abstractos pueden ser utilizados también por otros escenarios abstractos.

Una técnica para extraer escenarios es identificar actores abstractos. Un actor abstracto típicamente describe un rol que debe ser jugado en el sistema. Cuando diferentes actores juegan roles similares pueden heredar un actor abstracto común. La ventaja del modelado del actor abstracto es que expresa similaridades en los escenarios. Si la misma parte del escenario puede ser desarrollada por diferentes actores los escenarios necesitan ser especificados sólo con respecto a un actor y no a todos.

La asociación de escenarios es entonces usada cuando dos o más escenarios tienen un comportamiento común. Normalmente, no hay razón para crear escenarios abstractos que son utilizados sólo por un escenario.

Como la asociación y la extensión de escenarios abstractos son asociaciones de clases, es lógico preguntar cuándo se debe de utilizar una u otra. Un criterio importante es que tan fuertemente está acoplada la funcionalidad en el escenario. Si el curso a ser extendido es significante por sí mismo, la adición puede ser descrita usando extensión. La extensión entonces sólo necesita acceder al flujo del escenario que está siendo extendido. Si el curso está fuertemente acoplado en su funcionalidad y la inserción debe tener lugar para obtener el curso completo, deben ser elegidos los escenarios abstractos. Hay una diferencia en como son encontrados; los escenarios abstractos son encontrados a través de la extracción de secuencias comunes de diferentes escenarios mientras las extensiones son encontradas cuando son introducidos nuevos cursos, o cuando hay, extensiones de un escenario existente que el usuario desea desarrollar en algunos casos específicos.

Diseñando de esta manera el modelo del sistema va a ser un manejo de escenarios. Cuando se quiera cambiar el comportamiento del sistema , se remodelará el actor y escenario apropiado. Toda la arquitectura del sistema va a estar controlada por ¿Qué quieren hacer los usuarios con el sistema?. Como se tiene rastreabilidad a través de los modelos, será posible modificar el sistema con los nuevos requerimientos. Se pregunta a los usuarios que es lo que quieren cambiar (qué escenario) y vemos dónde se deben de hacer los cambios en otros modelos.

Otra importante característica para el modelo de requerimientos es que se puede discutir con los usuarios y encontrar sus requerimientos y preferencias. Este modelo es fácil de entender y formular desde la perspectiva del usuario, así se puede fácilmente hablar con los usuarios y ver si estamos construyendo el sistema de acuerdo a sus requerimientos.

Descripción de Interfaz

Cuando se describen los escenarios y se comunican con los usuarios potenciales, muchas veces es apropiado describir las interfaces en más detalle. Si es una interfaz hombre – máquina se pueden usar esquemas para mostrar qué va a ver el usuario en la pantalla. Si las interfaces son protocolos de hardware, podemos referirnos a varios estándares. Estas descripciones de las interfaces son una parte esencial de las descripciones de escenarios y deben de acompañarlas.

Cuando se diseñan interfaces de usuario, es esencial tener a los usuarios involucrados. Es esencial que la interfaz refleje la vista lógica que tiene el usuario del sistema.

Dominio del Problema

Cuando trabajamos en el modelo de requerimientos, puede ser difícil algunas veces definir la tarea del sistema y especialmente los límites del sistema. Este es el caso típico donde la especificación de requerimientos existe sólo en una forma vaga. Entonces, una muy buena herramienta es empezar a desarrollar una vista lógica del sistema usando los objetos del dominio del problema, esto es, objetos que tienen una contraparte directa en el ambiente de la aplicación y que el sistema lo sabe.

Así el modelo de dominio de problema nos va a ayudar a realizar una lista sustantiva que será un gran soporte cuando se especifiquen los escenarios. De este modelo es posible definir los conceptos con los que el sistema va a trabajar. De esta forma tendremos un glosario que puede ser usado para formular la funcionalidad de los escenarios.

El mayor beneficio del modelo, es que es una muy buena herramienta para quien se comunica con el sistema. Como los usuarios y ordenantes reconocen todos los conceptos del sistema, el modelo puede ser utilizado cuando se define qué va a hacer el sistema. Para este modelo es recomendable que los usuarios potenciales participen activamente, de esta manera se

podrá obtener una terminología común en el desarrollo del modelo y disminuir la probabilidad de desentendimiento entre el desarrollador y el usuario potencial.

La experiencia muestra que muchos (si no todos) los objetos del dominio se van a mostrar como objetos de entidad en el modelo de análisis. Sin embargo es peligroso hacer esto mapeando mecánicamente ya que pueden existir cambios.

El modelo del dominio del problema puede ser usado para diferentes propósitos, por ejemplo, para tener un mejor entendimiento del problema. Este modelo también puede ser usado para capturar todos los conceptos fundamentales en un modelo.

El modelo de requerimientos como se ha dicho, es suficiente para especificar la funcionalidad del sistema, sin embargo, podemos elaborar este modelo no sólo para reforzar la reutilizabilidad, sino también para preparar la transición al modelo de análisis. Este trabajo no es muy interesante para la gente que pide el sistema ya que es un área que no le concierne.

3.3.1.2. MODELO DE ANÁLISIS

Una vez que se tiene el modelo de requerimientos donde se definen limitaciones y tendencias de el sistema, se puede empezar a desarrollar el sistema. Esto empieza con el desarrollo de el modelo de análisis. Este modelo plantea la estructura del sistema independientemente del ambiente de implantación actual. Esto significa que se enfoca en la estructura lógica del sistema. Es aquí donde se define la robustez, estabilidad y la estructura mantenible y extendible.

El modelo de análisis representa una estructura del sistema más estable y mantenible que será robusta para el ciclo de vida completo del sistema.

En el espacio de información para este modelo, se tiende a capturar información, comportamiento y presentación (ver figura 3.8). La dimensión de información especifica la información contenida en el sistema a corto y largo plazo. A lo largo de esta dimensión se describe el estado interno del sistema. En la dimensión de comportamiento, se especifica el comportamiento que el sistema adoptará. Aquí se especifica cuando y como el sistema cambia de estado. La dimensión de presentación proporciona detalles para presentar el sistema al mundo externo.

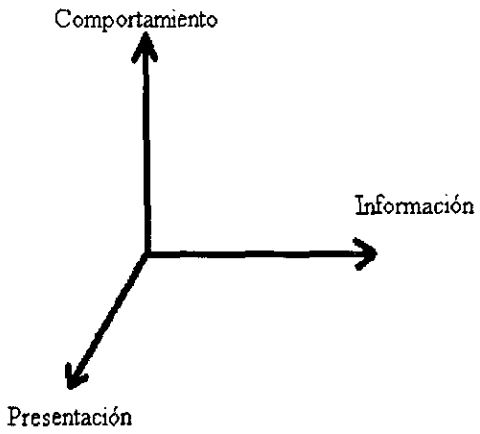


Figura 3.8. Espacio de información

El modelo de análisis es construido especificando objetos en este espacio de información. Una posibilidad es tener objetos que sólo expresen una dimensión. Este es el caso con los métodos estructurados, donde las funciones son colocadas a lo largo del eje de comportamiento y los datos son colocados en el eje de la información. Si diseñamos de esta forma obtendremos un sistema sensible a las modificaciones, ya que nosotros muchas veces debemos modificar el comportamiento cuando se modifica la estructura de información.

En este método se utilizan tres tipos de objetos, la razón es simplemente ayudarnos a obtener una estructura que va a ser más adaptable a los cambios. Los objetos utilizados en el modelo de análisis son "*objetos de entidad*", "*objetos de interfaz*" y "*objetos de control*" (ver figura 3.9).

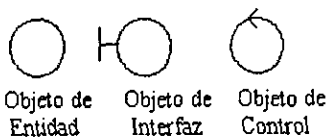


Figura 3.9. Tipos de objetos usados

Cada uno de estos tres tipos de objetos captura al menos dos de las tres dimensiones discutidas anteriormente. Sin embargo cada uno tiene una cierta inclinación hacia alguna de las dimensiones como se muestra en la figura siguiente.

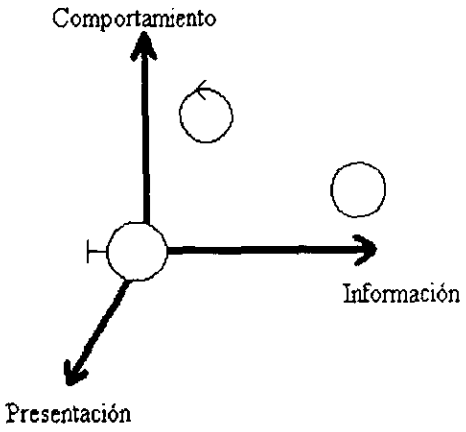


Figura 3.10. Los objetos en el espacio de información

De esta forma tenemos diferentes propósitos para los tres tipos de objeto. ¿Por qué esos tres objetos nos dan estabilidad al sistema?. Lo que se asume básicamente es que todos los sistemas cambian, por lo tanto la estabilidad ocurre en el sentido de que todos los cambios sean locales, esto es, afecten preferentemente a un solo objeto en el sistema. Podemos decir que los cambios más comunes a un sistema son en su funcionalidad y su interfaz, los cambios de la interfaz son realizados comúnmente en los objetos de interfaz, y los cambios de funcionalidad se realizan en el objeto que sea necesario ya que la funcionalidad que se quiere cambiar puede estar en cualquiera de los tipos de objeto.

Pueden encontrarse grandes similaridades entre los objetos de entidad de el modelo de análisis y los objetos dados en otros métodos. Sin embargo el comportamiento que colocamos en los objetos de control, es distribuido en varios objetos, en otros métodos haciendo complicado cambiar ese comportamiento.

La razón principal entonces por lo que modelamos con tres objetos es tener la localización en los cambios del sistema.

El modelo de análisis es formado a partir del modelo de escenarios. Cada escenario va a ser dividido en objetos de los tres tipos descritos anteriormente. En el modelo de requerimientos especificamos la funcionalidad completa del sistema. Esta funcionalidad debe ser estructurada para obtener una estructura robusta y extendible. De esta forma los escenarios son divididos en objetos de análisis (ver figura 3.11). En la práctica esto significa que la funcionalidad específica en los escenarios debe ser almacenada en objetos diferentes. De esta manera cada escenario va a estar dado por varios objetos en el modelo de análisis. Un objeto por supuesto puede ser común a diferentes escenarios.

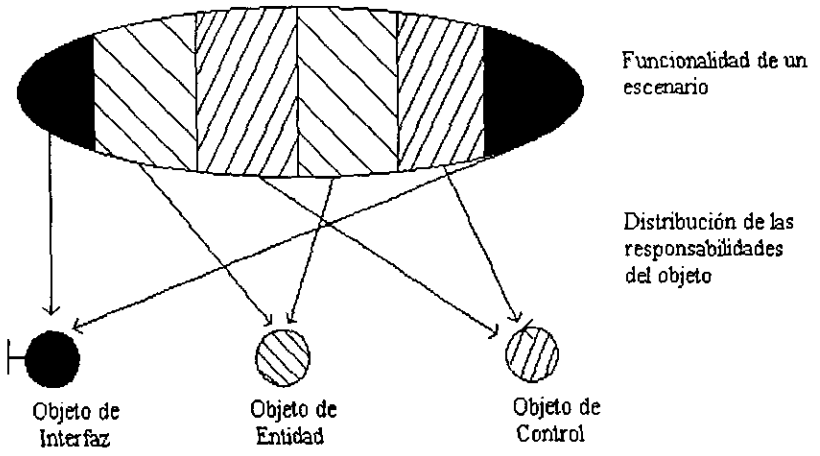


Figura 3.11. División de escenarios en objetos de análisis

Básicamente un escenario se divide de acuerdo a los siguientes principios :

- Aquellas funcionalidades de los escenarios que son directamente dependientes del ambiente del sistema, son colocadas en los objetos de interfaz.
- Aquellas funcionalidades relacionadas con el almacenamiento y manejo de información que no son naturalmente colocadas en cualquier objeto de interfaz son colocadas en un objeto de entidad.
- Funcionalidades específicas de uno o algunos escenarios y que no son naturalmente colocadas en cualquiera de los otros objetos son colocadas en objetos de control.

Llevando a cabo esta división, obtenemos una estructura que nos ayuda a entender el sistema de una vista lógica y también nos da una alta localización de los cambios y es menos sensible a las modificaciones.

Cuando hacemos esta división de funcionalidad es cuando decidimos sobre la robustez y mantenibilidad del sistema. El principio básico es obtener localización de los cambios.

Como es muy difícil delimitar claramente entre las funcionalidades en un sistema, en la práctica los desarrolladores son forzados a hacer muchos juicios acerca de donde dividir la funcionalidad entre los objetos. Es recomendable que se razone en términos de los cambios potenciales del sistema. De esta forma se desarrolla una estructura que es estable para la mayoría de los cambios, esto es, "especulación de los cambios potenciales", esto se obtiene con la experiencia.

Este modelo de análisis va a formar las bases de una arquitectura de un sistema y se observa que no se ha hecho ninguna consideración de los factores de implantación actual. Esto provee una arquitectura del sistema, ideal, sin tomar en cuenta los factores de implantación.

El modelo de análisis nos guía para crear una buena plataforma para el diseño del sistema y forma las bases del diseño. El modelo de requerimientos es estructurado entonces para el modelo de análisis.

El trabajo en el desarrollo del modelo de análisis realmente ocasiona la distribución del comportamiento especificado en las descripciones de escenarios a través de objetos en el modelo de análisis (ver figura 3.12). Un objeto puede ser común a varios escenarios diferentes. Así debemos establecer explícitamente cual objeto es responsable de que comportamiento en el escenario. Un procedimiento más natural es escribir una descripción verbal de las responsabilidades o roles de cada objeto.

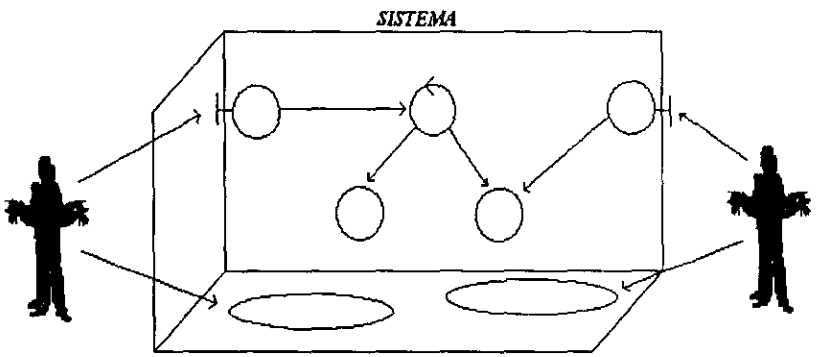


Figura 3.12. Objetos comunes a distintos escenarios

Objetos de Interfaz

Toda la funcionalidad especificada en las descripción de escenarios que es directamente dependiente del ambiente del sistema es colocada en los objetos de interfaz. Es a través de estos objetos que los actores se comunican con el sistema. La tarea de un objeto de interfaz es traducir la entrada de un actor en eventos del sistema, y traducir esos eventos en el sistema en algo que el actor está interesado y que se le presenta. Los objetos de interfaz, en otras palabras, describen la comunicación bidireccional entre el sistema y sus usuarios.

Los objetos de interfaz son muy fáciles de identificar. Se tienen tres estrategias :

- Pueden claramente identificarse de las descripciones de interfaz del sistema acompañadas del modelo de requerimientos.
- Se puede empezar por los actores.
- Se pueden leer las descripciones de los escenarios y extraer la funcionalidad específica de la interfaz.

Cada actor en concreto necesita su propia interfaz para su comunicación con el sistema. En muchos casos un actor puede necesitar varios objetos de interfaz.

Una vez identificados los lugares en la descripción de los escenarios donde está involucrada la funcionalidad de la interfaz. La descripción del escenario es tomada directamente de los requerimientos del modelo.

Es evidente que los objetos de interfaz no son enteramente independientes entre sí pero ellos deben de saber que lo otros son capaces de resolver ciertas tareas.

Esto se resuelve con la introducción de "asociaciones de conocimiento", entre los objetos. Una asociación de conocimiento es una asociación estática entre instancias y significa que una instancia sabe de la existencia de otra instancia. Esto no da al objeto el derecho de cambiar información con el otro objeto; para ese propósito se necesita una asociación dinámica. Las asociaciones de instancia son dibujadas con líneas sólidas dirigidas.

Un objeto puede asociar varias instancias de la misma clase. Por lo tanto debemos describir cuantas instancias pueden ser asociadas con la asociación de conocimiento. Esto se realiza asignando "cardinalidad" a cada asociación. Esta cardinalidad indica cuantas instancias pueden ser asociadas. También le damos a las asociaciones de conocimiento un nombre que aclare de que qué trata la relación.

Una asociación de conocimiento es entonces una asociación estática de instancia que es dibujada con una línea dirigida, sólida y tiene un nombre y cardinalidad, sólo es unidireccional.

Un tipo especial de asociación de conocimiento es la asociación "consiste en", la que es usada para establecer que un objeto está compuesto por otros objetos. Como una estructura, donde un objeto tiene asociaciones con las partes participantes, es llamado algunas veces "agregado".

Esto es común en los objetos de interfaz (ver figura 3.15). En una ventana del sistema, podemos establecer que la ventana puede consistir en botones, menús y barras de desplazamiento Cada una de estas unidades puede ser modelada por un objeto de interfaz individual. El resultado sería una estructura de interfaz formando un árbol. Esto muchas veces es llamado "*jerarquía contenedora*" o "*partición jerárquica*".

Habiendo identificado los objetos de interfaz, va a ser fácil modificar una interfaz en el sistema, ya que sólo serán convenientes los cambios a un objeto de interfaz. Teniendo todo lo concerniente a una interfaz en un objeto todo cambio será local. Como los cambios de interfaz son muy comunes es vital que esto sea manejable.

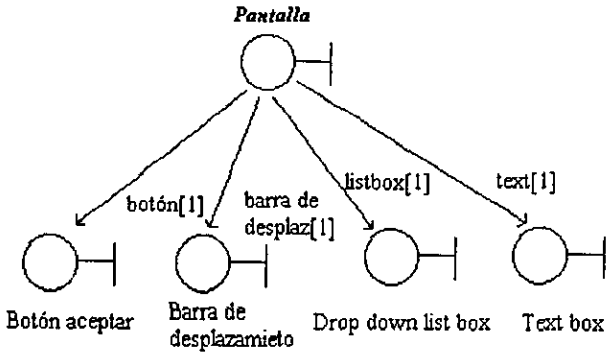


Figura 3.15. Ejemplo de asociación "*consiste en*" en un objeto interfaz

Es obvio que hay dos tipos de modelo de interfaz: interfaces para otros sistemas e interfaces para humanos.

Para los objetos de interfaz que se comunican con otros sistemas, la comunicación es usualmente descrita en términos de protocolos de comunicación. Estos objetos de interfaz pueden ser de un tipo que traduzca a un protocolo estandarizado, o tal vez sólo mande un estímulo que es producido internamente sin ninguna conversión compleja. Esto constituye otra ventaja: si el protocolo es cambiado estos cambios van a ser locales a este objeto de interfaz. Actualmente en un proyecto en el que sabemos que el protocolo va a ir cambiando modularizamos el objeto, de esta forma los cambios van a ser muy fáciles de incorporar.

La otra clase de objeto de interfaz se comunica con los usuarios humanos. Ahora muchas veces se solicita una GUI (Grafic User Interface) entre el sistema y el usuario. Los objetos de interfaz pueden ser difíciles de modelar; pero pueden ser modelados con estructuras completas de objetos de interfaz como se mencionó anteriormente. Existen varias técnicas diferentes para el buen diseño de la interfaz. Es fundamental que el usuario tenga un esquema lógico y coherente del sistema.

Tomando a los usuarios prospectos como una parte activa, garantizamos que las interfaces satisfacerán las necesidades de los usuarios. En aplicaciones de interfaz intensiva no es anormal que la mayor parte de la aplicación sea de interfaz.

Llega a ser evidente que los objetos de interfaz son especiales para presentación, pero ellos también pueden manejar información y tener comportamiento. Cuanta información y comportamiento puede ser añadido al objeto de interfaz debe de ser decidido caso por caso. En un extremo, el objeto de interfaz sólo pasa el estímulo que recibe de un actor a otros objetos en el

sistema, sin participar activamente en el curso de los eventos. En el otro extremo, el comportamiento del objeto interfaz es muy complejo; información compleja es añadida al objeto de interfaz y puede funcionar independientemente de otros objetos.

¿Cómo es posible decidir qué comportamiento en el escenario debe añadirse a un objeto de interfaz en particular?, generalmente el potencial de cambio debe de decidir esto.

Así para identificar qué parte del flujo en un escenario debe ser colocado en objetos de interfaz, nos enfocamos en las interacciones entre los actores y los escenarios. Esto significa que nos debemos de fijar por unidades de despliegue (pantallas) una o más de las siguientes características:

- Presentan información al actor o solicitan información al actor.
- Su funcionalidad es cambiada si cambia el comportamiento del actor.
- El curso es dependiente de un tipo particular de interfaz.

Se pueden diferenciar entre varias estrategias diferentes para almacenar la funcionalidad.[40]

1) *CONTROL DOMINANTE DE COMPUTACIÓN O CONTROL EMBEBIDO.* Es donde colocamos el control de la funcionalidad dentro del sistema, esto es, en los objetos de control y los objetos de entidad. Aquí los objetos de interfaz no tienen mucha funcionalidad. Esta estructuración puede ser eficiente en ejecución pero difícil de hacer un prototipo de ella, ya que no es introducida mucha funcionalidad en el objeto interfaz. Se puede poner toda la secuencialidad localmente en un objeto de control para su fácil modificación.

2) *CONTROL DOMINANTE DE DIÁLOGO.* Es donde colocamos mucho control de funcionalidad en los objetos de interfaz y estos objetos modelan mucha de la funcionalidad del sistema. En este caso no se tienen muchos objetos de control en el modelo. Esta estrategia es fácil de hacer en prototipo, pero incremente la complejidad de las interfaces ya que son mezclados diferentes niveles de abstracción.

3) *CONTROL MEZCLADO.* Coloca el control en ambos lados permitiendo invocación de diálogo del lado computacional y viceversa. Esto ofrece más flexibilidad, pero requiere que los programadores sean más disciplinados para mantener la independencia de diálogo.

4) *CONTROL BALANCEADO.* Es donde separamos el control de ambos el diálogo y la computación. El componente global, que típicamente es un objeto de control, gobierna la secuencia a través de invocaciones de código y funciones computacionales.

[40] Hartson H.R. y Hix D, HUMAN-COMPUTER INTERFACE DEVELOPMENT: CONCEPTS AND SYSTEMS FOR ITS MANAGEMENT, ACM Computing Surveys, 1989, pág. 5-92, citado en Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998, pág. 183

Dependiendo de la estrategia escogida van a verse los modelos de análisis. En la mayoría de los casos es recomendable la alternativa (4) de Control Balanceado, la separación del control de otros tipos de funcionalidad. La razón para esto es tener alta localización en futuros cambios de funcionalidad.

Objetos de Entidad

Para modelar la información que el sistema va a manejar por un largo periodo de tiempo usamos objetos entidad. Típicamente tal información sobrevive escenarios, de este modo la información debe ser almacenada aún si el escenario ha sido completado. Además de la información que es manejada, también es necesario almacenar el comportamiento que pertenece naturalmente a esta información en el objeto entidad.

Los objetos de entidad son identificados de los escenarios como los objetos de interfaz. Muchos de los objetos de entidad son encontrados tempranamente y son obvios. Estos objetos de entidad "obvios" son muchas veces identificados en el modelo de objetos de dominio del problema de. Otros pueden ser difíciles de encontrar. Las entidades usualmente corresponden a algún concepto en la vida real fuera del sistema, sin embargo este no siempre es el caso. Es muy fácil modelar muchos objetos de entidad creyendo que es necesaria más información que la que realmente se necesita. Lo difícil es modelar sólo los objetos entidad que se necesita realmente. Es por lo tanto esencial trabajar de una manera estructurada cuando se modelan objetos de entidad. Las necesidades de los escenarios deben de ser los lineamientos establecidos y sólo los objetos de entidad que son justificados en las descripciones de los escenarios deben de ser incluidos.

Para almacenar información, los objetos usan atributos. Cada objeto entidad va a tener varios atributos. Cada atributo tiene un tipo, que puede ser un tipo de dato permitido, como un entero o string, o un tipo de dato compuesto. Un atributo es descrito como una asociación con un nombre y cardinalidad indicando el tipo del atributo (ver figura 3.16), note las similaridades entre ésta y la asociación de conocimiento. Los atributos pueden ser usados en todos los tipos de objetos para describir la información que será almacenada.

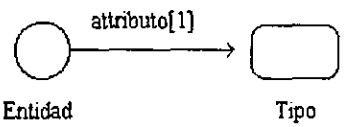


Figura 3.16. Atributo de un objeto

No siempre es fácil decidir cuándo una cierta parte de información debe ser modelada como un objeto entidad o como un atributo. Para ser capaces de decidir, debemos ver cómo será utilizada la información. La información que es manejada separadamente debe ser modelada como un objeto entidad, por otro lado la información que tiene un fuerte acoplamiento a alguna cierta información y nunca es usada por si misma debe ser colocada en atributos de un objeto de entidad. En otras palabras, lo que es decisivo es cómo los objetos manejan la información. Cierta información puede llegar a ser un objeto de entidad en un sistema, mientras que en otro llegaría a

ser un atributo. Si es usada para dos diferentes direcciones y por diferentes razones debe de formar un objeto de entidad separado.

Usualmente es fácil encontrar los objetos de entidad necesarios, pero es mucho más difícil identificar qué operaciones y cuales atributos pueden ofrecer esos objetos de entidad. La única manera de manipular un objeto de entidad es a través de sus operaciones. Por lo tanto las operaciones identificadas deben de ser suficientes para todos los que quieran utilizar el objeto entidad. La descripción detallada de los escenarios es un medio extremadamente valioso de encontrar las operaciones deseadas. Siguiendo la descripción, naturalmente se llega a las operaciones necesarias. Todo el curso de eventos es descrito en los escenarios y extrayendo esas partes concernientes a nuestro objeto entidad, van a aparecer las operaciones.

Las mismas reglas básicas que aplican a los objetos entidad se aplican a los otros objetos, esto es, todo comportamiento e información que es conectado naturalmente con el objeto entidad es colocado en él. De la misma manera es importante qué consecuencias tendrían con los cambios que se tuvieran. El objetivo es que cualquier tipo de cambio sea tan local como sea posible.

La siguiente lista es de las operaciones típicas que puede ofrecer un objeto de entidad.

- Almacenar y traer información.
- Comportamiento que debe cambiar si el objeto de entidad cambia.
- Crear y remover el objeto de entidad.

Una operación en un objeto de entidad puede significar que el objeto entidad contacte a otro objeto entidad y pregunte por información acerca de algo (figura 3.17). Esta comunicación toma lugar a través de las "asociaciones de comunicación". Una asociación de comunicación modela la comunicación entre dos objetos. A través de estas asociaciones un objeto manda y recibe estímulos. La asociación empieza del objeto que va a llevar a cabo la manipulación (esto es, el que manda el primer estímulo), y es dirigido al objeto donde tendrá lugar la manipulación. Como esto es una asociación de instancia, la línea es sólida y como es dinámica no se necesita nombrar la asociación.

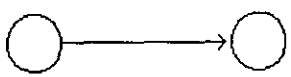


Figura 3.17. Objeto entidad solicitando información a otro objeto entidad

Cuando son modelados los objetos de entidad, se encontrará qué objetos de entidad similares ocurren en varios casos. Debe decidirse si se deben de separar los objetos entidad. Aún si los escenarios no hacen exactamente las mismas demandas en ellos, el objeto de entidad puede ofrecer operaciones que los escenarios usen en la forma que ellos quieran. La regla para decidir si dos objetos de entidad son actualmente uno, es que: si es la misma ocurrencia, debe de ser sólo un objeto de entidad; en caso contrario los objetos deben de quedar separados. Cuando se decide

que los objetos de entidad deben de ser combinados, operaciones asociaciones y atributos deben ser integradas también en el nuevo objeto entidad de esta forma la misma funcionalidad que existe en los objetos de entidad separados puede ser encontrada en el nuevo objeto de entidad. Considerando que es posible crear operaciones en objetos de entidad, esta posibilidad no debe de ser sobreutilizada. Las operaciones, después de todo van a ser diseñadas e implementadas después.

Objetos de Control

Ahora hemos partido el flujo de los escenarios en objetos de interfaz y objetos de entidad. En algunos casos todo lo del escenario ha sido colocado en objetos de estos dos tipos, para este caso no se necesitarían objetos de control para el escenario. Sin embargo en escenarios más complejos, muchas veces sobra comportamiento que no es colocado naturalmente en alguno de los dos tipos de objetos mencionados antes. Tal comportamiento es colocado en objetos de control. La razón por la cual dicho comportamiento es difícil de colocar en alguno de los otros dos tipos de objetos es que no pertenece realmente a la interfaz de el sistema o a cómo es manejada la información. Una posibilidad es dividir el comportamiento sobre esos dos tipos de objetos de todas maneras, como es sugerido por algunos métodos, pero la solución no es ideal desde el punto de vista de la variabilidad. Un cambio en tal comportamiento (muchas veces en la funcionalidad) puede afectar varios objetos y puede ser muy difícil de incorporar.

Los objetos de control generalmente actúan como pegamento que une a los otros objetos para formar un escenario. Es comúnmente el más efímero de todos los tipos de objetos y usualmente dura tanto tiempo como dure el desempeño de un escenario. Es sin embargo, difícil descubrir un balance entre qué es colocado en los objetos de entidad, de control y de interfaz. Aquí se dan unas heurísticas de cómo encontrarlos y especificarlos.

Los objetos de control son normalmente encontrados de los escenarios directamente. En un primer borrador asignaremos un objeto de control para cada escenario abstracto y concreto. Cada escenario normalmente envuelve objetos de interfaz y objetos de entidad. De esta forma el comportamiento que queda después de que a los objetos de interfaz y de entidad se les ha asignado su comportamiento, será colocado en los objetos de control.

Desviaciones de esta aproximación inicial, pueden ser hechas por diferentes razones. La primera es el caso extremo donde no hay comportamiento que quede para modelar. Un objeto de control, por supuesto, no se necesita. Si por otro lado, queda comportamiento muy complicado (después de la distribución de comportamiento relevante entre los objetos de entidad y de interfaz), la funcionalidad puede ser dividida en varios objetos de control. Ellos deben de tener tareas limitadas y deben de ser simples de entender y de escribir. Si un objeto de control se acopla a varios actores diferentes, esto indica que el comportamiento es diferente para los diferentes actores y debe ser dividido a través de varios objetos de control. El objetivo es tener un solo actor por cada objeto de control. La razón para esto es que los cambios del sistema muchas veces se originan por los actores y, si cada objeto de control es dependiente de un solo actor, entonces los cambios en el sistema pueden ser aislados.

Tipos comunes de funcionalidad colocada en objetos de control son :

- Comportamiento relacionado con la transición,
- Secuencias de control específicas de uno o varios escenarios,
- Funcionalidad que separa a los objetos de entidad de los de interfaz.

Los objetos de control tienen cursos de eventos y tienen comunicación con otros objetos.

La descripción de los objetos de control se encuentra directamente de los escenarios checando cómo el escenario corre sobre los otros tipos de objetos. En los lugares donde se necesite involucrar a un objeto de control, se tomará el rol del objeto de control y lo que involucra el rol y será colocado.

Las asociaciones entre escenarios son muchas veces mapeados a asociaciones de comunicación entre los objetos correspondientes. Si los escenarios tienen una asociación de extensión, ésta puede normalmente ser transferida directamente en una asociación de extensión entre objetos.

Subsistemas

Rara vez es posible tener una vista clara del número de objetos después de haberlos identificado en el análisis, así que los objetos necesitan ser colocados en grupos. Esto puede ser hecho a uno o varios niveles dependiendo del tamaño del sistema. Tales grupos de objetos son llamados "subsistemas". El sistema entonces consiste en un número de subsistemas. Al fondo de esa jerarquía están los objetos de análisis. Los subsistemas son entonces una manera de estructurar el sistema para su mejor desarrollo y mantenimiento.

La tarea del subsistema es empaquetar objetos, de esta forma la complejidad se reduce. Los subsistemas también trabajan manejando unidades en la organización, por ejemplo, desarrollando mercadeo, ventas, facturación. Un subsistema puede ser una unidad de sistema obligatorio pero también puede ser opcional.

El más bajo nivel de subsistema puede ser visto como unidad de cambio. A estas unidades se les llama "*paquetes de servicio*". Esto puede ser visto como atómico; si el cliente quiere, puede tener todo o nada. Cuando el sistema tiene un cambio, este cambio no corresponderá a más de un subsistema, o a objetos contenidos en este subsistema. Esto significa que el criterio más importante para esta división de subsistemas es predecir que el sistema cambiará y entonces hacer la división en base a esta aseveración. Un subsistema debe preferentemente ser acoplado a un actor, ya que los cambios son usualmente ocasionados por los actores.

La división en subsistemas debe ser basada en la funcionalidad del sistema. Todos los objetos que tienen un fuerte acoplamiento funcional deben ser colocados en el mismo subsistema. Para identificar objetos con un fuerte acoplamiento mutuo, podemos empezar con un objeto y estudiar su ambiente. Otro criterio para la división es que debe de haber tan poca comunicación entre los subsistemas como sea posible.

Lo que es más conveniente es empezar a buscar por subsistemas opcionales, éstos no son sólo aquellos que son opcionales para una función específica del sistema, debe ser considerada cualquier cosa que pueda ser opcional.

Después de que se han identificado los subsistemas opcionales, lo que queda muchas veces son esos objetos que son centrales al sistema y por lo tanto siempre tienen que ser repartidos. Para distribuir esto entre diferentes subsistemas, necesitamos buscar en la funcionalidad del sistema. Todos los objetos envueltos en una parte específica de la funcionalidad serán colocados en el mismo subsistema. Para identificar partes funcionales, podemos buscar en cada objeto. ¿Qué sucederá si quitamos este objeto?. Los objetos que lleguen a ser superfluos en una manera u otra, y son conectados al objeto removido o que se quitó, deben ser parte del mismo subsistema. Hay varios criterios similares los cuales pueden ser usados para decidir si dos objetos son fuertemente relacionados funcionalmente. Aquí hay una lista de algunos de ellos.

- ¿Los cambios en un objeto ocasionan cambios en el otro objeto?
- ¿Se comunican con el mismo actor?
- ¿Los dos son dependientes de un tercer objeto, como un objeto de interfaz o de entidad?
- ¿Un objeto lleva a cabo varias operaciones sobre el otro?

El objetivo es tener un fuerte acoplamiento funcional en un subsistema y un débil acoplamiento entre subsistemas. Un buen comienzo es colocar el objeto de control en un subsistema y después colocar los objetos de entidad y de interfaz fuertemente acoplados en el mismo subsistema.

Si se ve muy difícil colocar un objeto particular en un subsistema, puede ser colocado fuera de todos los subsistemas. Esto puede ser porque está completamente separado de los otros subsistemas, o porque tiene el mismo acoplamiento funcional de dos o más subsistemas.

Cuando se hace un división en subsistemas, en algunos casos puede ser deseable modificar los objetos de análisis. Este puede ser el caso, p.e., cuando un objeto entidad ha separado comportamiento que está relacionado funcionalmente con más de un subsistema. Si este comportamiento es extraído, puede ser más fácil colocar el objeto entidad en un subsistema.

Para expresar como están relacionados los subsistemas podemos asignar una relación de dependencia entre los subsistemas. Esta relación significa que los objetos de un subsistema van a utilizar, de alguna manera, objetos de otro subsistema.

La división de subsistemas en pequeños proyectos es normalmente hecha al final del análisis, cuando ya es clara la arquitectura. En proyectos grandes, sin embargo, muchas veces puede ser hecha mucho antes (en muchos casos antes de que se haya desarrollado el modelo de análisis). Para proyectos grandes hay otros criterios para la división de subsistemas, por ejemplo:

- Diferentes grupos de trabajo tienen diferente competencia o recursos y puede ser deseable distribuir conforme al trabajo a desarrollar (los grupos pueden estar geográficamente separados).

- En un ambiente distribuído, un subsistema puede ser requerido en cada nodo lógico.
- Si un producto existente puede ser usado en este sistema, ese producto puede ser visto como un subsistema.

En sistemas grandes es muchas veces esencial desarrollar el sistema en capas. En este caso un subsistema para la funcionalidad básica es desarrollado para las aplicaciones que serán construidas; las aplicaciones entonces usan esa funcionalidad base. Ver la figura siguiente.

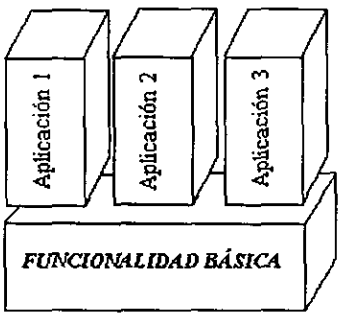


Figura 3.18. Sistema desarrollado en capas

3.3.2. CONSTRUCCIÓN

La construcción es dividida en dos fases, diseño e implantación, cada una desarrolla un modelo diferente ver la figura siguiente.

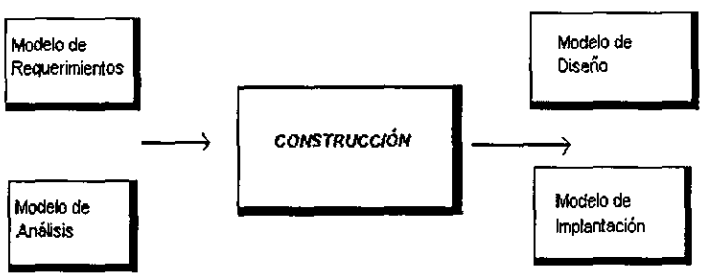


Figura 3.19. Fases de la construcción

El proceso de construcción finaliza hasta que el código es completado y las unidades de código son probadas. La construcción consiste en el diseño y la implantación. Si es posible se puede empezar la construcción antes de que el análisis sea completado.

Aquí hay tres razones para tener un modelo de construcción :

- El modelo de análisis no es suficientemente formal. Se deben refinar los objetos que ofrecen sus operaciones, exactamente cómo se verá la comunicación entre los objetos, que estímulo se envía y cosas así.

- El sistema actual debe ser adaptado al ambiente de implantación. En el análisis asumimos un mundo ideal para nuestro sistema. Pero en realidad no hay un mundo ideal, en realidad se deben de hacer adaptaciones para el ambiente en el cual el sistema será implementado. Esto significa que debemos inicialmente transformar el modelo de análisis de un espacio de tres dimensiones a un espacio de más dimensiones (ver figura 3.20). Debemos por ejemplo considerar requerimientos de desempeño de tiempo real y concurrencia, las propiedades del lenguaje de programación, el DBMS a usar etc.

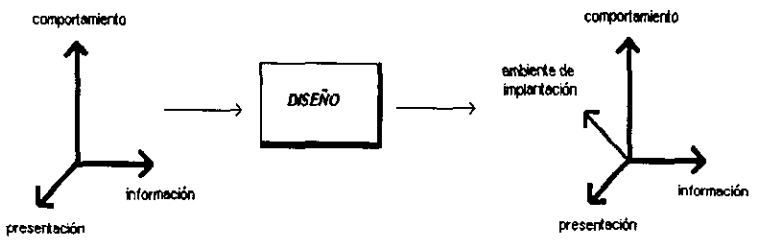


Figura 3.20. transformando el modelo de análisis

- Se quiere validar el resultado del análisis. Conforme nuestro sistema crece y llega a ser más formal, podemos ver cómo el modelo de análisis y requerimientos describen el sistema. Durante la construcción, podemos ver en etapas iniciales si el resultado del análisis va a ser apropiado para la construcción. Si descubrimos puntos que no son claros en el modelo de análisis o en el modelo de requerimientos se pueden aclarar quizá regresando al proceso de análisis.

Es posible que parezca que hay diferencias entre el resultado de la fase de análisis y la fase de diseño. Esto es un punto de vista incorrecto, ya que el propósito de la fase de análisis es entender el sistema y darle una buena estructura.

Los cambios en la arquitectura del sistema para mejorar el desempeño deben ser pospuestos por regla hasta que el sistema esté parcialmente construido. Para hacer suposiciones correctas necesariamente se tiene que tomar en cuenta la optimización del desempeño, en muchos de los casos, se necesita algo para medir. De otra forma sólo se harán suposiciones las cuales pueden o no ser apropiadas. Como no se tiene nada que medir hasta que el sistema ha sido

construido, no podemos hacer estas optimizaciones en etapas tempranas. Una forma de evitar este dilema es simular las partes críticas del sistema. Otra forma es desarrollar prototipos para tener una opinión temprana de cómo lucirá el sistema, pero éste es un método riesgoso ya que se puede fácilmente hacer simplificaciones no realistas. Un prototipo siempre tiende a resaltar características particulares. Las conclusiones generales sobre otras características no deben ser esquematizadas en un prototipo que no las resalte.

3.3.2.1. MODELO DE DISEÑO

El modelo de diseño es un refinamiento y formalización del modelo de análisis, donde son tomados en cuenta las consecuencias del ambiente de implantación.

Para desarrollar el modelo de diseño se siguen 3 pasos simples:

1) Identificar el ambiente de implantación: este paso icluye identificar e investigar las consecuencias que el ambiente de implantación tendrá en el diseño. Aquí todas las decisiones de implantación estratégica deben de ser hechas. ¿Cómo será incorporado el DBMS en el sistema?, ¿Cuáles y cómo serán usadas las librerías de componentes?, ¿Cómo será manejado el proceso de comunicación? etc.

2) Incorporar esas conclusiones y desarrollar una primera aproximación al modelo de diseño: aquí usamos el modelo de análisis como una base y traducimos los objetos de análisis en objetos de diseño en el modelo de diseño que se apegan al ambiente de implantación actual. Desde la perspectiva del proyecto esto debe ser directamente incorporado en el modelo de análisis, pero para propósitos de mantenimiento y de entendibilidad esto no es recomendado. Cuando se hace un desarrollo más amplio, el modelo de análisis forma las bases lógicas para entender el sistema y es un modelo esencial para mantener claramente todo el ciclo de vida del sistema.

3) Describir como interactúan los objetos en cada escenario específico: aquí el modelo de diseño es formalizado para describir todos los estímulos enviados entre los objetos y también para definir qué va a hacer cada operación de cada objeto. El modelo de escenarios va a ser de gran ayuda durante este trabajo y nos va a ayudar a especificar cada flujo específico en detalle del sistema. Este paso nos da las interfaces de los objetos.

Del modelo de diseño obtenemos especificaciones detalladas de todos los objetos, incluyendo sus operaciones y atributos.

El modelo de diseño va a ser un refinamiento más amplio del modelo de análisis en el sentido del ambiente de implantación actual. El modelo de análisis se ha desarrollado asumiendo condiciones ideales. Se deben adaptar ahora a la realidad, existen dos poderosas razones para no introducir el ambiente de implantación antes. La primera es que no deseamos afectar la estructura básica del sistema, ya que las circunstancias actuales pueden cambiar de alguna manera u otra en el ciclo de vida del sistema. La segunda es que no queremos que nuestro problema se vea afectado por la típica complejidad introducida con el ambiente de implantación. De esta manera

podemos enfocarnos en lo esencial cuando desarrollamos los aspectos más importantes del sistema, llamándole la estructura básica.

El espacio de diseño expande el espacio de análisis, incluyendo una dimensión para el ambiente de implantación. Esta dimensión significa que se han introducido nuevos conceptos, por lo cual el modelo de análisis debe ser adaptado. Ver figura siguiente.

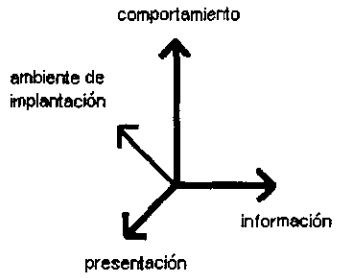


Figura 3.21. Dimensión del ambiente de implantación

Esto significa que se quiere adaptar el modelo de análisis al ambiente de implantación al mismo tiempo que lo refinamos. Nuestro objetivo es refinarlo mientras es sencillo escribir su código fuente. Como el modelo de análisis tiene todas las propiedades que queremos para el sistema, queremos que esta estructura forme la base del modelo de diseño. Sin embargo, van a existir cambios cuando se introduzca un DBMS relacional, ambiente distribuido, requerimientos de desempeño, un lenguaje específico de programación etc., esta es la razón por la cual se desarrolla un nuevo modelo.

Por un lado se quiere hacer todo el trabajo que sea posible en el modelo de análisis, donde podemos enfocarnos en lo esencial, pero por otro lado no se debe de hacer tanto que se necesite cambiarlo para adaptarlo al ambiente de implantación. Lo que realmente se quiere es un continuo refinamiento en los modelos en donde el cambio de modelos ocurra, para empezar a ver las consecuencias del ambiente de implantación.

Se usará el concepto de "bloque" para describir la intención de cómo deberá ser producido el código. Los bloques son los objetos de diseño y son dibujados como rectángulos como se muestra en la figura 3.22. Un bloque normalmente tiende a implementar un objeto de análisis. Aquí puede ser posible utilizar diferentes tipos de bloques, es preferible bloques equivalentes a los tipos de objetos, es decir: bloques de interfaz, bloques de control y bloques de entidad para observar la rastreabilidad. Es importante observar que los bloques no son los mismos objetos que los objetos de análisis. Estos bloques serán implementados más tarde como código fuente.

Figura 3.22. Esquema de un bloque

Los bloques van a abstraer la implantación real. La implantación de los bloques puede ser una clase específica en el código fuente, esto es, un bloque es implementado por un clase. Sin embargo muchas veces un bloque es implementado por varias clases diferentes. De todas formas los bloques son una forma de abstraer el código fuente.

El nivel de módulo de un lenguaje de programación lo denotamos genéricamente como módulo objeto. En lenguajes de programación estos módulos serán clases. Aquí usaremos los términos objeto, módulo o clase.

El primer intento para el modelo de diseño puede ser hecho mecánicamente basado en el modelo de análisis. Inicialmente cada objeto de análisis va a llegar a ser un bloque. Esta regla de transformación significa que obtenemos una clara rastreabilidad entre los modelos. Empezamos modelando el sistema en el modelo de análisis de una manera que provea una estructura robusta para el sistema. Como cada objeto de análisis es rastreable a un bloque, los cambios introducidos en el modelo de análisis van a ser locales al bloque correspondiente en el modelo de diseño y así también en el código fuente. Nótese que la rastreabilidad es bidireccional.

La rastreabilidad es una propiedad tremendamente importante en el desarrollo del sistema. Cada sistema va a ser alterado durante su ciclo de vida. Si los cambios emanan del cambio de requerimientos o como respuesta a problemas, siempre necesitaremos saber dónde deberán ser hechos los cambios en el código fuente.

Aquí tenemos una gran ventaja de rastreabilidad; se puede encontrar fácilmente la parte en el sistema, aún si ha sido sometido a grandes cambios. Es importante tener una alta localización funcional con el sentido de conocer qué cambios no van a influir grandes partes del sistema.

La semántica del modelo de diseño es algo diferente al modelo de análisis. El modelo de análisis es desarrollado en términos lógicos y es sólo un esquema conceptual del sistema a construir. Por lo tanto es fundamental mantener y congelar el modelo de análisis para mantenimiento futuro aún después de que el modelo de diseño es finalizado. El modelo de diseño sin embargo es una abstracción de cómo es construido realmente el sistema actual. El primer intento de este modelo es un mapeo directo del modelo de análisis. En la estructura final, sin embargo, va a reflejar cómo el ambiente de implantación ha afectado la construcción. El objetivo es mantener la estructura encontrada en el modelo de análisis y no viciarla innecesariamente en el diseño.

La semántica de los bloques debe reflejar la semántica de los objetos que existen en el sistema y las asociaciones entre los objetos deben también reflejar cómo están realmente relacionados los objetos en el sistema. Por ejemplo: en muchos lenguajes de programación no se tiene ninguna manera de implementar la asociación de extensión. Durante el diseño debemos decidir cómo será implementada esta asociación y cambiar el modelo de diseño de tal forma que lo refleje.

Esta es consecuentemente la primera idea de arquitectura de un sistema. Enfatizamos que es la primera porque puede cambiar cuando se considera el ambiente de implantación. Podemos tener que romper o dividir los bloques porque se quieren distribuir en diferentes niveles en nuestro sistema de computadora, o podemos necesitar también nuevos bloques para encapsular completamente un DBMS existente.

El modelo de diseño nos habilita para reducir la complejidad del sistema. Los bloques son actualmente un mecanismo de abstracción para el código fuente.

Se ha notado que cualitativamente es más fácil evitar errores de construcción reduciendo la complejidad en etapas previas que buscar en el sistema por fallas y corregirlas cuando el sistema está ya completo

Para adaptar el modelo de diseño al ambiente de implantación actual debemos primero identificar las condiciones técnicas actuales bajo las cuales debe ser construido el sistema.

¿Qué debemos incluir en el ambiente de implantación?. Las cosas más claras suelen ser el ambiente destino, (donde se va a ejecutar el sistema), lenguaje de programación, DBMS, pero de hecho todo lo que afecte la realización del sistema debe ser incluido en este concepto.

Debido a que uno de los cambios más frecuentes al sistema es en el ambiente de implantación, es preferible manejarlo de la misma manera "variable" usada para todo el sistema. Esto implica que tan pocos objetos como sea posible deben de saber de las restricciones del ambiente de implantación actual. Por lo tanto la estrategia global para manejarlo es encapsularlo lo más posible. De esta manera cualquier cambio del ambiente será local y no afectará a varios objetos.

Si queremos tener la posibilidad de cambiar partes en el ambiente destino, estas partes deben ser encapsuladas en un nuevo bloque. Se deben de crear entonces nuevos bloques que representen ocurrencias en el ambiente destino.

El lenguaje de programación utilizado afecta el diseño, las propiedades básicas del lenguaje y su ambiente son fundamentales para el diseño. La existencia de herencia, polimorfismo, tipos de datos etc, son ejemplos de tales propiedades.

También la estrategia de manejo de memoria debe ser escogida tempranamente. Algunos lenguajes OO tienen una recolección automática de basura, pero en otros es obligación del programador limpiar las instancias que ya no se usen.

Algo que está muy relacionado con el lenguaje de programación es el uso de componentes para programación. El uso de librerías de componentes puede afectar el diseño. Un ejemplo es el diseño de objetos de interfaz. Hay componentes (o herramientas) para construir interfaces, y, esto puede afectar la forma en que se hizo el diseño.

Muchas veces existen productos que son usados cuando se desarrollan nuevos sistemas. La estrategia normal para manejar este producto es introducir nuevos bloques para encapsularlos de la misma manera que se describió el ambiente destino.

Otras implicaciones vienen de los estándares y reglas de codificación.

Durante la construcción debemos de tomar en consideración cualquier requerimiento para desempeño o limitación de memoria. Estos requerimientos también pueden afectar el diseño.

Para investigar tempranamente problemas potenciales de optimización, la simulación o el prototipo pueden ser utilizados antes de que sea hecho el diseño real. De esta forma los diseñadores tienen mucho mejores bases dónde deben de ser hechas las optimizaciones reales. Por supuesto, la experiencia en el área de aplicación puede ayudar a juzgar de forma temprana dónde deben hacerse las optimizaciones. El mensaje aquí es que si no estás seguro de una corrección o desempeño de una optimización, no se debe de hacer hasta que se esté seguro de cómo debe hacerse.

La gente y organización envuelta en el desarrollo puede afectar el diseño.

Los cambios hechos en el modelo de diseño no siempre afectan al modelo de análisis siempre y cuando se manejen como decisiones de diseño, se observa la diferencia con los términos usados, en una decisión de diseño se utiliza un bloque y en el análisis un objeto.

Se puede tener que cambiar el diseño ideal en varias formas:

- Introduciendo más bloques en el modelo de diseño que no tienen ninguna representación en el modelo de análisis.
- Borrar bloques del modelo de diseño.
- Cambiar bloques en el modelo de diseño (dividir y unir bloques existentes).
- Cambiar las asociaciones entre los bloques en el modelo de diseño.

Normalmente, adicionar bloques para manejar el ambiente es un buen cambio. Adicionar bloques para otra funcionalidad no debe ser normalmente hecho, ya que se deberían de introducir a través del modelo de análisis. Borrar bloques es más sospechoso. ¿Por qué se borra el bloque? Si se tienen buenas razones para ello (muchas veces razones de implantación), está bien, pero si se está cambiando la estructura lógica del sistema, tal cambio debe de hacerse primero en el modelo de análisis.

Dividir y unir bloques son también cambios sospechosos. Cualquiera de estos cambios va muchas veces a decrementar o reducir la robustez del sistema y debe ser hecha con mucho cuidado. Por razones de implantación o desempeño está justificado, pero el diseñador debe dedicar una gran cantidad de pensamiento a tales cambios.

Cambios de asociaciones son muchas veces los cambios más comunes en el modelo de diseño. Estos cambios muchas veces vienen del modelo de implantación.

Cualquier asociación de herencia debe ser revisada si no se puede implementar en el lenguaje de programación.

DIAGRAMAS DE INTERACCIÓN

Vemos los bloques como una abstracción de la implantación actual del sistema. Los bloques van a agrupar el código fuente. Para conocer cómo implementar los bloques, debemos refinar el modelo de diseño. Esto lo hacemos describiendo cómo se comunican los bloques durante su ejecución. Para describir la comunicación entre los bloques usamos "estímulos". Un estímulo es enviado de un bloque disparando una ejecución en el que recibe. Esta ejecución puede mandar nuevos estímulos a otros bloques.

Para escribir una secuencia de estímulos, usamos "diagramas de interacción". En estos podemos describir, como varios bloques se comunican mandándose estímulos entre sí. Como una base para estos diagramas de interacción utilizamos de nuevo el modelo de escenarios.

Para cada escenario concretamente se dibujará un diagrama de interacción, el diagrama de interacción describe como cada escenario es creado por la comunicación de objetos. El diagrama muestra como los objetos participantes realizan el escenario a través de su interacción.

La interacción toma lugar cuando los bloques se mandan estímulos entre sí. En los diagramas de interacción se definen todos los estímulos, incluyendo sus parámetros, el principal propósito del diseño de escenarios es definir los protocolos de los bloques.

Con el modelo de escenarios hemos descrito qué se realiza en cada escenario. El modelo de análisis describe cuales objetos ofrecen ese comportamiento, ahora en el modelo de diseño se refina la descripción de los escenarios mostrando en el diagrama de interacción cómo se comportarán los objetos en cada escenario específico.

Por otro lado el modelo de escenarios implica una gran cantidad de trabajo. Aquí se debe definir exactamente cómo los objetos participantes se van a comunicar.

Cuando se diseña un escenario se empieza identificando los bloques que participan en ese escenario. Esto se hace fácilmente buscando qué bloques ofrece el escenario en el modelo de análisis; los objetos correspondientes serán incluidos en el diseño. Esta identificación es por lo tanto un proceso completamente mecánico. Además puede haber nuevos bloques introducidos

durante la construcción (por ejemplo para el ambiente de implantación que también participan en el escenario).

El diagrama de interacción tiene la misma función de tratar de descubrir la comunicación entre los diferentes bloques.

Cada bloque participante es representado por una barra. Estas barras son dibujadas como líneas verticales en el diagrama. El orden entre las barras es insignificante y debe ser seleccionado para dar una mayor claridad. El objetivo es obtener una buena apreciación global.

Si va a haber varias instancias de clases de bloques, se puede dibujar las instancias como barras diferentes. Todo el comportamiento de un objeto va a ser pegado a la barra que representa el bloque actual.

A veces se tiene que utilizar polimorfismo en el diagrama de interacción; ésta es una técnica muy útil.

En todos los diagramas de interacción se tiene también una barra para el mundo que lo rodea, para limitar qué es lo que está afuera de lo que queremos describir. Usualmente esta barra se llama "borde del sistema". Esta barra representa la interfaz con todo fuera de los bloques en el diagrama, como actores externos, y consecuentemente puede corresponder a diferentes interfaces fuera del sistema.

El eje del tiempo en el diagrama de interacción es visto yendo hacia abajo. El escenario entonces empieza con el comportamiento descrito en el principio del diagrama de interacción. El eje del tiempo del diagrama de interacción no es lineal, sino que debe tomarse en cuenta como un controlador de eventos. La distancia entre dos eventos del diagrama no tiene relación al tiempo real entre esos eventos.

A la izquierda del diagrama de interacción, a la izquierda del borde del sistema, se pueden describir las secuencias. Esta descripción es textual y consiste en texto estructurado o pseudocódigo. Si usamos pseudocódigo, debe ser construído con el lenguaje de programación a utilizar. Esto es para facilitar nuestra migración a la implantación. Sin embargo esto nos hace dependientes de un lenguaje específico. El texto describe qué está sucediendo en esta parte del escenario. Tal parte la llamamos "operación". También marcamos la barra a la cual pertenece la operación con un rectángulo representando la operación. La descripción textual consecuentemente pertenece al bloque donde la operación toma lugar y donde posteriormente será implementada, la barra es por lo tanto marcada en el lugar de la operación. Entre operaciones podemos dibujar líneas punteadas horizontales para que sea más claro. Esta técnica puede ser usada para cualquier partición del escenario(ver figura 3.27).

Los diagramas de interacción son controlados por eventos. Un nuevo evento da origen a una nueva operación. Estos eventos son estímulos que son enviados de un objeto a otro e inician una operación.

Un estímulo es dibujado o esquematizado en los diagramas de interacción como una flecha horizontal que inicia en la barra correspondiente al bloque que envía y termina en la barra correspondiente al bloque que recibe. La mayoría de los diagramas de interacción (y también los

escenarios) comienzan con un estímulo externo. Este es esquematizado o dibujado desde el borde del sistema. De la descripción de escenarios podemos hacer un diseño de escenarios, se deben de considerar las fallas o errores que puedan ocurrir.

Cuando se trabaja en el diseño de escenarios se van a definir los estímulos que cada objeto va a ser capaz de recibir. La definición encierra el nombre y los parámetros de cada estímulo. Definir estímulos es una de las partes más críticas del desarrollo, ya que varias personas están involucradas en el diseño de escenarios.

Cuando se define un estímulo se debe de tener en mente :

- La reusabilidad incrementa si se tienen pocos parámetros. Teniendo pocos parámetros va a ser fácil de entender el estímulo e incrementa la posibilidad de que el estímulo sea similar a otro y reutilizable. Un estímulo con varios parámetros debe ser redefinido y posiblemente dividido en varios estímulos.

- El estímulo que invoque comportamiento similar debe de tener el mismo nombre independientemente de los parámetros que contenga o a los objetos que sea enviado. No hay conflicto entre nombres duplicados. Si el estímulo tiene los mismos nombres es fácil de ver con claridad similaridades entre dos estímulos.

- El nombre del estímulo debe de reflejar la distribución de responsabilidades entre bloques para hacer cada bloque tan independiente como sea posible, es uno de los principios básicos en todos los contextos de ingeniería de software

- Cada estímulo debe mostrar claramente el intercambio de información, esto es, el nombre debe ser seleccionado cuidadosamente. El nombramiento es una de las habilidades más difíciles en programación y por lo tanto muchas veces existen reglas para este propósito. Los nombres indican una semántica intuitiva; consecuentemente el nombre es una guía extremadamente importante para entender y encontrar el estímulo que será reutilizado.

- El manejo y creación de nuevas instancias o iniciación de nuevos procesos son desarrollados en la misma manera como el manejo ordinario de estímulos.

La descripción de un escenario es dividida normalmente en cursos básicos y cursos alternativos. El curso básico es la secuencia común (la más importante) en el escenario. Siempre se diseña primero. Los cursos alternativos constituyen todas las otras secuencias que el escenario puede seguir, típicamente secuencias de manejos de errores. Una vez que se ha diseñado el curso básico se continúa con el(los) curso(s) alternativo(s).

En la medida que se describan la mayoría de las secuencias alternativas, la mayoría de las secuencias serán anticipadas y será en la medida que se incremente la robustez en el diseño del sistema.

Un estímulo puede tener diferentes semánticas, p.e. puede ser una comunicación interproceso o intraproceso. Un estímulo intraproceso es una llamada normal dentro de un proceso. Normalmente a esto se le llama "mensaje". Esto es, es el modo normal de comunicación

en lenguajes de programación orientada a objetos. Un estímulo interproceso es un estímulo enviado entre dos procesos. Usualmente llamamos a ese estímulo señal. Las señales pueden ser síncronas y asíncronas (la ejecución del que envía continúa después de que la señal es enviada).

Cuando una señal se envía el que la envía continúa en ejecución inmediatamente y no espera por el receptor. El receptor puede estar activo cuando llegue la señal y así la señal se encola hasta que el receptor la procese.

Las señales y los mensajes son esquematizados de diferente forma en los diagramas de interacción como se muestra en la figura 3.23. Los mensajes son dibujados con una flecha cerrada y las señales con una flecha abierta.

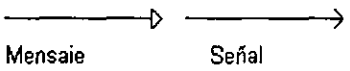


Figura 3.23. Representación de mensajes y señales.

Los diagramas de interacción dan al diseñador una habilidad única de ver la secuencia completa en un escenario, un esquema de cómo progresa la secuencia sobre los objetos que participan en el escenario. El poder ver como luce la estructura de un escenario muchas veces nos dice mucho acerca de las características de la realización del escenario.

En el siguiente diagrama (figura 3.24) podemos ver que todo es manejado y controlado por un "objeto1". Este objeto controla el flujo y opera sobre los otros objetos, y decide qué hacer. Este objeto debe contener las reglas para realizar algo. Aquí se tiene una estructura "centralizada".

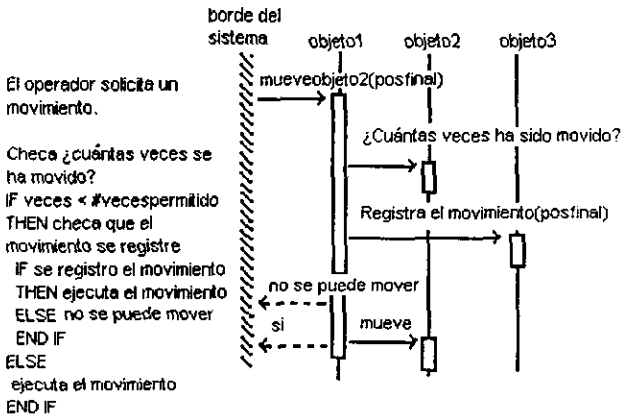


Figura 3.24. Esquema centralizado

En la segunda figura (figura 3.25) se ha delegado la decisión a un "objeto2" a ver si se puede o no mover por sí mismo, de esta forma el "objeto1" sólo pregunta si la carga ha sido movida o no. El "objeto2" checa si se puede o no mover y es lo que regresa al "objeto1" y dependiendo de eso el "objeto1" mueve o no la carga.

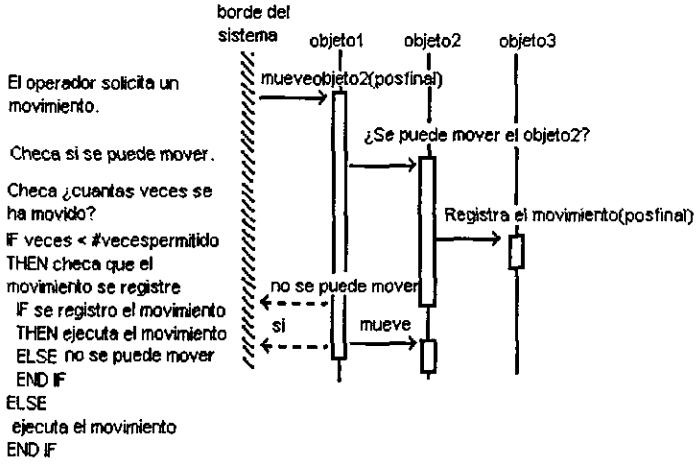


Figura 3.25. Esquema descentralizado

Aquí vemos que las decisiones son descentralizadas y son hechas por el conocimiento de las condiciones. Tenemos por lo tanto una estructura "descentralizada". La diferencia entre estos escenarios es fundamental y esta viene de la fase de análisis donde se han almacenado los escenarios en los objetos de análisis. ¿Cuál de estos escenarios es más sensible a los cambios?.

En el diagrama de interacción podemos ver la estructura del escenario claramente. El diagrama de interacción nos ayuda a ver como es el sistema descentralizado. Podemos distinguir dos tipos extremos de la estructura para los diagramas de interacción (figura 3.26).

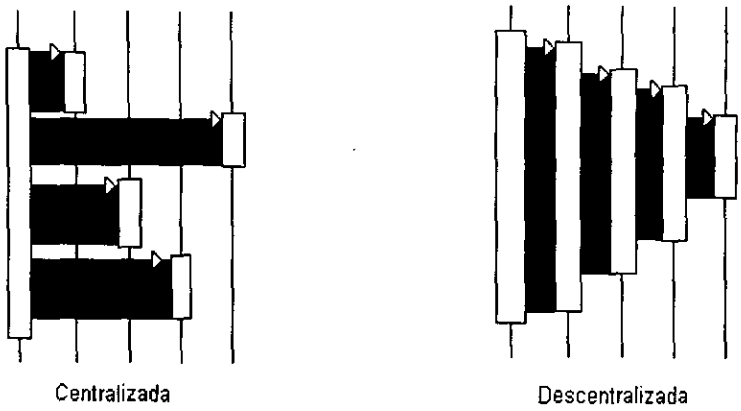


Figura 3.26. Tipos de diagramas de interacción

El primer extremo lo llamamos "diagrama de tenedor" (fork diagram). Este tipo es caracterizado por el hecho de que es un objeto actuando como araña en una telaraña y controlando los otros objetos. La mayoría del comportamiento es colocado en el objeto controlador el cual conoce todos los demás objetos y muchas veces los usa con preguntas directas o comandos.

Un diagrama de tenedor indica una estructura centralizada. Típicamente la secuencia de control es colocada en un objeto, muchas veces un objeto de control. Los otros objetos típicamente son utilizados como correos de información o como una interfaz de usuario. Una estructura de tenedor muchas veces coloca una gran responsabilidad en el diseñador del objeto controlador ya que llegará a ser más complejo que los otros objetos.

El otro extremo lo llamamos "diagrama de escalera". Este es caracterizado por la delegación de responsabilidad. Cada objeto sólo conoce un poco de los otros objetos y sobre que objetos pueden ayudar a un comportamiento específico. Aquí no se tiene un objeto central.

Los diagramas de escalera muchas veces indican una estructura descentralizada. Cada objeto tiene una tarea separada y sólo conoce los objetos circundantes si los necesita para obtener ayuda para realizar su tarea.

¿Cuál de estos dos tipos de esquemas se debe de elegir?, ¿Cuál es mejor?. Muchas veces se ha afirmado que la estructura de escalera es más orientada a objetos por lo tanto es mejor. La mayoría de responsabilidades se ha dividido. Sin embargo esto no siempre es cierto, lo que queremos es ser capaces de introducir cambios fácil y también diseñar objetos y estímulos reutilizables. Pueden ocurrir diferentes tipos de cambios. Para manejar cambios de cómo una secuencia es desarrollada actualmente, se debe encapsular la secuencia actual y después obtener una estructura descentralizada. Si se quiere cambiar el orden actual de la operaciones, es mejor tener una estructura centralizada, ya que los cambios serían locales a un objeto. Vemos que las dos estructuras tienen sus beneficios.

Hemos encontrado las siguientes reglas principales :

- Una estructura de control descentralizado (escalera) es apropiada cuando :
  a) Las operaciones tienen una fuerte conexión.
  b) Las operaciones siempre se desarrollan en el mismo orden.

- Una estructura de control centralizada (tenedor) es apropiada cuando:
  a) Las operaciones pueden cambiar de orden.
  b) Se pueden insertar nuevas operaciones.

Esto nos lleva a la conclusión de que estas dos estructuras pueden y deben ser combinadas para obtener una estructura robusta y estable.

Hemos visto que el orden de las operaciones es la fuente potencial de los cambios. La solución natural para la robustez es encapsular las cosas susceptibles de cambios. Esto es exactamente lo que se hace en la aproximación centralizada donde el ordenamiento de las operaciones está definido en un objeto.

La posibilidad de usar un escenario para extender otro es descrita por medio de una asociación de extensión entre los dos escenarios. En el diagrama de interacción esto es descrito por una "posición de prueba" (posición de sondeo).

La posición de sondeo especifica dónde debe de ser insertada la secuencia. Por una posición de sondeo entendemos la posición en una descripción de escenario o su diagrama de interacción correspondiente, donde, cuando un escenario sigue esta descripción, puede ser insertado otro comportamiento. Ya que la descripción original debe ser independiente (no tiene conocimiento) del nuevo comportamiento insertado, toda la estructura de control debe ser colocada en el bloque que contiene el escenario insertado.

Desafortunadamente no podemos lograr esto con los lenguajes de programación hoy en día, y estamos forzados a desviarnos de ese ideal. En lugar de que la sonda sea implementada en el bloque donde la secuencia debe de ser insertada debemos adicionar una variable para guardar una referencia a la funcionalidad insertada y también el estímulo enviado a este objeto

Una posición de prueba indica una posición en el escenario que se va a extender y muchas veces es acompañada por una condición, la cual indica bajo qué circunstancias debe de llevarse a cabo la extensión. La posición de prueba es descrita en el diagrama de interacción del escenario de la extensión e indica una posición en el diagrama de interacción del escenario a ser extendido. La posición de prueba entonces pertenece al escenario que extiende y no al extendido; esto es

permitido cambiando el escenario original cuando son adicionadas nuevas extensiones. Ver la figura siguiente para ver la posición indicada por la sonda (prueba) en la descripción de una operación en un bloque.

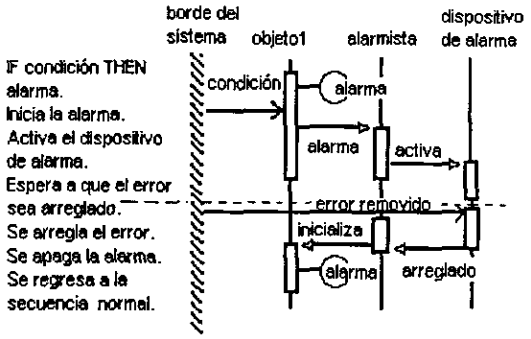


Figura 3.27. Posición de sonda (prueba) en la descripción de un bloque

La semántica de la sonda es como sigue. El escenario original corre sobre su descripción. Cuando el escenario alcanza la posición de la sonda checa si la condición de extensión es verdadera. Si es, el escenario ahora empieza a seguir la descripción del escenario de extensión. Cuando alcanza el final de la descripción, regresa a la descripción original y continua donde se quedó.

De la descripción en el diagrama de interacción debemos ser capaces de encontrar exactamente donde será insertada la sonda.

Los escenarios son normalmente desarrollados en paralelo y por varios diseñadores que pueden trabajar más o menos independientemente que otros. Esto significa que los estímulos son definidos por varios diseñadores diferentes. Normalmente se debe de estar de acuerdo en las reglas de definición de estímulos, sobre todo para el proyecto o el producto, pero preferentemente en la organización entera.

Debemos de homogeneizar todos los estímulos cuando los escenarios han sido diseñados. La homogeneización puede por supuesto también ser hecha como trabajo sobre el diseño de los procedimientos de los escenarios.

La homogeneización significa que trataremos de tener tan pocos estímulos como sea posible, y todos ellos deben de ser mayormente reutilizables y fácil de trabajar con ellos. Es difícil dar reglas estrictas pero la reusabilidad y robustez deben de ser el principal criterio.

Cuando se han diseñado todos los escenarios, o casi todos los escenarios de un bloque específico, podemos diseñar el bloque. A través del diseño de los escenarios tenemos una descripción completa de todos los requerimientos externos del bloque.

A través del diseño de escenarios tenemos implícitamente especificado el protocolo de cada bloque. Tomando todos los diagramas de interacción donde participa el bloque y extrayendo todas las operaciones definidas para ese bloque tendremos un esquema completo de las interfaces del bloque. Estas interfaces deben de aparecer durante el diseño de escenarios donde normalmente varias gentes son involucradas y las interfaces no son recolectadas regularmente en un documento. Las interfaces existen explícitamente, y son conectadas al bloque apropiado. Por lo tanto es simple extraer esas interfaces mecánicamente.

Yendo a través de todos los diagramas de interacción, vamos a obtener la interfaz completa de un bloque.

Hasta esta etapa es posible congelar la interfaz de los bloques, se puede iniciar en paralelo las actividades de diseño del bloque.

Además de los requerimientos establecidos por el diseño de escenarios hay otros requerimientos sobre el bloque. Estos requerimientos pueden ser identificados del modelo de análisis, donde encontramos por ejemplo, qué atributos y tipos de atributos debe tener el bloque.

En muchos casos un bloque va a corresponder exactamente a una clase y sus instancias. Entonces es fácil mapear la interfaz de un objeto en el modelo de diseño a una interfaz en una clase específica en código fuente. Otras veces, como hemos visto, varias clases deben ser usadas para implementar un bloque. En la mayoría de los casos, la razón para utilizar varias clases es manejar el ambiente de implantación, pero otras cosas comunes incluyen la implantación de atributos, uso de componentes y extracción de funcionalidad común en clases abstractas.

Para manejar la interfaz de un bloque podemos utilizar conceptos del lenguaje de programación como se ha mostrado. Sin embargo los bloques son una abstracción también y un mecanismo de encapsulación. De esta forma queremos expresar la interfaz de un bloque más explícitamente y también ser capaces de encapsular su implantación. Esto lo hacemos definiendo métodos privados y públicos.

Esconder los detalles de la implantación interna es una manera de manejar la complejidad del código fuente, lo que realmente se necesita es sólo ser capaz de especificar una interfaz para un rol que el bloque deba exportar. Para otros implementadores de bloques que necesiten programar utilizando nuestro bloque, necesitamos definir los tipos que ofrecemos y las operaciones que ofrecemos sobre esos tipos.

DIAGRAMAS DE TRANSICIÓN DE ESTADOS

Como un nivel intermedio para las entrañas del objeto antes de buscar la implantación, podemos usar diagramas de transición de estados. Su propósito es proveer una descripción simplificada que incremente el entendimiento del bloque sin necesidad de ir al nivel del código fuente, y proveer una descripción que es menos dependiente del lenguaje de programación seleccionado. En estos esquemas describimos que estímulos pueden ser recibidos y qué sucederá cuando un estímulo es recibido.

Para que un estado esté absolutamente correcto y completo, debe de ser la unión de todos los valores que describen la situación presente.

Los estados y transiciones de estados pueden ser descritos de muchas maneras. Se debe de tener en cuenta que una transición es dependiente sobre el estímulo recibido y sobre el estado interno expresado con algunas condiciones.

Existen varias técnicas que pueden ser utilizadas para describir la computación de un objeto, para gráficas de transición de estados. La técnica escogida para la descripción no es importante; lo realmente importante es que nos ayude para abstraer el código actual. Para nuestros ejemplo usaremos una notación estándar SDL (lenguaje de especificación y descripción). La notación se describe en la figura 3.28.

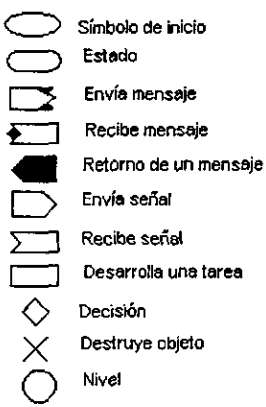


Figura 3.28. Notación SDL

Estos símbolos están conectados para describir la computación de un objeto. El símbolo de "inicio" indica dónde comienza la creación de un objeto. Un símbolo de "estado" muestra un estado estable de un objeto entre diferentes operaciones. "Envía mensaje" desarrolla una operación sobre otro objeto y corresponde a un "recibe mensaje" en el objeto a ser operado. Para

describir el regreso de una operación, un símbolo de "return" es usado. "Enviando" y "recibiendo señales" son denotados por los símbolos correspondientes. La computación actual, no incluye la comunicación explícita, es decir, por una caja desarrolladora de tareas. El símbolo de "decisión" es usado para describir alternativas en la ejecución y el "destruir" objeto describe cuando una instancia es borrada. El símbolo de "nivel" es usado para hacer los diagramas fáciles de leer y muchas veces es usado para describir ciclos.

Cada operación debe de empezar bajo un estado, o con el símbolo de "inicio" y termina en un estado (posiblemente el mismo estado). Esto significa que después de cada estado debemos tener un símbolo de "recibir" un "recibe mensaje" o "recibe señal". El símbolo de "return" es usado solamente si tenemos un valor de regreso. Si no es explícito el valor de regreso, la condición por default es que un mensaje recibido es finalizado justo antes del siguiente estado. Ver un ejemplo en la figura 3.29.

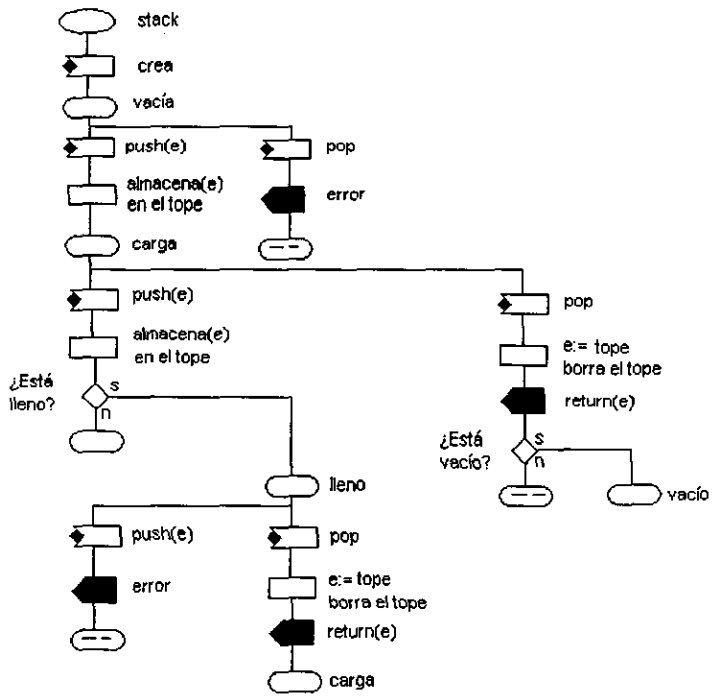


Figura 3.29. Ejemplo de la notación SDL

Esta notación se mapea directamente sobre los diagramas de interacción presentados anteriormente (ver figura 3.30). En inciso a) tenemos una operación extraída de un diagrama de interacción. Esta operación es descrita por la transición de estados en b). Estas secuencias pueden ser generadas automáticamente del diagrama de interacción. Sin embargo, en el caso

general la semántica exacta no puede ser generada. También el diagrama de interacción describe una ruta específica a través del gráfico, pero el gráfico debe describir todas las rutas. Por lo tanto sólo esqueletos o estructuras de rutas pueden ser generados y el desarrollador debe de relacionar las rutas y estados con cada uno.

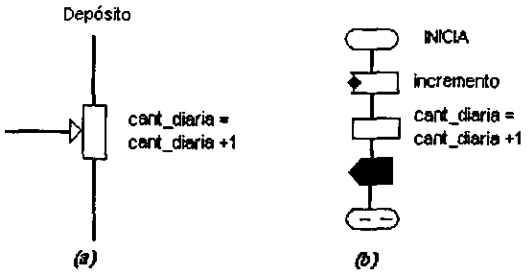


Figura 3.30. Mapeo directo sobre los diagramas de interacción

Un objeto que va a desarrollar la misma operación independientemente del estado cuando un cierto estímulo es recibido es llamado "objeto controlado por estímulos". El orden en que los estímulos son recibidos en el objeto no es relevante ; siempre desarrolla la misma tarea si recibe el mismo estímulo. Esta es la forma en que los objetos normalmente actúan en un contexto orientado a objetos. Las operaciones pueden (en principio) ser ejecutadas en un orden opcional. Esto es algunas veces "aproximación de lista de compras"[41] ya que nos recuerda a una lista de compras: el orden en que se recogen los artículos no importa, lo que importa es que todos los artículos sean recogidos. Los objetos que implementan los objetos de entidad son usualmente "controlados por estímulos".

Los objetos que seleccionan operaciones no sólo por el estímulo recibido sino también por el estado actual son llamados "objetos controlados por estados". Esto muestra una fuerte relación temporal entre el estado actual del objeto y el estímulo que puede recibir; por ejemplo, una operación puede sólo ser ejecutada en estados específicos. El "stack" p.e., es "controlado por estados".

Los objetos que implementan objetos de control muchas veces tienen la tendencia de ser "controlados por estados" , esto no significa que los "objetos controlados por estados" sean independientes de los estímulos recibidos, pero hay una gran relación entre el estimulo que se va a recibir y el estímulo que se recibió previamente y en que orden se hayan recibido.

Para objetos controlados por estímulos y para objetos simples controlados por estados no es necesario dibujar un gráfico de transición de estados ya que sería de poco valor debido a su simplicidad. Los beneficios de esos gráficos se incrementan cuando el comportamiento es más

[41] Meyer B., OBJECT ORIENTED SOFTWARE CONSTRUCTION, Englewood Cliffs, N.J. Prentice Hall, 1998, citado en Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998, pág. 183

complicado. Especialmente en comportamiento donde los estados son importantes, estos gráficos serán de gran ayuda.

Cuando hemos identificado esas propiedades del bloque, podemos hacer un bosquejo de la estructura interna del bloque. La estructura interna de un bloque va a consistir de modelos del objeto como se había visto previamente. Si nuestro lenguaje de implantación es orientado a objetos, estos modelos de objetos van a llegar a ser clases en el lenguaje; si no tenemos clases naturalmente en el lenguaje entonces serán unidades de módulo en el código fuente.

Los bloques usados en el modelo de diseño son actualmente mecanismos de abstracción para estos módulos de objeto.

Cuando es hecha la estructura de un bloque, definimos las clases en el sistema. Aquí es de interés, por supuesto, el diseñar clases reutilizables y de alta calidad. De la misma manera como hemos hablado acerca de homogeneización entre estímulos también queremos que las clases sean homogeneizadas. Esto significa que no queremos dos clases ofreciendo funcionalidad similar, a no ser que estén relacionadas a través de la herencia. Adicionalmente queremos que ofrezcan fuerte acoplamiento internamente. Por ejemplo, si la mitad de las operaciones accesan la mitad de las variables de instancia y las otras operaciones operan sobre las otras variables, debemos considerar dividirla en dos clases. Otras heurísticas del diseño de buenas clases incluyen un juicio del valor potencial de reusabilidad de la clase. Sin embargo, toma tiempo diseñar una buena clase, y no siempre es eficiente el desarrollar cada clase en el sistema para que sea lo más general posible. Una estimación común es que toma de cinco a diez veces más el diseñar una clase componente que una clase ordinaria.

La forma más conveniente de implementar una interfaz del bloque es tener una clase implementando la interfaz entera. Esto es muchas veces una solución apropiada. Sin embargo, habrá muchas veces cuando es mejor dividir la interfaz del objeto para que sea implementada por dos o más clases.

Algunas veces la funcionalidad de los bloques no va a necesitar ser implementada como una clase e instancia de una clase. La razón en este caso es que no necesita ninguna variable de instancia. Una colección de operaciones pueden ser suficientes . Esto en algunos lenguajes es posible pero en algunos otros lenguajes orientado a objetos no, en este caso las operaciones se deben de poner en una clase

Es importante utilizar componentes en la estructuración de las entrañas de un bloque. Algunos componentes deben de ser adaptados antes de que puedan ser usados y por lo tanto puedan dar origen a nuevas clases que actúan como cascarón del componente.

3.3.2.2. MODELO DE IMPLANTACIÓN

La base para la implantación es el modelo de diseño. Aquí se ha especificado la interfaz de cada bloque y se ha descrito también el comportamiento que se está esperando detrás de la interfaz.

Lo más común es que se necesitan seres humanos para hacer esta transición final a código fuente.

La adaptación del lenguaje de programación va a ser hecha de acuerdo a la especialización del proceso de construcción del lenguaje seleccionado. La adaptación de lenguajes de construcción no se hace antes de la fase de construcción, se ha visto durante el proceso entero de diseño, estamos forzados a considerar cómo resolver problemas que surgen debido al ambiente de implantación en cuestión.

La especialización del lenguaje describe como traduciremos los términos utilizados en el diseño en términos y propiedades en el lenguaje de implantación. Todos los lenguajes van a tener sus especialidades durante la implantación. En esta discusión vamos a considerar lenguajes que sean orientados a objetos.

Muchos proyectos y organizaciones tienen reglas de código acerca de cómo debe de escribirse el código fuente. El propósito de estas reglas es obtener un código fuente uniforme que facilite el entendimiento y lectura .

En los lenguajes que ofrecen clases y herencia, estas clases van a corresponder a los módulos objetos que hemos diseñado.

Va a ser deseable agrupar estas clase de la misma forma que se agruparon anteriormente en bloques y subsistemas.

La herencia entre módulos objetos es directamente mapeada a herencia entre clases en el lenguaje de programación.

Los atributos que hayan sido identificados durante la fase de análisis que aparezcan durante el trabajo de diseño son usualmente implementados como una variable de un tipo específico.

Las asociaciones de conocimiento son normalmente implementadas en la misma manera que los atributos, llamándolas como referencias ordinarias de instancia o punteros de variables de instancia. Lo mismo se puede aplicar para las asociaciones de comunicación, pero esto podría ser

representado por variables ordinarias locales en una operación o por nada si un valor de retorno es usado como una referencia a un objeto. Por estas asociaciones queremos también un estímulo manejado. Un estímulo normalmente llega a hacer mensajes o llamadas a rutinas ordinarias.

Hemos visto que los conceptos son directamente traducidos a propiedades en el lenguaje de programación. En la tabla 3.1 se ha resumido la traducción hacia C++.

ANÁLISIS	DISEÑO	CÓDIGO FUENTE C++
Objeto de análisis	Bloque	1..N clases
Comportamiento de un objeto	Operación	Función miembro
Atributo (de la clase)	Atributo (de la clase)	Variable estática
Atributo (de instancia)	Atributo (de instancia)	Variable de instancia
Asociación de conocimiento	Asociación de conocimiento	Variable de instancia
Asociación de comunicación	Asociación de comunicación	Referencia a una función miembro
Interacción entre objetos	Estímulos	Llamada a una función miembro
Escenario	Diseño de escenario	Secuencia de llamadas

Tabla 3.1 Rastreo de los conceptos OO hacia C++

Para bloques controlados por estados muchas veces necesitamos mantener un rastro del estado actual. Las bases son muchas veces una descripción del gráfico de transición de estados. Una solución es usar sentencias CASE o IF para checar el estado actual.

El instanciamiento es manejado en la manera ofrecida por el lenguaje. El instanciamiento es descrito normalmente en los diagramas de interacción.

El ocultamiento de datos es una propiedad importante utilizada en contextos orientados a objetos. Todos los atributos deben de ser privados a menos que haya razones especiales para no serlo.

Si nuestro lenguaje no es orientado a objetos debemos traducir los conceptos fundamentales, como herencia y encapsulación de otra manera.

La gran diferencia con los lenguajes de programación orientados a objetos es que todo está sobre el concepto de herencia. El problema más grande es cómo debe de ser traducida la herencia.

El esquema que hemos dado hasta aquí es que el proceso de construcción es recto y fácil de seguir. Sin embargo esto no es cierto. El desarrollo es un proceso incremental y deben de ser hechas muchas iteraciones antes de que todos los estímulos sean definidos y las interfaces puedan ser congeladas. Además cuando empieza la implantación, los cambios a las definiciones de estímulos van a ocurrir.

Podemos mencionar algunas acciones que se pueden tomar en cuenta para la implantación:

- Empezar la construcción temprano, preferible al mismo tiempo que se empieza a trabajar en el modelo de diseño. El primer paso en la construcción es identificar el ambiente de implantación y esto puede ser hecho en paralelo con el diseño, así se está listo cuando la construcción actual empieza.

- Seguramente se ha notado que el modelo de diseño es un refinamiento del modelo de análisis tomando en cuenta cierto ambiente de implantación. ¿Cuándo se debe de hacer la transición?. Si se ha hecho un modelo de análisis muy detallado, el grado de refinamiento va a ser muy pequeño, a menos, que el ambiente de implantación de problemas. Cuando se debe de hacer la transición debe de ser decidido en cada proyecto. Es importante decidir esto temprano, y la decisión debe de ser basada en el resultado de identificación del ambiente de implantación.

Las iteraciones pueden existir en diferentes niveles de granularidad. Pequeñas iteraciones pueden ocurrir cuando, después de la inspección, se decide cambiar algo en el diseño, por ejemplo, en que objeto será colocada una cierta funcionalidad. El más bajo nivel de iteraciones es probablemente un diseñador individual quien simplemente hace cambios en sus clases.

3.4. PRUEBAS

El modelo de pruebas es el último modelo desarrollado en el desarrollo de un sistema. Describe un estado simplemente, el resultado de las pruebas. Los conceptos fundamentales en las pruebas son "la especificación de las pruebas" y "el resultado de las pruebas".

La prueba del software representa una revisión final de las especificaciones del diseño y de la codificación. Por lo tanto se debe de planear de la misma manera que el análisis y la construcción.

"Los objetivos de la prueba son :

- *La prueba es un proceso de ejecución de un programa con la intención de descubrir un error.*
- *Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.*
- *Una prueba tiene éxito si descubre un error no detectado hasta entonces"*[42]

Con los objetivos anteriores podemos ver que la perspectiva cambia con respecto al éxito o fracaso de una prueba, sin embargo hay algo muy importante que no se debe de perder de vista que una prueba nunca va a poder demostrar que el programa no tiene errores, sólo puede demostrar que los tiene.

Probar un producto es relativamente independiente del método utilizado para desarrollarlo. Las actividades de pruebas se dividen normalmente en dos :

- Verificación : Checa que el resultado esté de acuerdo con la especificación, sin embargo esto no garantiza la satisfacción del cliente.
- Validación : Checa que el resultado sea lo que el ordenante quiere, se enfoca a la satisfacción del cliente.

Estos dos términos se resumen con dos preguntas: ¿Se está construyendo el sistema correctamente? (verificación), ¿Se está construyendo el sistema correcto? (validación).

Cuando empieza la prueba va a existir un conflicto de intereses, se pide a la gente que diseño el sistema que lo pruebe, esto es bueno aparentemente porque nadie conoce el sistema mejor que la persona que lo hizo, sin embargo por esta misma razón *"porque lo hizo"*, va a tratar de demostrar que el software funciona correctamente, que está libre de errores, que funciona de acuerdo a las especificaciones del cliente, por lo tanto se pueden estar haciendo pruebas que traten de demostrar que el sistema funciona correctamente y no tiene problemas, sin embargo los errores seguirán estando.

Podemos decir entonces que las pruebas realizadas por el creador del sistema no sean ciertas 100 %, sin embargo, el puede ser el único indicado y capas de realizar algunos tipos de pruebas.

Esto nos orilla a que se plante una opción bastante considerable, la introducción de un grupo de prueba. De esta forma el que desarrolla el sistema es responsable de probar las unidades

[42] Mycrs, g., THE ART OF SOFTWARE TESTING, Wiley, 1979, citado en Pressman S. Roger, INGENIERÍA DE SOFTWARE UN ENFOQUE PRÁCTICO, 3a edición, McGraw-Hill, España, 1993, pág. 624,625

individuales del sistema, garantizando que cada una hace la función que debe de hacer. También se encargará de la prueba de integración que cada subsistema se comunique correctamente con los otros. De esta forma el grupo de prueba sólo entra en acción cuando la estructura del sistema esté completa. De esta forma se evita el conflicto de intereses antes mencionado ya que en la prueba total de sistema no participa el desarrollador y al grupo de prueba lo que le interesa es encontrar errores.

Es muy importante que el desarrollador esté disponible durante las pruebas desarrolladas por el grupo de prueba para aclarar cualquier duda y corregir errores que se presenten.

Es bien conocido el fenómeno de que al corregir errores que se hayan detectado es posible que se introduzcan nuevos errores en el sistema. Por esto cuando se corrige un error se debe de volver a probar el software, cuando menos la(s) unidad(es) que funcionan como cliente de la unidad que se corrigió.

La estrategia para el desarrollo del software, se podría ver como una espiral en la cual al ir dando vueltas se va disminuyendo el nivel de abstracción , es decir se pasa desde la adquisición de requerimientos, análisis, diseño y codificación. *"Tomando en cuenta que el desarrollo del software se puede ver como un proceso "constructivo" y el proceso de probar el software como un proceso "destructivo"*"[43], podemos ver lo procesos de una forma contraria o encontrados, por esta razón podemos ver la estrategia de prueba del software moviéndonos hacia fuera de la espiral de tal forma que en cada vuelta se aumenta el alcance de la prueba.

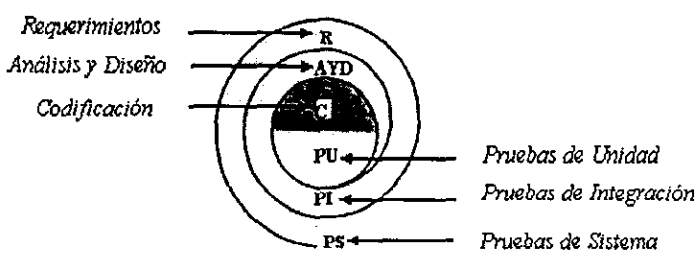


Figura 3.31. Estrategia de prueba

Inicialmente se prueba a bajos niveles es decir bloques y módulos (*prueba de unidad*), estos son probados por los diseñadores, después son probados subsistemas de bajo niveles. De esta forma se llega a la *prueba de integración*, que no va a ser un gran evento sino que se va formando conforme se van integrando más y más partes. Una herramienta para la prueba de integración es usar el modelo de escenarios para integrar un escenario a la vez. Así se llega a la *prueba completa* del sistema donde se prueba como un todo el sistema.

[43] Pressman S. Roger, INGENIERÍA DE SOFTWARE UN ENFOQUE PRÁCTICO, 3a edición, McGraw-Hill, España, 1993, pág. 624

De esta forma la prueba del software es entonces llevada a cabo empezando a probar niveles más bajos, posteriormente los escenarios y finalmente el sistema entero. Para hacer las pruebas es recomendable apoyarse en los modelos que proporcionan la estructura del sistema. El modelo de requerimientos es útil cuando probamos cada escenario, checamos que cada uno de los objetos se comunique correctamente en ese escenario particularmente. Similarmente, checamos las interfaces de usuario descritas en el modelo de requerimientos entonces vemos que el modelo de requerimientos es verificado por el proceso de pruebas

El diseño de las pruebas para un software requiere tanto esfuerzo como el análisis y el diseño del producto, sin embargo por lo general se trata como algo de menor importancia, además de que por razones que ya hemos discutido antes desarrollan casos de pruebas en donde "*se sienten bien*", pero que no siempre son completos.

A cualquier software se le pueden aplicar dos pruebas:

- De caja negra: conociendo la función específica para la que fue diseñado el producto, se pueden llevar a cabo pruebas que demuestren que cada función es completamente operativa.

- De caja blanca: conociendo el funcionamiento del producto, se pueden desarrollar pruebas que aseguren que todas las piezas encajan, o sea, que la operación interna se ajusta a las especificaciones y que todos los componentes se han comprobado de forma adecuada.

Las pruebas de caja negra son diseñadas para validar los requisitos funcionales sin fijarse en el funcionamiento interno de un programa, es decir, valida que se acepta de forma adecuada la entrada y se produce una salida correcta. Por otro lado las pruebas de caja blanca se centran en la estructura de control del programa, aquí se derivan casos de prueba que aseguren que durante la prueba se han ejecutado por lo menos una vez todas las sentencias del programa y que se ejercita todas las condiciones lógicas. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coincide con el esperado o afirmado.

Hetzel[44] denomina a la prueba de caja blanca como prueba a pequeña escala, esto se debe a que las pruebas de caja blanca se deben de aplicar a pequeños componentes de programas, la prueba de caja negra la denomina como prueba a gran escala.

PRUEBAS DE UNIDAD

Esta prueba centra el proceso de verificación de la unidad más pequeña del diseño del software (módulo). Se deben de probar los flujos principales y alternativos que se siguen en el módulo. La prueba de unidad siempre está orientada a la caja blanca.

[44] Hetzel, W., THE COMPLETE GUIDE TO SOFTWARE TESTING, QED Information Sciences, 1984. Citado en Pressman S. Roger, INGENIERÍA DE SOFTWARE UN ENFOQUE PRÁCTICO, 3a edición, McGraw-Hill, España, 1993, pág. 657

En la prueba de unidad se divide en varias partes que se muestran en la figura siguiente.

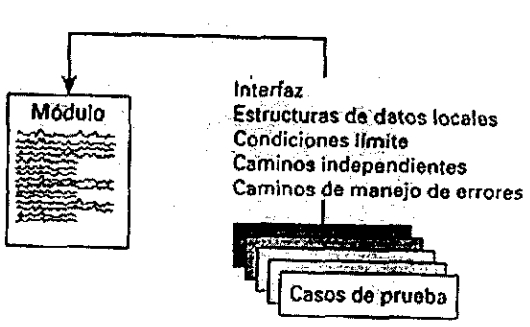


Figura 3.32. Prueba de unidad

Se prueba la interfaz para asegurar que la información fluye correctamente desde y hacia el módulo que se está probando. Las estructuras de datos se prueban para asegurarnos de que conserven su integridad durante los pasos de ejecución del algoritmo del módulo. También se deben de probar las condiciones límite para asegurar que el módulo funciona correctamente en los límites establecidos como restricciones del procesamiento. Se prueban los caminos principales para asegurarnos que todas las sentencias se ejecutan al menos una vez y posteriormente los caminos alternativos y manejo de errores.

Esta prueba se deben diseñar casos de prueba para detectar errores en cálculos incorrectos, comparaciones incorrectas o flujos de control incorrectos. Esta prueba es muy común que no se realice estrictamente mediante un plan sino a lo largo de la implementación de los módulos con algunas facilidades que presentan los lenguajes de programación como los "debuggers".

PRUEBAS DE INTEGRACIÓN

Es muy válido el que alguien se cuestione el que si los módulos funcionan independientemente, ¿Cómo estar seguro de que funcionan en conjunto?. La prueba de integración es una técnica para construir la estructura del programa, mientras al mismo tiempo se llevan a cabo pruebas para detectar errores asociados con la interacción. El objetivo va a ser considerar módulos probados en unidad y construir un estructura que esté de acuerdo con lo que dicta el diseño del programa.

Existen dos tipos de integración: incremental y no incremental.

- Integración no incremental: Trata de construir el programa con un enfoque de "big bang". Aquí se combinan todos los módulos por anticipado y se prueba todo el sistema en conjunto. Normalmente se encuentra un gran número de errores y la corrección se hace difícil debido a

que no es fácil aislar las causas de los errores al tener el programa completo. Una vez corregidos los errores se vuelve a probar y el proceso continúa hasta que no hay errores.

- Integración incremental: Es lo contrario a la integración no incremental. El programa se crea y se prueba en pequeñas partes en las que los errores son más fáciles de aislar y de corregir. Dentro de la integración incremental existen dos estrategias : integración descendente e integración ascendente.

La integración descendente es un planteamiento incremental. Se integran los módulos moviéndose hacia abajo por la jerarquía de control, comenzando con el módulo de control principal . Los módulos subordinados se van integrando a la estructura bien de forma *primero en profundidad* o *primero en anchura.* La integración primero en profundidad integra todos los módulos de un camino de control principal de la estructura, la selección del camino es arbitraria. La integración primero en anchura incorpora todos los módulos directamente subordinados a cada nivel, moviéndose por la estructura en forma horizontal.

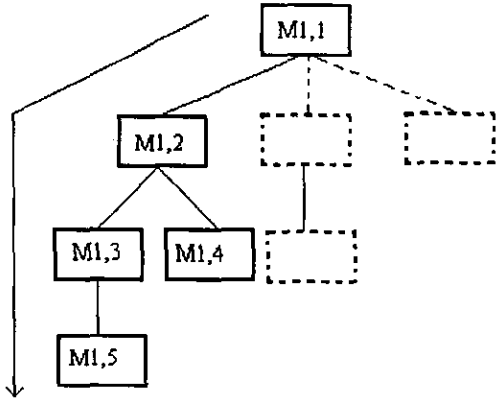


Figura 3.33. Integración descendente

La integración ascendente empieza con la construcción y prueba de los módulos atómicos, dado que los módulos se integran de abajo hacia arriba, el comportamiento requerido por los módulos subordinados siempre está disponible.

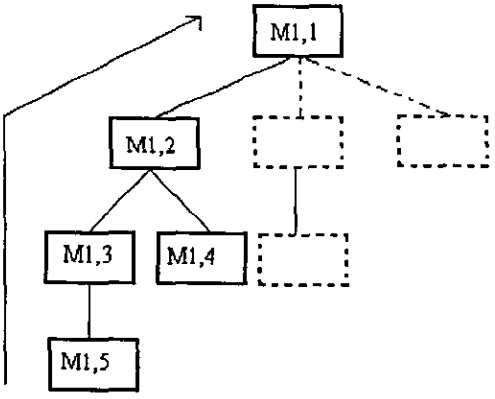


Figura 3.34. Integración ascendente

Hay ventajas y desventajas de la prueba de integración ascendente frente a la descendente. Una desventaja de la integración ascendente es que el programa como entidad no existe hasta que se ha añadido el último módulo, sin embargo esta estrategia nos permite mayor facilidad de casos de prueba. La ventaja de la integración descendente es que se pueden probar de antemano las principales funciones de control del programa. Una recomendación es utilizar las dos estrategias la descendente para niveles superiores de la estructura del programa y la ascendente para niveles inferiores.

PRUEBAS DEL SISTEMA

El software puede ser un elemento de un sistema mayor basado en computadora. Por lo tanto la prueba del sistema , realmente, está constituida por una serie de pruebas, aunque cada una tiene propósito distinto en conjunto trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que se realizan las funciones apropiadas.

Pruebas De Recuperación

Muchos sistemas basados en computadora deben de recuperarse de los fallos y resumir el procesamiento en un período de tiempo establecido. Esta prueba fuerza el fallo del sistema de varias formas y verifica que la recuperación se lleve acabo apropiadamente. Si la recuperación es

automática hay que evaluar los mecanismos de recuperación del estado del sistema, de la recuperación de datos y de arranque. Si la recuperación requiere la intervención humana hay que evaluar los tiempos medios de reparación para determinar si están dentro de unos límites aceptables.

Pruebas De Seguridad

Cualquier sistema de computadora que maneje información sensible o lleve a cabo acciones que puedan perjudicar o beneficiar a los individuos es objeto de penetraciones ilegales. Esta prueba trata de verificar que los mecanismos de seguridad implantados en el sistema funcionarán correctamente impidiendo penetraciones ilegales, es decir protegerá al sistema. En esta prueba se intentará entrar al sistema por cualquier medio, lo que el desarrollador deberá de hacer es que el costo de entrar sea mayor que el valor de la información.

Prueba De Resistencia

Las pruebas de caja blanca y caja negra nos proporcionan resultados de la evaluación de los sistemas en situaciones normales, las pruebas de resistencia están diseñadas para enfrentar a los programas a situaciones anormales. Esta prueba ejecuta un sistema de forma que demande recursos en cantidad, frecuencia o volúmenes anormales.

Prueba De Sensibilidad

Es una variación de la prueba de resistencia, en algunas situaciones principalmente en algoritmos matemáticos, un rango muy pequeño de datos dentro de los límites de una entrada válida para un programa puede producir un procesamiento extremo e incluso erróneo o una profunda degradación del rendimiento.

Prueba De Rendimiento

Es inaceptable que sistemas en tiempo real no proporcionen los resultados que se ajusten a los requisitos de rendimiento aunque los resultados sean correctos. La prueba de rendimiento debe diseñarse para probar el rendimiento del software en tiempo de ejecución dentro del contexto de un sistema integrado.

4. CONSIDERACIONES PARA ELEGIR UN LENGUAJE ORIENTADO A OBJETOS

Han surgido lenguajes nuevos de programación, muchos de los cuales resultan muy atractivos, sin embargo, muchas ocasiones es preferible escoger un lenguaje "antiguo" que se encuentre bien documentado, con el que se tenga familiaridad y se haya utilizado exitosamente antes. Sin embargo se deben de evaluar a conciencia los lenguajes "nuevos" ya que se debe de propiciar siempre una transición entre los viejos y los nuevos.

Aunque los lenguajes de programación proporcionan los medios de traducción hombre - máquina, es un hecho que el resultado depende más directamente de las actividades que preceden y siguen a la codificación.

Después de establecidos los requisitos del software se hacen más importantes las características técnicas de los lenguajes de programación. Por ejemplo si se requieren aplicaciones en tiempo real, si se requieren estructuras de datos complejas, o eficiencia en manejo de memoria etc. En teoría los lenguajes de programación se deberían de ajustar al sistema o al procesamiento que se va a realizar, pero como muchas otras veces, en la práctica muchas veces se selecciona un lenguaje porque es el único que se tiene, o es en el que se debe de realizar obligadamente.

Podemos decir que en teoría, la creación de software orientado a objetos se podría llevar a cabo con cualquier lenguaje, pero la realidad es que el soporte para métodos orientados a objetos debe de existir internamente en el lenguaje a utilizar. Se mencionaron los conceptos fundamentales de la orientación a objetos en el capítulo tercero.

El concepto más importante que un lenguaje orientado a objetos debe de soportar es el concepto de objeto. De esta forma el lenguaje debe de soportar la definición de un conjunto de operaciones para el objeto, llamémosle interfaz del objeto, y una implantación parte del objeto. La implantación del objeto es entonces encapsulada y oculta para el usuario. Debe también soportar la definición de clases, el polimorfismo, le herencia, el paso de mensajes.

Dada la diversidad de lenguajes de orientación a objetos los programadores son confundidos muchas veces acerca de cómo seleccionar un lenguaje para un proyecto particular. Aquí se dan algunos puntos importantes a considerar además de lo ya mencionado.

FAMILIARIDAD. Si muchos programadores de los que van a realizar un proyecto están familiarizados con un cierto lenguaje, entonces es razonable explotar esa familiaridad continuando con el uso de ese lenguaje. Es muy importante no tomar mucho tiempo para familiarizarse especialmente para lenguajes que no son orientados a objetos. Además de que la familiaridad de un programador con un lenguaje puede hacer que el programador pierda los fundamentos para programar en orientación a objetos.

PLATAFORMAS. No todos los lenguajes orientados a objetos corren en todas las plataformas. Se debe de tomar en cuenta entonces en que plataforma se va a correr la aplicación final.

IMPORTANCIA DEL TIEMPO DE PROGRAMACION RELATIVO AL TIEMPO DE EJECUCIÓN. Existen lenguajes que son diseñados para tener un eficiente tiempo de ejecución y otros que hacen énfasis en el fácil desarrollo del programa muchas veces sacrificando el tiempo de ejecución.

PRESENCIA. Un lenguaje popular, es común que tenga una diversa colección de tutoriales, guías de entrenamiento, herramientas y material de lectura que proporciono un mejor conocimiento del lenguaje.

ACCESO A CODIGO EXISTENTE. Este acceso es útil no sólo por pedagogía (muchas veces es la mejor manera de aprender como programar en un nuevo estilo, es estudiar el código existente), además por reusabilidad. Si una fracción larga de código de una aplicación existente que pueda ser utilizada posteriormente puede ser construida fuera de las librerías disponibles en el lenguaje, entonces al realizar otra aplicación se reducirá el tiempo ya que puede ser reutilizada.

AMBIENTE DE PROGRAMACION. La disponibilidad de editores, "debuggers" (depuradores), y otras herramientas de programación simplifica la creación de nuevas aplicaciones.

5. CONSIDERACIONES PARA EL DESARROLLO DE LA INTERFAZ DE USUARIO.

La interfaz es la envoltura de un software. Hay interfaces que son confusas, complicadas de utilizar y difíciles de aprender, sin embargo la gente que las diseñó no crea estos problemas intencionalmente, pero estos son una parte fundamental para la aceptación de cualquier software. Debido a que el uso de computadoras ha ido aumentando considerablemente, se le ha dado mayor importancia al diseño de la interfaz, ya que si esta es simple y fácil de aprender el usuario podrá hacer un buen uso de lo que hay adentro, sino, aparecerán invariablemente problemas.

El diseño del software es muy importante para la calidad global del software, pero este está en gran parte oculto al usuario final, sin embargo, el diseño de la interfaz es diferente ya que el usuario siempre va a estar en contacto con ella, y de eso depende que el usuario se sienta a gusto con el modo de interactuar desarrollando así, un ritmo normal de interacción.

La interfaz, entonces va a ser el medio de comunicación entre el hombre y la máquina, por esta razón cuando se diseña una interfaz es necesario tomar en cuenta factores humanos ya que los usuarios son personas. Por factores humanos vamos a entender cosas como: percepción sensorial y psicológica, memoria, razonamiento inductivo (general – particular), deductivo (particular - general), el comportamiento del usuario hacia el que va dirigido, el ambiente donde se utilizará.

A través de los sentidos es como el ser humano percibe el mundo, por lo tanto cuando se considera una interfaz hombre – máquina predominan los sistemas visuales auditivos y táctiles. Esto permite al usuario recibir la información y almacenarla. Además de los sentidos es necesario tomar en cuenta que entre los usuarios van a existir diferencias de habilidad, personalidad, comportamiento y experiencia. De este modo una interfaz aceptable para un ingeniero puede no ser la adecuada para un abogado, etc. De esto podemos decir que la habilidad que tenga el usuario del sistema, repercutirá en la habilidad que tenga para obtener información de la interfaz.

La habilidad de un usuario no está dada principalmente por la cultura o inteligencia, sino por el conocimiento del contexto de algún problema. La personalidad de los usuarios finales está ligada a un estilo cognitivo y cada usuario tiene personalidad diferente por lo tanto la interfaz "ideal" debería adaptarse a las diferencias de personalidad, o a una personalidad común de entre los usuarios finales.

El diseño de la interfaz es algo extra o algo además del diseño del software, ya que en éste, solamente se llega al nivel de conocer los datos que se deben de introducir o de mostrar a través de la interfaz, más nunca el diseño de la misma.

Un aspecto importante en el diseño de la interfaz es el análisis y comprensión de las tareas que se realizan manualmente o de la forma convencional y transformarlas en un conjunto parecido, pero en el contexto de una interfaz de computadora.

EVOLUCIÓN DE ESTILOS DE INTERACCIÓN

La evolución de las computadoras ha estado ligada a los estilos de interacción hombre – máquina y a las tendencias de las interfaces, aunque muchos sistemas modernos siguen utilizando estilos de interacción anteriores.

En un principio la única forma de interacción hombre – máquina era a través de la *"interfaz de preguntas y órdenes"*, la interacción era textual, como en este ejemplo:

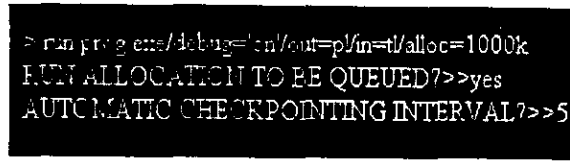


Figura 5.1. Interfaz de preguntas y órdenes

Aun si estas preguntas eran concisas, eran propensas a errores y bastante difíciles de aprender.

Una mejora a éste tipo de interfaz fue la *"interfaz de menú simple"*, en esta se presenta una lista de opciones y la selección se hace con la introducción de un código, se muestra un ejemplo en la siguiente figura. El menú nos ofrece un contexto global, y es menos propenso a errores, pero dependiendo del nivel de profundidad que tenga, es decir, los niveles a los que pueda llegar, puede ser tedioso encontrar la opción deseada.

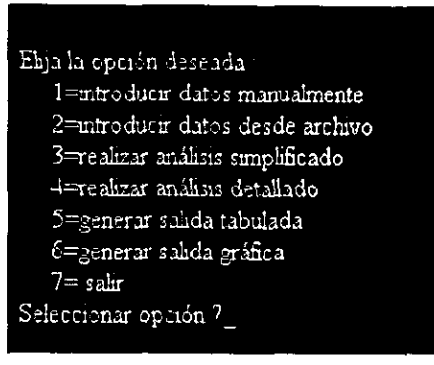


Figura 5.2. Interfaz de menú simple

La siguiente evolución, fue debido al mayor aprendizaje de los factores humanos y se crearon las *"interfaces orientadas a ventanas con opción de señalar y elegir"*, este tipo de interfaces nos proporcionan algunas ventajas como:

- Se pueden visualizar diferentes tipos información simultáneamente permitiendo cambiar de contexto.

- El esquema de menú desplegable permite realizar varias tareas diferentes interactivamente.

- La utilización de iconos gráficos, menús desplegables, y botones reducen el número de pulsaciones en el teclado.

La interfaz de mayor actual reúne los atributos de la interfaz orientada a ventanas con el hipertexto y la multitarea que es la habilidad para realizar diferentes tares al mismo tiempo.

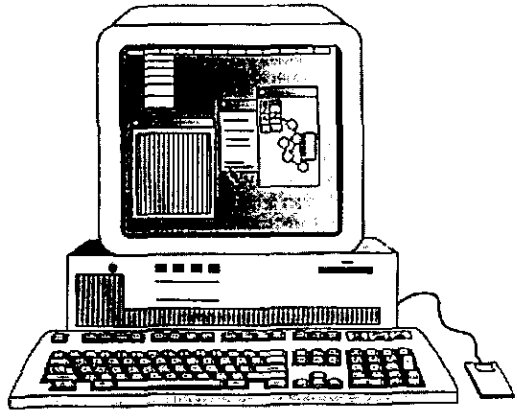


Figura 5.3. Interfaces orientadas a ventanas con opción de señalar y elegir

Los dos últimos tipos de interfaces son los más amigables y accesibles siempre y cuando se haga un análisis cuidadoso de ellos. Se han descrito algunos tipos de interfaces comunes, esto no quiere decir que sean los únicos, sin embargo si son los más conocidos por la mayoría de los usuarios de computadoras.

CICLO DEL DESARROLLO DE LA INTERFAZ

El proceso de desarrollo de la interfaz es iterativo, empieza con una descripción de las características de la interfaz por escrito, desarrollada por los usuarios, se realiza posteriormente un prototipo de dichas características, posteriormente se le da a revisar y/o evaluar al usuario para ver si cumple sus necesidades. Este hará sus comentarios y/o correcciones, se le presenta nuevamente, y así de forma cíclica hasta que quedan satisfechas las necesidades. Se muestra un esquema del ciclo del desarrollo de la interfaz a continuación.

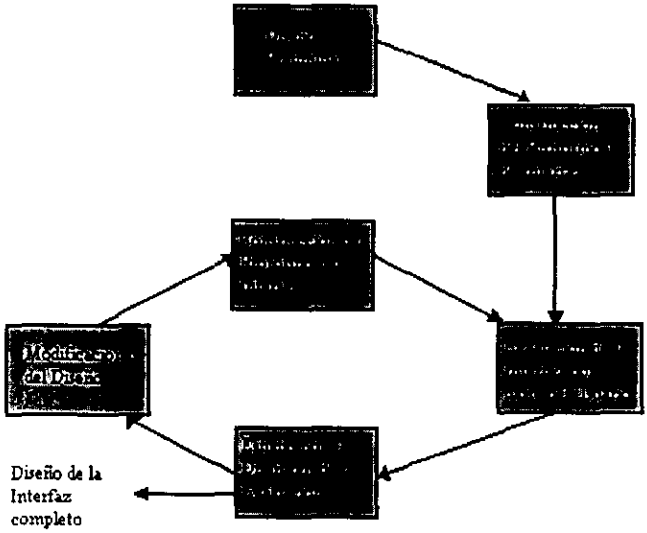


Figura 5.4. Ciclo del desarrollo de la interfaz

Las técnicas que pueden utilizar para la evaluación[45] de la interfaz varían desde las pruebas directas de la interfaz hasta estudios estadísticos.

ASPECTOS DE DISEÑO DE UNA INTERFAZ

Para el diseño de una interfaz deben de ser considerados algunos aspectos como tiempo de respuesta, facilidad de ayuda al usuario, manejo de errores y asignación de nombres a las

[45] Moran, T.P., THE COMMAND LANGUAGE GRAMMAR: A REPRESENTATION FOR THE USER INTERFACE OF INTERACTIVE COMPUTER SYSTEMS, International Journal of Man – Machine studies, vol. 15, pp. 3 – 50, citado por Pressman S. Roger, INGENIERÍA DE SOFTWARE UN ENFOQUE PRÁCTICO, 3a edición, McGraw-Hill, España, 1993, pág. 494

órdenes. Es muy importante tomar estos aspectos desde el principio del diseño ya que si aparecen en etapas posteriores podría ser más costosa su solución.

El tiempo de respuesta es uno de los aspectos más importantes ya que todos buscamos que los sistemas hagan lo que tienen que hacer, de forma rápida. El tiempo de respuesta se considera desde que se realiza una acción de control por parte del usuario hasta que se obtiene la respuesta a esa acción de control por parte de la computadora. Es muy común pensar que se quiere la respuesta lo más pronto posible, sin embargo esto no siempre es lo mejor.

Existen dos características en el tiempo de respuesta "variabilidad" y "retardo". La variabilidad es una característica muy importante ya que se refiere a la desviación del tiempo de respuesta medio. Una variabilidad muy baja nos va a permitir establecer un ritmo adecuado de interacción, incluso si el retardo es muy grande.

El retardo se refiere a que tanto se tarda en responder el sistema, si es muy grande puede ocasionar desesperación en los usuarios y si es muy corto puede inducir al usuario a ir deprisa y cometer errores.

Con lo que respecta a la ayuda se sabe que todos los usuarios de los sistemas de computadora necesitan ayuda algunas veces. Existen en esencia dos tipos de ayuda; ayuda integrada y ayuda añadida. La ayuda integrada se diseña y se encuentra dentro del software, obviamente esto va a reducir el tiempo que se requiere para obtener la ayuda (si es que está bien estructurada), además de que hace la interfaz más amigable. La ayuda añadida es la que se integra al sistema después de construido, son generalmente los manuales del sistema, una desventaja es que cuando el sistema es muy complejo se entrega mucha información.

Los mensajes de error son una característica que se debe de tomar en cuenta ya que su impacto sobre el usuario puede ser determinante. Muchos de los mensajes son confusos o sin utilidad. Los mensajes de error dentro de sus características deben indicar:

- Qué ha salido mal, en un lenguaje que entienda el usuario.
- Debe de ir acompañado de una señal audible o visible, es posible combinar las dos.
- No debe culpar al usuario.
- El mensaje debe de indicar las consecuencias del error.
- Si no puede ser muy claro, debe indicar donde se puede ampliar la información.

El diseño de las interfaces de usuario la mayoría de las veces se realiza en base a la experiencia del desarrollador ya que por lo general no hay tiempo para realizar una investigación documental, para obtener las directivas del diseño de la interfaz. Aquí se mencionan de forma rápida algunas directivas a considerar para el buen diseño de la interfaz, además de los aspectos ya mencionados:

- Ser consistente en todo momento, selección de menús entrada de órdenes, visualización de datos, etc.

- Ofrecer una realimentación significativa, es decir, asegurar una comunicación bidireccional entre el usuario y la máquina.

- También hay que preguntar la verificación de cualquier acción destructiva no trivial.

- Reducir la cantidad de información a ser memorizada para el desarrollo de las acciones.

- El sistema debe de protegerse de los errores del usuario que puedan causarle daño.

- Minimizar el número de pulsaciones del teclado.

- Categorizar las actividades en base a su función y organizar la geografía de la pantalla convenientemente.

- Utilizar verbos de acción simples o frases verbales cortas para nombrar las órdenes.

- Es muy importante mostrar sólo la información que sea relevante al contexto actual, así como mostrar un formato de presentación que permita una rápida asimilación de la información mostrada.

- Si es posible las gráficas y esquemas deben de reemplazar a los resultados tabulares.

- Producir mensajes de error significativos.

- La mayoría de la información presentada en una interfaz es texto por lo tanto es muy importante utilizar todo lo que esté a nuestro alcance para que sea fácilmente asimilable, como: usar mayúsculas y minúsculas, tabulaciones y agrupaciones.

- Utilizar, si es posible, ventanas para modularizar la información. De esta forma se tiene información distinta accesible al usuario de forma comprensible.

- Si es posible, utilizar representaciones análogas para mostrar información, esto es algo que tenga mayor impacto sobre el usuario y que sea más claro.

- Buscar mantener la consistencia de la información visualizada a lo largo del campo de adquisición de datos, o sea las pantallas de captura.

- La interacción debe ser flexible en el sentido de que hay usuarios que prefieren usar el teclado al "mouse", y se les debe de permitir.

- Es recomendable desactivar las órdenes que no son apropiadas al contexto con la finalidad de evitar su ejecución.

- Se deben de evitar las entradas innecesarias de datos, utilizar valores por "defoult", y nunca obligar a introducir información que pueda ser calculada automáticamente dentro de la programación.

6. DESARROLLO DE LA APLICACIÓN

6.1. DESCRIPCIÓN DEL PROBLEMA: REQUERIMIENTOS

El problema a resolver es automatizar un proceso de análisis estadístico existente que se utiliza para la valoración de pruebas de capacidad física (este se describe adelante), originalmente lo que se debe de hacer es aplicar algunas pruebas de capacidad física a un grupo de personas que participan en algún deporte o que se quieran evaluar, posteriormente sobre los resultados se realiza un análisis estadístico de tal forma que se pueda obtener una evaluación sobre los resultados obtenidos, una vez que se obtienen los resultados se tiene que trabajar para mejorar las deficiencias encontradas. Si tomamos en cuenta que la gente que sabe de aplicar las pruebas de capacidad física son aveces profesores de educación física o entrenadores algunos preparados y otros empíricos podemos concluir que no necesariamente tienen conocimientos en estadística de tal forma que pudieran hacer este análisis, por lo que muchas veces queda inconcluso el proceso antes descrito, aún sí alguien tuviera los conocimientos les llevaría mucho tiempo el realizar el análisis debido a la cantidad de personas que tiene una agrupación.

Por estas razones al automatizar el proceso se les daría acceso a las gentes que estuvieran interesadas en concluir el proceso de evaluación, además de que se reduciría el tiempo en un porcentaje muy grande, y que se pueden por medio de la computadora dar algunas facilidades como mayor presentación en los resultados para que se puedan obtener y manipular gráficas de ellos sin dejar de presentarlos de forma tabular, posibilidad de consultas sobre datos personales y resultados de pruebas, regeneración de resultados de alguna prueba en cualquier momento, distintos niveles de abstracción para realizar alguna evaluación, almacenamiento de información evitando almacenamiento de papel.

El análisis estadístico que se quiere automatizar es el siguiente :

"PARA EXPLICAR EL PROCEDIMIENTO LARGO, RECURRIREMOS A LOS MISMOS DATOS QUE SE MANEJARON EN EL EJEMPLO ANTERIOR, CONSIDERANDO QUE SON EL RESULTADO DE UNA MEDICIÓN DE LA PRUEBA DE ABDOMINALES QUE, EN UN MINUTO REALIZARON CADA UNO DE LOS JUGADORES Y SON ESTOS:

62 61 60 60 60
59 58 57 57 56
56 56 55 55 54
54 53 51 50 50
49 48 48 47 46
46 45 44 43 43
42 42 42 41 41
40 40 39 39 39
39 38 38 37 37
36 36 35 33 32

EL SIGUIENTE PASO ES SABER COMO ACOMODAREMOS ESOS FACTORES, PARA ELLO SE HACE LA CONCENTRACION DE LOS NÚMEROS 0 INTERVALOS, UTILIZANDO LA SIGUIENTE REGLA ESTADÍSTICA:

(CM -cm) / 11 ó 13 = INTERVALO.

(62 - 32) / 11 = 30 / 11 = 2.7

(62 - 32)/ 13 = 30 / 13 = 2.3

POR LO TANTO SI ELEGIMOS DIVIDIR LA FORMULA ENTRE 11 TENDREMOS INTERVALOS DE TRES ESPACIOS, SI ELEGIMOS DIVIDIR ENTRE 13, TENDREMOS INTERVALOS DE DOS ESPACIOS.

ANTES DE INICIAR PROPIAMENTE EL PROCEDIMIENTO DEBEMOS CONOCER ALGUNAS LETRAS Y ABREVIATURAS EN CUANTO A SU SIGNIFICADO Y QUE UTILIZAREMOS A LO LARGO DEL PROCESO:

N = ES EL NÚMERO DE CASOS A MANEJAR.
CM = ES EL CÓMPUTO MAYOR.
cm = ES EL CÓMPUTO MENOR.
OSC = ES LA OSCILACIÓN 0 DIFERENCIA ENTRE EL CM Y EL cm.
INT = ES EL INTERVALO.
SFD = ES LA SUMA DE FRECUENCIAS POR DESVIACIONES.
SFD2= ES LA SUMA DE FRECUENCIAS POR DESVIACIONES ALCUADRADO.
C = ES EL CORRECTIVO.
C2= ES EL CORRECTIVO AL CUADRADO.

Ms = ES LA MEDIA SUPUESTA.
M = ES LA MEDIA.
S = ES LA SIGMA.
1/2 S = ES LA MEDIA SIGMA.

AHORA BIEN, PARA INICIAR EL DESARROLLO DEL PROCEDIMIENTO EMPEZAREMOS POR LOCALIZAR LOS DATOS Y FÓRMULAS SIGUIENTES:

TENIENDO EL DATO ANTERIOR NOS DEDICAREMOS AHORA A ELABORAR LA PRIMERA COLUMNA DEL CUADRO; ORDENANDO LOS DATOS DE ACUERDO AL NÚMERO DE INTERVALOS ESCOGIDO, EN ESTE CASO SERÁ CON EL RESULTADO DE LA DIVISIÓN DE:

$$\frac{CM - cm}{13} = 3:$$

INTERVALOS
60 - 62
57 - 59
54 -56
51-53
48 - 50
45 -47
42 - 44
39 -41

HABIENDO ELABORADO LA PRIMERA COLUMNA, NOS DAREMOS A LA TAREA DE DESPEJAR ALGUNOS DATOS Y FÓRMULAS QUE NOS PIDEN AL PRINCIPIO:

N = ESTO ES IGUAL A 50 PUESTO QUE SON LOS RESULTADOS QUE TENEMOS, AL CONTAR LOS FACTORES ORDENADOS AL INICIO DEL PROCESO.
CM = EL CONPUTO MAYOR DE ESOS FACTOR.ES ES 62, VERIFICAR EN EL MISMO ORDENAIMENTO DE DATOS.
cm = ES 32,VERIFICAR EN EL ORDENAMIENTO.
OSC = ESTE DATO SE SACA DE LA RESTA ENTRE EL CM Y EL cm, ES DECIR 62 MENOS 32 ES IGUAL A 30.
INT = ES 3, VER DATOS ANTERIORES.

POR LO TANTO QUEDARÍA DE LA SIGUIENTE FORMA LOS DATOS INICIALES:

CM=62
cm = 32
OSC 30
INT 3

SE DARÁN CUENTA QUE HASTA AQUI, NO HAY DIFICULTAD PARA LLEVAR A CABO ESTE PROCEDIMIENTO. AHORA CONSTRUIREMOS LA COLUMNA QUE SE DENOMINARÁ DE FRECUENCIAS Y ESTO QUIERE DECIR, QUE PONDREMOS FRENTE A CADA INTERVALO EL NÚMERO DE VECES O FRECUENCIAS QUE SE ENCUENTRAN EN ESE ESPACIO.

DATOS ORDENADOS	INTERVALOS	F
62 61 60 60 60	60 - 62	5
59 58 57 57 56	57 - 59	4
56 56 55 55 54	54 - 56	7
54 53 51 50 50	51 - 53	2
49 48 48 47 46	48 - 50	5
46 45 44 43 43	45 - 47	4
42 42 42 41 41	42 - 44	6
40 40 39 39 39	39 - 41	9
39 39 38 37 37	36 - 38	5
36 36 35 33 32	33 - 35	2
	30 - 32	1
	TOTAL	50

EL SIGUIENTE PASO SERÁ PARA CONSTRUIR LA COLUMNA DENOMINADA DE DESVIACIONES, GENERALMENTE SE FORMA TOMANDO COMO REFERENCIA EL LUGAR DONDE SE CONCENTRAN LA MAYOR CANTIDAD DE FRECUENCIAS, SIN EMBARGO, EXISTE OTRA ALTERNATIVA Y ES TOMAR COMO REFERENCIA LA MITAD DEL NÚMERO DE CASOS Y EN AMBOS CASOS A ESTE INTERVALO CORRESPONDERA EL NUMERO 0.

EN EL EJEMPLO QUE ESTAMOS MANEJANDO, ELEGIMOS LA SEGUNDA ALTERNATIVA, EN VIRTUD DE QUE HAY UNA BUENA CONCENTRACION DE CASOS TANTO EN LA PARTE SUPERIOR COMO EN LA INFERIOR, QUEDANDO LA ESTRUCTURA DE LA COLUMNA DE LA SIGUIENTE MANERA:

INTERVALOS	F	D
60 - 62	5	+5
57 - 59	4	+4
54 - 56	7	+3
51 - 53	2	+2
48 - 50	5	+1
45 - 47	4	0
42 - 44	6	-1
39 - 41	9	-2
36 - 38	5	-3
33 - 35	2	-4
30 - 32	1	-5

DEL NÚMERO 0 HACIA ARRIBA SERÁN NÚMEROS POSITIVOS Y HACIA ABAJO DEL 0 SERÁN NEGATIVOS.

LA ESTRUCTURA DE LA SIGUIENTE COLUMNA SERÁ LA MULTIPLICACIÓN DE LA FRECUENCIA POR LAS DESVIACIONES, O SEA, LA F POR LA D, EJEMPLO:

INTERVALOS	F	D	FD
60 - 62	5	+5	25
57 - 59	4	+4	16
54 - 56	7	+3	21
51 - 53	2	+2	4
48 - 50	5	+1	5
45 - 47	4	0	0
42 - 44	6	1	-6
39 - 41	9	-2	- 18
36 - 38	5	-3	- 15
33 - 35	2	-4	- 8
30 - 32	1	-5	- 5

AHORA, SUMAREMOS LOS RESULTADOS DE LOS NÚMEROS POSITIVOS Y LOS NEGATIVOS DE LA COLUMNA DE D POR SEPARADO, ES DECIR:

+25 - 6

+16 -18
+21 -15 POR LO TANTO LA SUMA ALGEBRAICA ES:
+ 4 - 8 +71 – 52 = 19
+ 5 - 5
71 -52

ESTE RESULTADO, 19, SERÁ EL QUE MANEJAREMOS COMO SFD = 19 Y EL CUAL AGREGAREMOS AL CONCENTRADO QUE TENEMOS DE DATOS Y FÓRMULAS, QUEDANDO DE LA SIGUIENTE MANERA :

N= 50
CM= 62
cm= 32
OSC = 30
INT= 3
SFD = 19

LA COLUMNA QUE A CONTINUACIÓN ESTRUCTURAREMOS, SE LLAMARÁ SUMA DE FRECUENCIAS POR DESVIACIONES, Y ÉSTA SE FORMARA DE LA MULTIPLICACIÓN DE LOS DATOS DE LA COLUMNA DF, DESVIACIONFS (D), POR LOS DATOS DE LA COLUMNA DE FRECUENCIAS POR DESVIACIONES (FD); YA FORMADA LA COLUMNA SE DENOMINARA FRECUENCIA POR DESVIACIONES AL CUADRADO (FD2), EJEMPLO:

ITERVALOS	F	D	FD	FD2
60 - 62	5	+5	25	125
57 - 59	4	+4	16	64
54 - 56	7	+3	21	63
51 - 53	2	+2	4	8
48 - 50	5	+1	5	5
45 - 47	4	0	0	0
42 - 44	6	-1	-6	6
39 - 41	9	-2	- 18	36
36 - 38	5	-3	-15	45
33 - 35	2	-4	-8	32
30 - 32	1	-5	-5	25

DEBO HACER MENCIÓN QUE CUANDO SE MULTIPLICAN LAS DESVIACIONFS (D). POR LAS FRECUENCIAS POR DESVIACIONES (FD), EL SIGNO NUEVAMENTE QUEDA EN POSITIVO.

AHORA SUMAREMOS ESTA COLUMNA QUE ACABAMOS DE FORMAR Y EL RESULTADO DE ESA SUMA SFRA LA SUMA DE LA FRECUENCIA POR DESVIACIONES SUMATORIA (FD2).

FD2
125
64
63
8
5
0
6
36
45
32
+ 25
409

EL RESULTADO LO AGREGAREMOS AL CONCENTRADO DE DATOS Y FÓRMULAS QUE ESTAMOS INTEGRANDO ENTONCES TENDREMOS:

N = 50
CM= 62
cm= 32
OSC= 30
INT= 3
SFD= 19
SFD2= 409

NOS DAREMOS A LA TAREA DE ENCONTRAR OTRO DATO QUE ES NECESARIO PARA PODER DESPEJAR LAS FÓRMULAS Y ASI PODER HACER LA DISTRIBUCIÓN ADECUADA DE LAS APRECIACIONES. ESTE DATO QUE, HACEMOS REFERENCIA ES EL COPRECTIVO (C) Y EL CUAL SE OBTIENE DE LA DIVISIÓN DE LA SUMA DE LAS FRECUENCIAS POR LA DESVIACIÓN (SED), ENTRE EL NÚMERO DE CASOS (N), ESTO ES LO SIGUIENTE:

SFD / N =C = 19 / 50 = 0.38O ES DECIR ; C= 0.380

Y SI MULTIPLICAMOS EL CORRECTIVO (C) POR SI MISMO, NOS DARÁ EL CORRECTIVO AL CUADRADO Y ENTONCES TENDREMOS:

0.380 x 0.380 = 0.1414 POR LO TANTO, C2=. 1414

ESTOS DATOS (C Y C2), LOS AGREGAMOS AL CONCENTRADO DE DATOS Y FÓRMULAS, QUEDANDO ASÍ:

N = 50
CM= 62
cm= 32
OSC= 30
INT= 3
SED= 19
SFD2= 409

C = 0.380
C2 = 0.1414

EL ÚLTIMO DATO QUE NECESITAMOS PARA DESARROLLAR LAS FÓRMULAS ES EL QUE CORRESPONDE A LA MEDIA SUPUESTA (Ms) Y ÉSTA CORRESPONDE AL PUNTO MEDIO DEL INTERVALO DE CLASE, CON DESVIACIÓN 0. TOMANDO EN CUENTA EL EJEMPLO QUE HEMOS ESTADO MANEJANDO, CORRESPONDE AL NUMERO 46, POR QUE SE UBICA EN EL PUNTO MEDIO DEL INTERVALO 45 - 47, ESTO ES:

INTERVALO	F	D	FD	FD2
60 - 62	5	+5	25	125
57 - 59	4	+4	16	64
54 - 56	7	+3	21	63
51 - 53	2	+2	4	8
48 - 50	5	+l	5	5
45 - 47	4	0	0	0
42 - 44	6	-l	-6	6
39 - 41	9	-2	-18	36
36 - 38	5	-3	-15	45
33 - 35	2	-4	-8	32
30 - 32	1	-5	-5	25

TENEMOS AHORA SI TODOS LOS DATOS NECESARIOS PARA PODER DESPEJAR LAS FÓRMULAS QUE SE DETERMINARON AL PRINCIPIO, Y QUE SE ENCUENTRAN EN EL CUADRO DE CONCENTRACION DE DATOS, ES DECIR:

N = 50
CM= 62
cm= 32
OSC= 30
INT= 3
SFD= 19
SFD2= 409
Ms = 46
SFD/N =C = 0.380
C2 =.1414

Y LAS FÓRMULAS QUE DEBEMOS DFSPEJAR SON LAS SIGUIENTES:

Ms+ (C x INT) = M
46 + (0.380 x 3) = 47.14, POR LO TANTO, M= 47 Y,

$$INT \sqrt{(SFD2/N)- C2} = S$$

$$3\sqrt{(409 / 50) -.1414} = 8.5,$$ POR LO TANTO, S=8 Y 1/2 S = 4

HABIENDO OBTENIDOS TODOS LOS DATOS Y TENIENDO EL RESULTADO DE HABER DESPEJADO LAS FÓRMULAS, EL CUADRO DE CONCENTRADO QUEDARA ASI:

N = 50
CM= 62
cm= 32
OSC= 30
INT= 3
SED= 19
SFD2= 409
Ms = 46
SFD/N =C = 0.380
C2 =0.1414
M = 47
S = 8
1/2S = 4

COMO MENCIONAMOS AL PRINCIPIO DE ESTE CAPÍTULO DECIR SINPLEMENTE QUE UN ALUMNO 0 JUGADOR REALIZO 45 ABDOMINALES O CORRIÓ EN 11 SEGUNDOS 50 METROS, NO INFORMA CON EXACTITUD EL DESEMPEÑO DE ESTE EN ESE TIPO DE PRUEBAS. SIN EMBARGO SI TOMAMOS EN CUENTA LOS RESULTADOS OBTENIDOS POR SUS CONPAÑEROS DE GRUPO O EQUIPO, TANTO A LOS MEJORES, COMO LOS PEORES QUE EL, TENDREMOS UNA IDEA MAS EXACTA DE LA POSICIÓN QUE TIENE CON RESPECTO A ÉSTOS.

POR TODO LO ANTERIOR, DEBEMOS EXPRESAR GRÁFICA Y OBJETIVANTE, Y TOMANDO COMO REFERENCIA EL EJEMPLO QUE HEMOS ESTADO DESARROLLANDO, EL JUICIO DE VALOR QUE, DETERMINE EL RESULTADO OBTENIDO EN LA PRUEBA DE ABDOMINALES.

PARA ELABORAR ESE CUADRO OBJETIVO, EMPEZAREMOS POR RETOMAR EL RESULTADO OBTENIDO EN LA MEDIA (M), Y ÉSTE, FUE EL NÚMERO 47, A PARTIR DE ESTE DATO SUMAREMOS HACIA ARRIBA UNA MEDIA SIGMA (1/2 S) O SEA EL NÚMERO 4 Y RESTAREMOS HACIA ABAJO TAMBIEN UNA MEDIA SIGMA, ESTO ES LO SIGUIENTE:

4 7+ 4 = 51 Y 47 - 4 = 43

DE ESTA MANERA YA TENEMOS ESTRUCTURADO EL PRIMER RANGO DEL PARÁMETRO QUE CORRESPONDE AL PUNTO MEDIO DE NUESTRA GRÁFICA, COLOCANDO EL PUNTO NÚMERO 43 EN LA PARTE IZQUIERDA Y EL 51 EN LA DERECHA DE LA COLUMNA DE INTERVALOS, QUEDANDO ASI:

INTERVALOS	PARAMETRO VALORACIÓN	No. ALUMNOS	PORCENTAJE	FACTOR
	EXCELENTE			
	MUYBIEN			
43 -51	BIEN 0 MEDIA			
	REGULAR			
	MINIMO			
	DEFICIENTE			

AL NÚMERO 51 LE SUMAREMOS OTRA MEDIA SIGMA, PARA CONOCER EL FACTOR SUPERIOR DEL OTRO INTERVALO Y A ESE RESULTADO SUMAREMOS OTRA MEDIA SIGMA PARA CONOCER EL FACTOR SUPERIOR DEL ÚLTIMO INTERVALO, EJEMPLO:

51 + 4 = 55 Y 55 + 4 = 59.

AGREGAREMOS AL PARAMETRO LOS FACTORES ENCONTRADOS Y FORMAREMOS LOS PARÁMETROS SUPERIORES:

INTERVALOS	PARAMETRO VALORACIÓN	No. ALUMNOS	PORCENTAJE	FACTOR
56 - 59	EXCELENTE			
52 - 55	MUYBIEN			
43 - 51	BIEN O MEDIA			
	REGULAR			
	MINIMO			
	DEFICIENTE			

PARA FORMAR LOS INTERVALOS DE LOS FACTORES INFERIORES PROCEDEREMOS DE LA MISMA MANERA, A PARTIR DEL NUMERO 43 RESTAREMOS MEDIA SIGMA (O SEA 4 PUNTOS) Y ASI TENDREMOS EL SIGUIENTE

PARÁMETRO Y EL ÚLTIMO LO INTEGRAREMOS RESTANDO OTRA MEDIA SIGMA (4 PUNTOS) AL RESULTADO ANTERIOR, ES DECIR:

43 - 4 = 39 Y 39 - 4 = 35, POR LO TANTO LA GRÁFICA QUEDARÁ ASI:

INTERVALOS	PARAMETRO VALORACIÓN	No. ALUMNOS	PORCENTAJE	FACTOR
56 - 59	EXCELENTE			
52 - 55	MUYBIEN			
43 - 51	BIEN 0 MEDIA			
39 - 42	REGULAR			
35 - 38	MINIMO			
- Ó 34	DEFICIENTE			

DEBEMOS RECORDAR QUE HASTA AQUI HEMOS UTILIZADO COMO EJEMPLO LOS RESULTADOS DE UNA PRUEBA DE ABDOMNALES APLICADA A UN GRUPO DE ALUMNOS, Y QUE ESTOS RESULTADOS SE ENCUENTRAN ORDENADOS AL PRINCIPIO DE ESTE EJEMPLO.

ESTOS DATOS, LOS UTILIZAMOS PARA CONFORMAR LA SIGUIENTE COLUMNA DEL PARÁMETRO, QUE SE DENOMINA NÚMERO DE JUGADOR. PARA ELLO BUSCAREMOS EN ESOS DATOS ORDENADOS CUÁNTOS JUGADORES HICIERON EL NÚMERO DE ABDOMINALES QUE SE ENCUENTRA EN CADA INTERVALO Y LOS UBICAMOS EN ESE CASILLERO.

Y ASI, DE ESTA MISMA MANERA SE PUEDE EVALUAR CADA UNA DE LAS CAPACIDADES FÍSICAS, DE ACUERDO AL CUADRO DE CONTROL DE LAS CAPACIDADES DE LOS JUGADORES O ALUMNOS.

EL RESULTADO DE TODAS Y CADA UNA DE LAS PRUEBAS DE CAPACIDAD FÍSICA, NOS DARÁN EN SU CONJUNTO EL RESULTADO TOTAL DE LA FORMA FÍSICA CON QUE CUENTAN LOS ALUMNOS.

AL ELABORAR PARÁMETROS CON RESULTADOS DE NUESTROS PROPIOS JUGADORES, TENDREMOS LA OPORTUNIDAD DE ESTABLECER MÍNIMOS Y MÁXIMOS DE LAS PRUEBAS, DESPUÉS DE SER APLICADAS TRES O CUATRO VECES, YA QUE TENDREMOS LAS CONSTANTES QUE PERMITAN DETERMINAR ESAS EXIGENCIAS, SIEMPRE Y CUANDO SEAN APLICADAS LAS PRUEBAS CON LAS MISMAS CARACTERÍSTICAS Y LOS MISMOS INDIVIDUOS Y DE ESTE MODO

NO TENER QUE RECURRIR A COMPARAR RESULTADOS CON PARÁMETROS ELABORADOS EN OTROS PAISES, CON OTRO TIPO DE PERSONAS, EN OTRAS CIRCUNSTANCIAS Y PARA OTROS FINES.

DOS ELEMENTOS MAS PARA LA EVALUACIÓN QUE DEBEMOS TOMAR EN CUENTA Y QUE SON FACTORES DETERMINANTES PARA EMITIR ESE JUICIO DE VALOR EN EL PROGRESO DE LOS ALUMNOS Y SON EL CONTROL DE SU PESO CORPORAL Y SU ESTATURA.

OTRA MANERA DE SABER CUÁNTOS JUGADORES CORRESPONDEN A CADA CASILLERO, LO PODEMOS ENCONTRAR EN LA COLUMNA DE FRECUENCIAS YA QUE ÉSTAS DETERIMNAN EL NÚMERO DE ALUMNOS QUE HICIERON ESAS ABDOMINALES. EJEMPLIFICAREMOS LO DICHO HASTA AQUI:

ABDOMNALES LOGRADOS POR LOS ALUMNOS

DATOS ORDENADOS:

62 61 60 60 60	ENTRE 56 Ó MÁS ABDOMINALES HAY 12 JUGADORES
59 58 57 57 56	ENTRE 52 Y 55 ABDOMINALES HAY 5 JUGADORES
56 56 55 55 54	ENTRE 43 Y 51 ABDOMINALES HAY 13 JUGADORES
54 53 51 50 50	ENTRE 39 Y 42 ABDOMINALES HAY 12 JUGADORES
49 48 48 47 46	ENTRE 35 Y 38 ABDOMINALES HAY 6 JUGADORES
46 45 44 43 43	ENTRE 34 Ó MENOS ABDOMINALES HAY 2 JUGADORES
42 42 42 41 41	
40 40 39 39 39	
39 39 38 37 37	
36 36 35 33 32	

POR LO TANTO, ESTA INFORMACIÓN LA VERTIMOS EN EL PARÁMETRO QUEDANDO DE LA SIGUIENTE FORMA:

	PARÁMETRO			
INTERVALOS	VALORACIÓN	No. ALUMNOS	PORCENTAJE	FACTOR
56- 59	EXCELENIE	12	24%	+
52 - 55	MUY BIEN	5	10%	+
43 - 51	BIEN O MEDIA	13	26%	+
39 - 42	REGULAR	12	24%	-
35 - 38	MÍNIMO	6	12%	-
- Ó 34	DEFICIENTE	2	2%	-

ESTE ES UN PARÁMETRO QUE NOS PERMITIRÁ ELABORAR UN JUICIO DE VALOR EN RELACIÓN A LA FUERZA GENERAL QUE, TIENEN LOS ALUMNOS QUE REALIZARON LA PRUEBA DE ABDOMINALES Y ASI DETERMINAR DE ACUERDO A SI ES UNA EVALUACIÓN INICIAL, INTERMEDIA O FINAL, LAS ALTERNATIVAS CORRESPONDIENTES PARA BENEFICIAR EL ENTRENAMIENTO DEPORTIVO DE LOS ALUMNOS O JUGADORES.

O SEA, QUE, SI LA INFORMACIÓN QUE, NOS BRINDA EL PARÁMETRO ES AL INICIO DE LA TENPORADA, NOS PERMTIRÁ PLANEAR ADECUADAMENTE LAS CARGAS DE TRABAJO PARA. SUPERAR LOS RESULTADOS OBTENIDOS DURANTE LA TEMPORADA ANTERIOR."[46][47][48]

[46] Valdez Rangel Francisco, MANUAL DE VALORACIÓN DE LA CAPACIDAD FÍSICA PARA LA EDUCACIÓN PRIMARIA, Edición de la DGENAMDF, México, 1996.
[47] Augusto Pila Teleña, EVALUACIÓN DE LA EDUCACIÓN FÍSICA Y LOS DEPORTES. Editorial Augusto Pila Teleña. Madrid, España, 1981.
[48] Avalos Falcon Eduardo, ESTADÍSTICA EDUCATIVA,Edición privada. ENSM, México 1978.

6.2. ANÁLISIS

6.2.1. MODELO DE REQUERIMIENTOS

Como se describió en el capítulo cuatro el modelo de requerimientos se divide en tres partes: modelo de dominio del problema, modelo de escenarios y descripción de interfaz.

Una primera vista del esquema de operación del sistema sería como sigue :

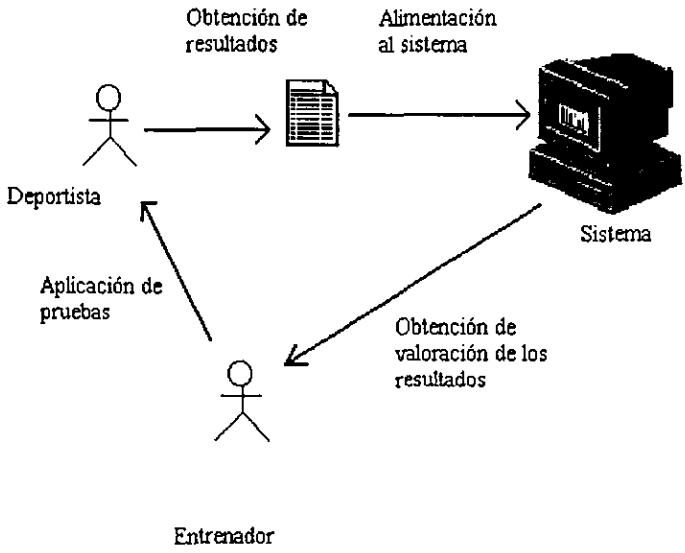


Figura.6.1. Esquema de operación del proceso a automatizar

MODELO DE ESCENARIOS

Este modelo como ya se mencionó nos ayuda a definir que existe fuera del sistema (actores) y que debe de ser desarrollado por el mismo (escenarios). El modelo de escenarios del sistema se vería de la siguiente forma :

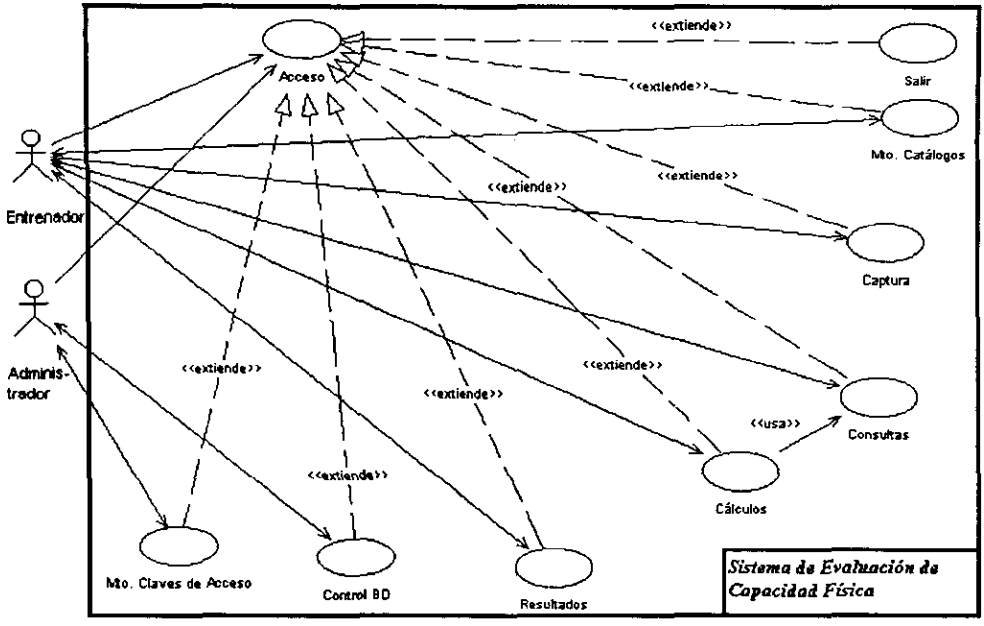


Figura.6.2. Modelo de escenarios

MODELO DE DOMINIO DEL PROBLEMA

El modelo del dominio del problema es una forma de expresar el modelo de escenarios en términos de objetos del dominio del problema, nos da una vista lógica del sistema usando objetos que por lo general tienen una contraparte directa en el ambiente de la aplicación, de este modelo es posible definir conceptos con los que el sistema va a trabajar. El mayor beneficio de este modelo es que los usuarios identifican todos los conceptos que aparecen en el de esta forma puede ser usado para definir lo que el sistema va a hacer.

El modelo de dominio del problema del sistema que se quiere es el que se muestra en la figura siguiente. Aquí podemos ver que existe un objeto donde se almacena la fecha de aplicación de las pruebas, otro donde se almacenan los datos personales de los individuos y otro más donde se almacenan los resultados que se obtienen de aplicarles las pruebas a los individuos de un conjunto en una fecha especifica, de esta manera se explican las relaciones expresadas en el sentido de que para cada fecha de aplicación pueden existir varios individuos que hayan realizado las pruebas.

Observamos también un objeto que es un usuario que selecciona un conjunto de individuos que se les hayan aplicado las pruebas, se pueden escoger un máximo de tres conjuntos de individuos, sobre ellos se realiza el análisis estadístico. Una vez concluido se guardan los resultados obtenidos y se presentan al usuario. El objeto administrador es heredado del objeto usuario.

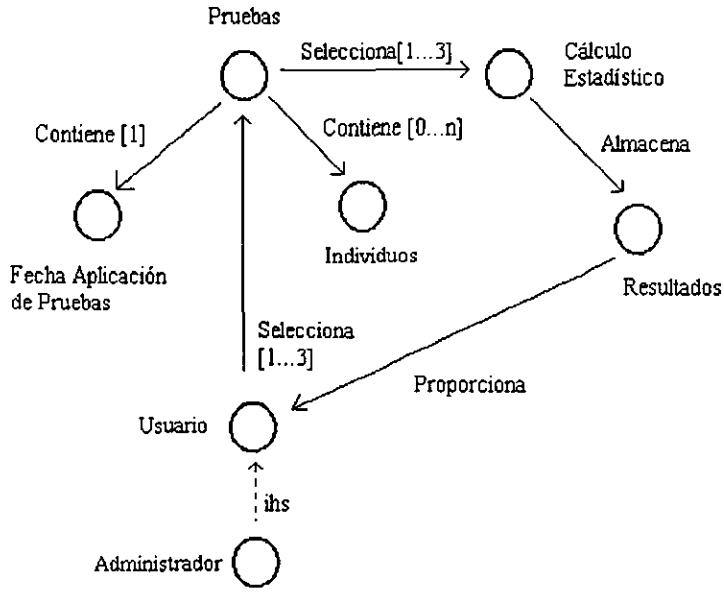


Figura.6.3. Modelo del dominio del problema.

Las letras "*ihs*" significan hereda, es decir en este caso, administrador hereda de usuario sus características, estas letras se usan en algunos otros modelos más adelante.

DESCRIPCIÓN DE LA INTERFAZ

Con respecto a la descripción de la interfaz podemos decir que se solamente se definió que tenía que ser bajo el esquema de Windows, es decir con manejo de ventanas y menús, de tal forma que se le facilite la operación a los usuarios, inclusive los que no tengan conocimiento de computación y/o estadístico. Y los resultados se deberían presentar en forma gráfica y tabular.

6.2.2. MODELO DE ANÁLISIS

En el modelo de análisis se va a plantear la estructura del sistema independientemente del ambiente de implantación. Como se describió anteriormente se utilizarán los tres tipos de objetos descritos. Debido a la gran cantidad de escenarios que se presentaron en el modelo de escenarios sólo se van a mostrar de este modelo en adelante, los más importantes del flujo principal que realiza el sistema, debido a que el paso de un modelo a otro es similar en cualquier escenario por lo tanto considero que para justificar el sistema no es necesario expresar todos los escenarios a lo largo de todos los modelos.

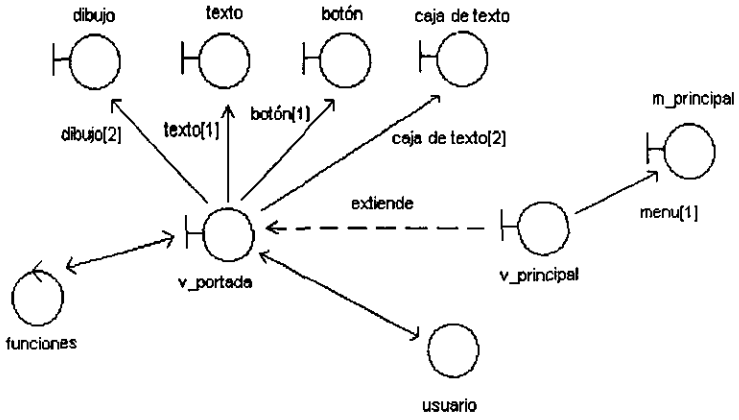


Figura.6.4. Modelo de análisis, escenario de acceso.

En este modelo de análisis del escenario podemos ver como un objeto de interfaz llamado v*_portada* (que va a ser una ventana) está compuesto por otros objetos: dos *dibujos*, un *texto*, un *botón* y dos *cajas de texto*. El objeto v*_portada* está heredado de el objeto v*_origen* (también es una ventana). Este objeto *origen* tiene interacción con el objeto *usuario*, para checar que la clave de acceso con la que se quiere entrar al sistema sea correcta. El objeto v*_principal* (es una ventana) extiende el comportamiento del objeto *portada* y además está compuesto por un objeto *m_principal* que es un menú. Se han usado algunos prefijos con la finalidad de identificar de forma rápida el tipo de objeto al que se hace referencia. De esta forma tenemos que para las ventanas el prefijo es "v" y para los menús es "m", si en los otros modelos se usa algún prefijo nuevo se especificará a que tipo de objeto pertenece.

En la descripción del modelo de análisis del escenario de acceso podemos ver las diferentes relaciones que se van a presentar en los otros escenarios, se hizo la descripción de este escenario debido a que es un escenario relativamente pequeño y la descripción no se extiende mucho. Como en los otros escenarios son más grandes se omitirá la descripción teniendo en cuenta que los tipos de relaciones que se usarán ya se explicaron en el escenario de acceso.

A continuación se muestra el modelo de análisis de otros tres escenarios el de captura, el de cálculo y el de resultados, en las figuras 6.5, 6.6, 6.7. En el escenario de cálculo que se presenta otra peculiaridad, esta es que en un objeto ventana de datos se colocó un "*" esto no está contemplado en el método de Jacobson pero lo utilizamos para ser más claros, en este caso existe un objeto *v_consultas* al que pertenece el objeto *ventana de datos**, sin embargo hay otro objeto que se hereda de *v_consultas* que se llama *v_cons_individuos* donde por herencia existe el objeto *ventana de datos* * , pero además cuando se hace la instancia ese objeto *ventana de datos* * toma los datos que va a presentar de otros objetos llamados *catálogos* e *individuos*.

Después de haber realizados el modelo de análisis para los escenarios, vamos a entrar más en detalle en los objetos de entidad indicando los atributos que tienen esto se observa en el esquema de atributos de los objetos figura 6.8.

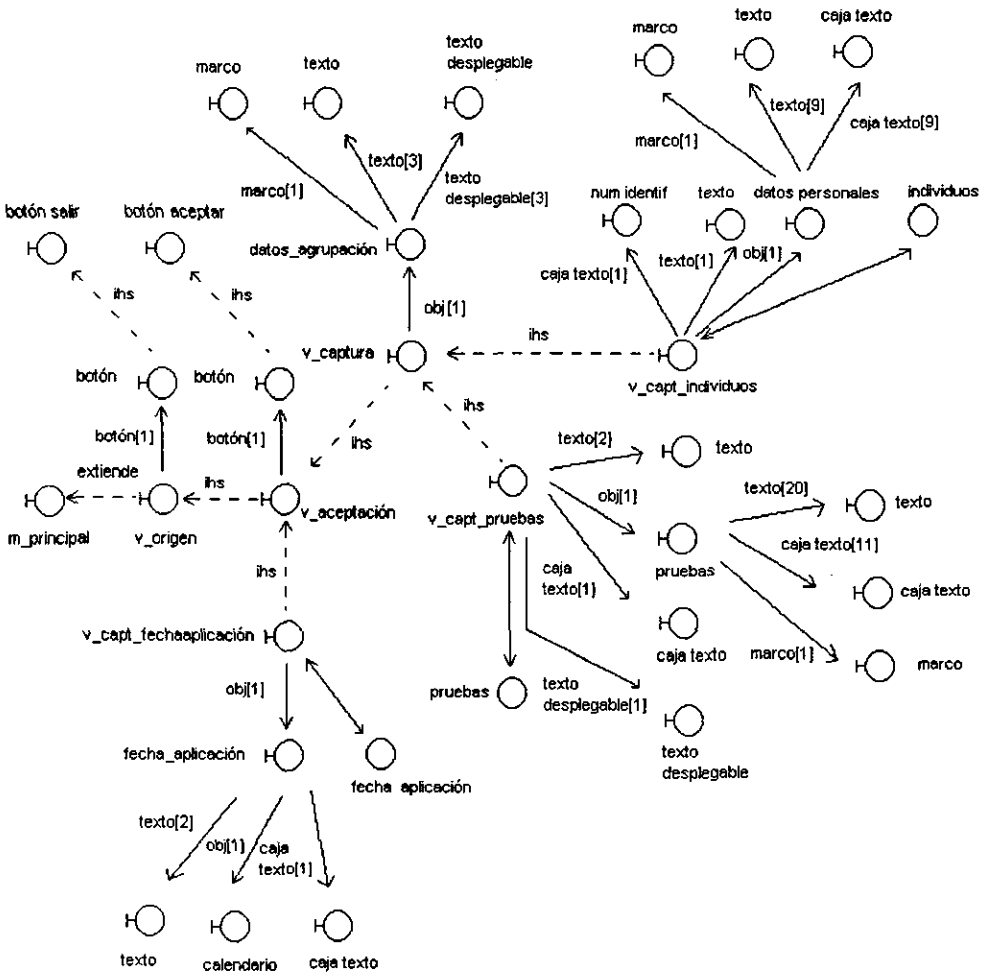


Figura.6.5. Modelo de análisis, escenario de captura.

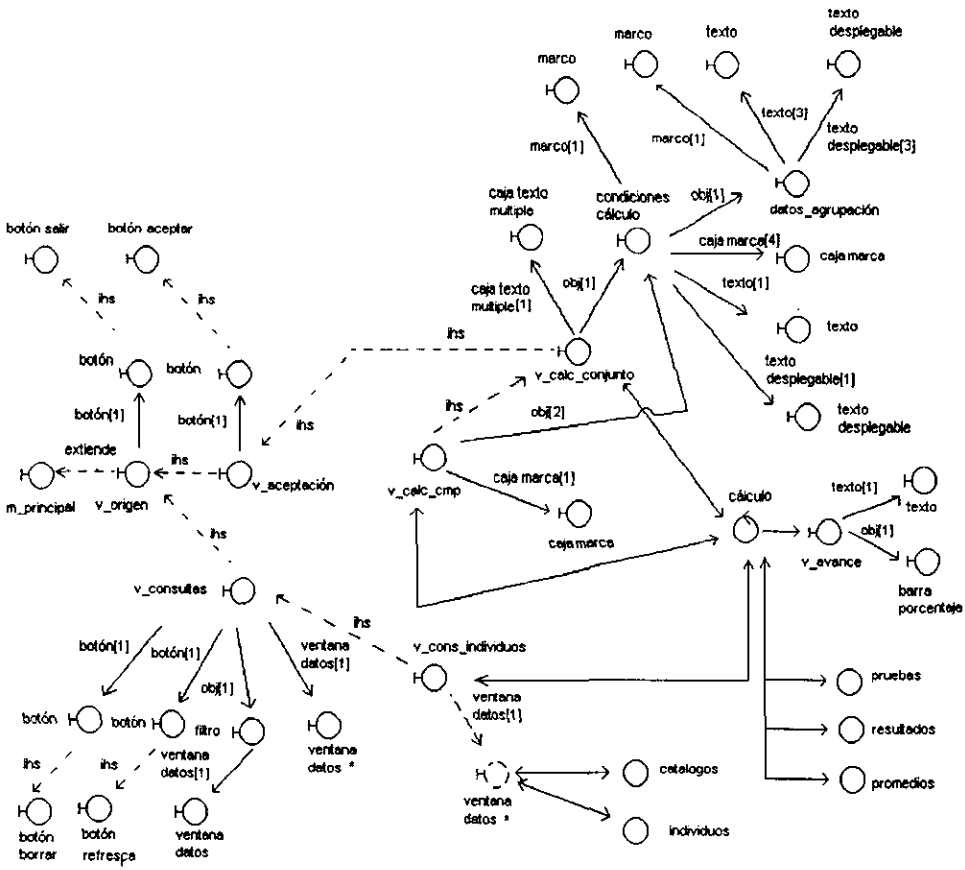


Figura.6.6. Modelo de análisis, escenario de cálculo.

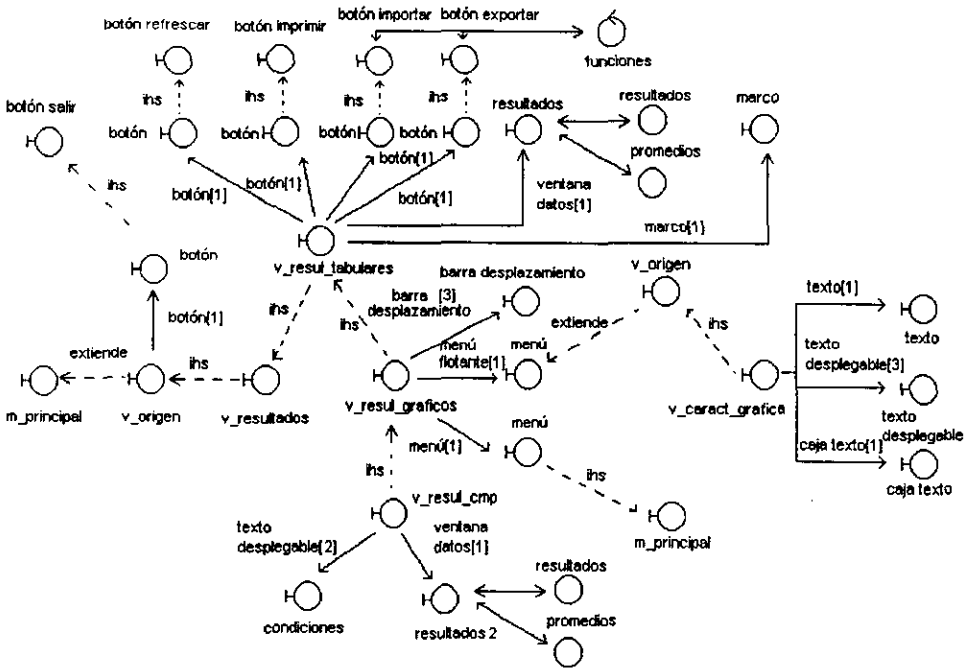


Figura.6.7. Modelo de análisis, escenario de resultados.

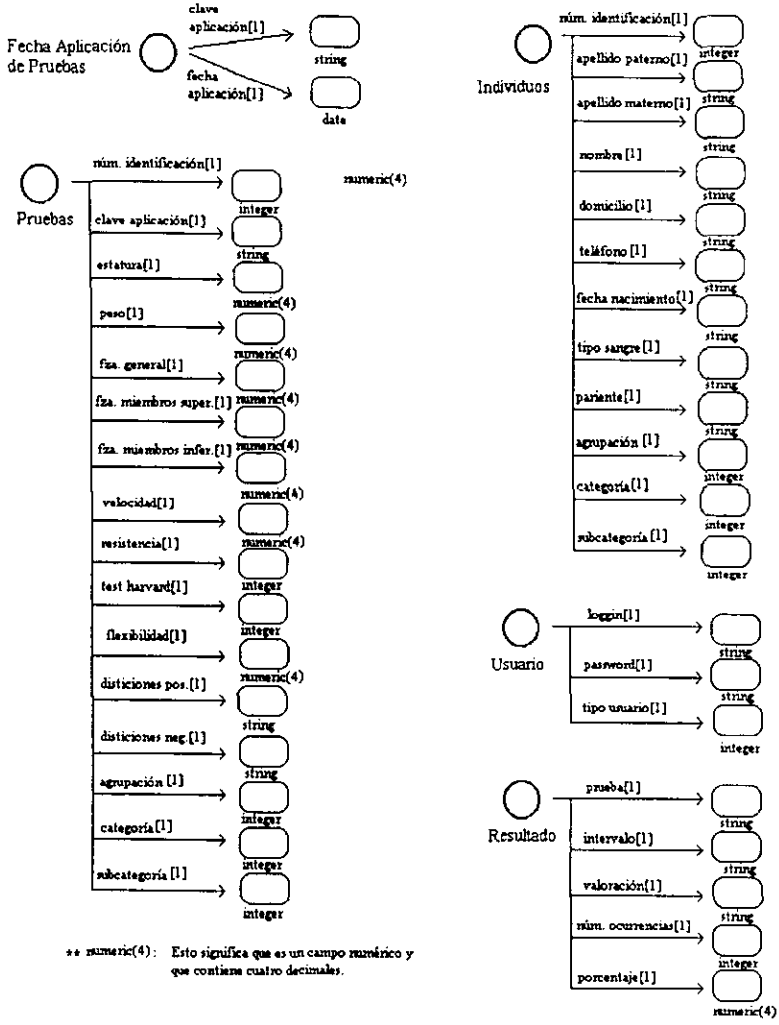


Figura.6.8. Atributos de los objetos entidad.

6.3. CONSTRUCCIÓN

6.3.1.MODELO DE DISEÑO

El modelo de diseño como se explicó en el capítulo cuatro está formado de diagramas del modelo de diseños de los escenarios y los diagramas de interacción de los escenarios, si se quiere algo más explícito se realizan diagramas de transición de estados.

También es en esta etapa donde se debe de elegir el lenguaje de implantación, como se describió en el capítulo cinco. Para desarrollar la aplicación se escogió el lenguaje de programación PowerBuilder, debido a que soporta los conceptos de orientación a objetos como objeto, herencia y polimorfismo, otra razón es que se cuenta con conocimiento del lenguaje, esto quiere decir familiaridad con el mismo. Es para una plataforma Windows, Windows95 y Windows98, Es fácil desarrollar la interfaz tipo Windows y para el caso del sistema a desarrollar se necesita eficiencia en tiempo de ejecución pero no es tan indispensable ya que no va a ser en tiempo real. Este lenguaje tiene presencia ya que se tienen varios tutoriales y herramientas para su utilización, además de que se tiene acceso a código ya existente de algunas aplicaciones desarrolladas previamente y de algunos tutoriales.

De esta forma vemos que PowerBuilder cuenta con las características que definimos en el capítulo cinco, y por eso se seleccionó para desarrollar la aplicación.

Con lo que respecta a la base de datos que se va a utilizar, PowerBuilder cuenta con un manejador de base de datos llamado SQLAnywhere, que es similar a Sybase con la diferencia que radica en una PC con Windows no en un servidor UNIX. PowerBuilder se conecta a la base de datos de forma casi automática, por lo tanto, no hay mucho que hacer para manejar ese vínculo.

Los diagramas del modelo de diseño de los escenarios se muestran a en las figuras 6.9, 6.10, 6.11,6.12, como ahora en lugar de los tres tipos de objetos se usan bloques se identificarán poniendo a los que son de entidad una "E" y a los de control un "C" esto no es parte de la metodología sólo se hace para poder interpretar de forma más clara a los objetos.

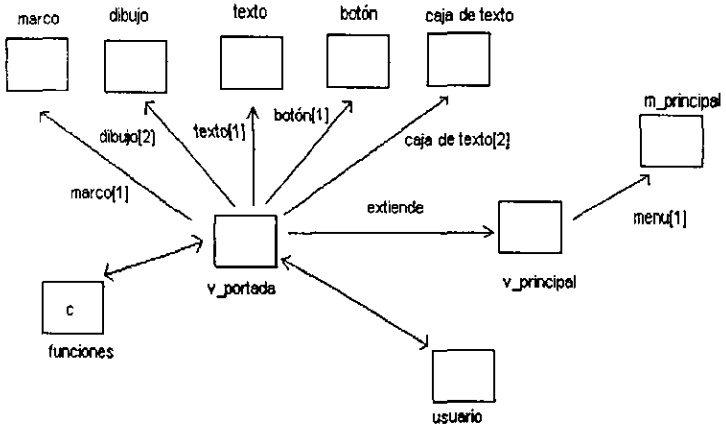


Figura.6.9. Modelo de diseño, escenario de acceso

Una diferencia que podemos encontrar en los diagramas del modelo de diseño es la forma en que se representa la extensión, en el modelo de análisis se representaba con una flecha punteada con origen en el objeto que extiende, sin embargo en el diagrama del modelo de diseño se representa como una flecha continua con origen en el objeto a ser extendido, es decir una asociación de comunicación. La extensión se describió anteriormente, significa que un escenario que va corriendo va a ser extendido, es decir, va a ejecutarse otro escenario completamente independiente y cuando se termina se regresa al escenario actual. Idealmente se quisiera que el escenario que extiende se insertara por si mismo en el escenario a extender, pero esto no es posible en los lenguajes de programación por lo tanto es en el modelo de diseño donde se decide como implementar la extensión.

Por lo tanto decimos que para la extensión se dejara que la iniciativa se origine en el escenario a ser extendido mediante el envío de un estimulo al escenario que extiende, como se mencionó antes una asociación de comunicación. Los demás modelos de diseño de los otros escenarios se muestran a continuación.

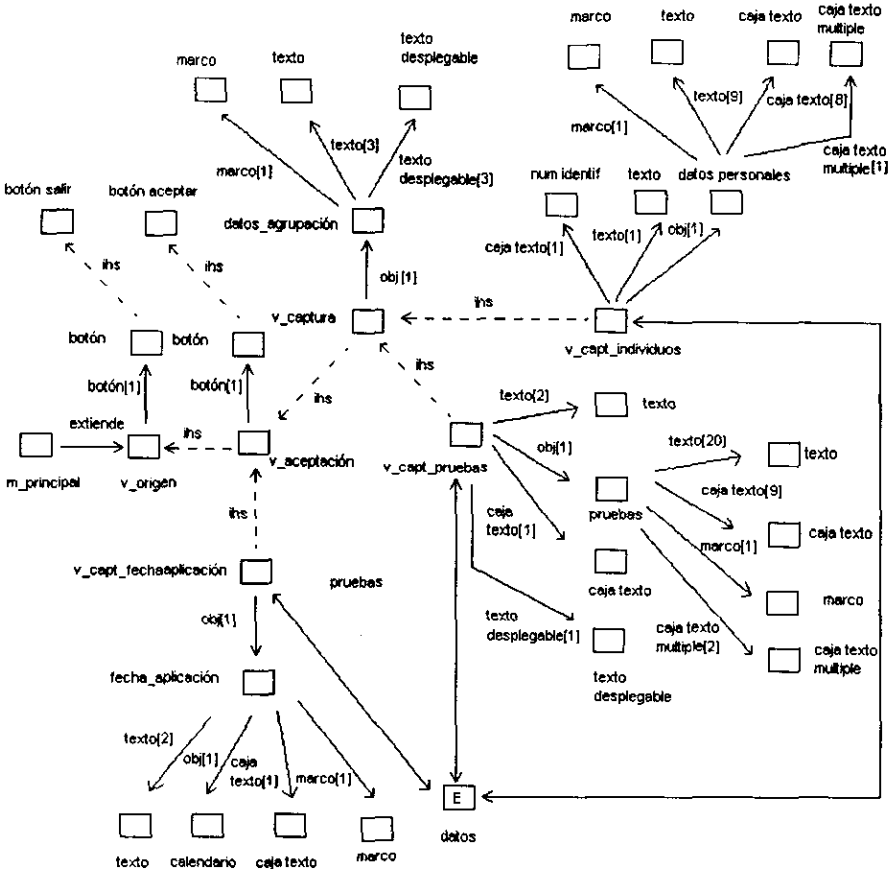


Figura.6.10. Modelo de diseño, escenario de captura.

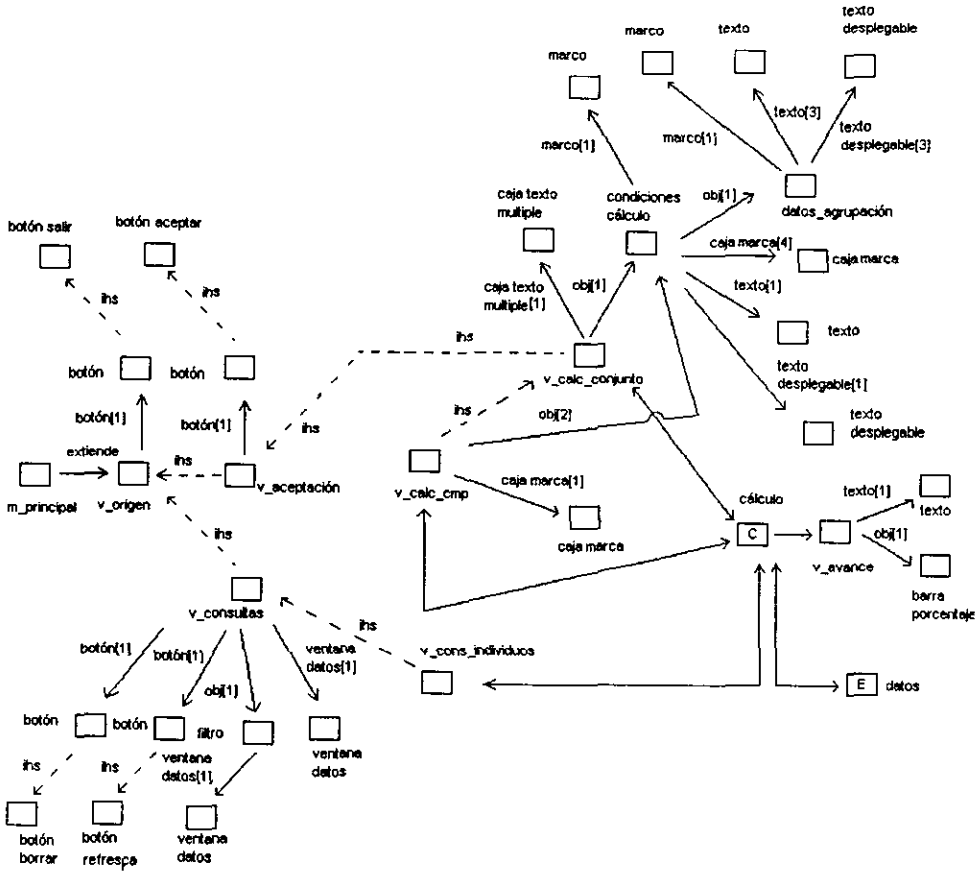


Figura.6.11. Modelo de diseño, escenario de cálculo.

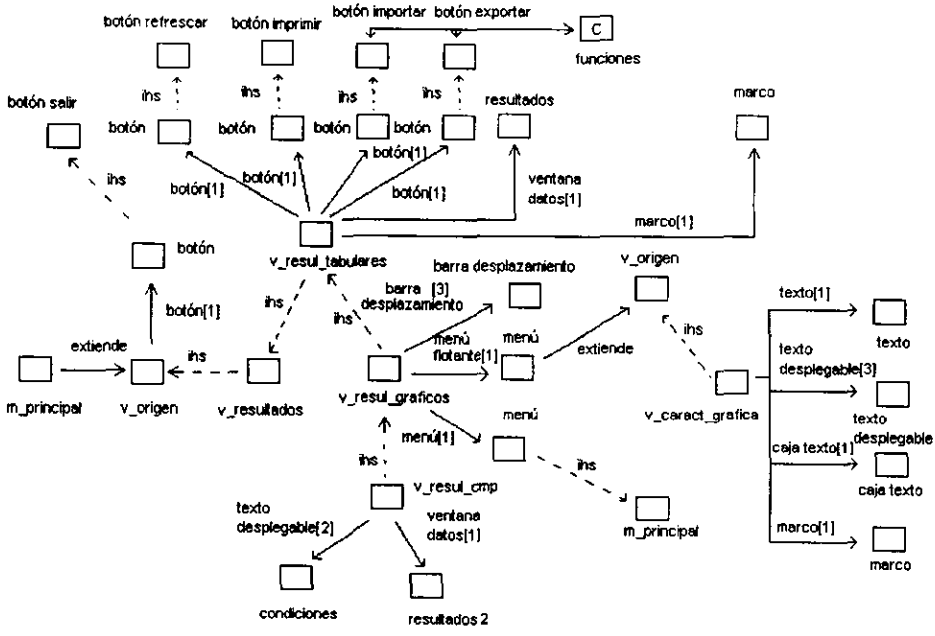


Figura.6.12. Modelo de diseño, escenario de resultados.

Posteriormente a los modelos de diseño se tienen que hacer los diagramas de interacción, éstos también se presentan en las páginas siguientes. Para los diagramas de interacción se definió la forma de representar la extensión mediante una posición de prueba, esto se ejemplifica en la figura 3.27 en el capitulo tres, sin embargo, en la herramienta que se hicieron los diagramas de interacción ROSE DE RATIONAL no era posible poner esta posición de prueba, por lo tanto no se puso en los diagramas de interacción.

Se ha realizado como muestra la operación del objeto de control llamado *cálculo* en notación SDL, el diagrama se muestra en la figura 6.13. Se escogió este objeto debido a su importancia ya que es el cual controla el proceso del cálculo estadístico además de que tiene interacción con el objeto entidad.

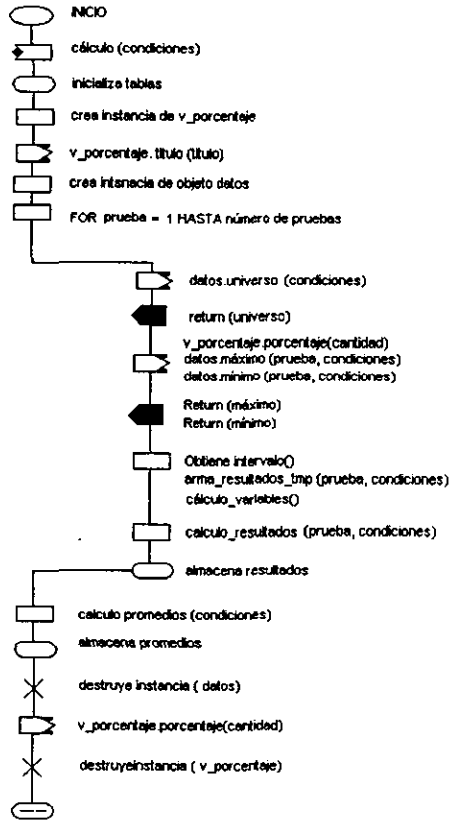


Figura 6.13. Diagrama en notación SDL

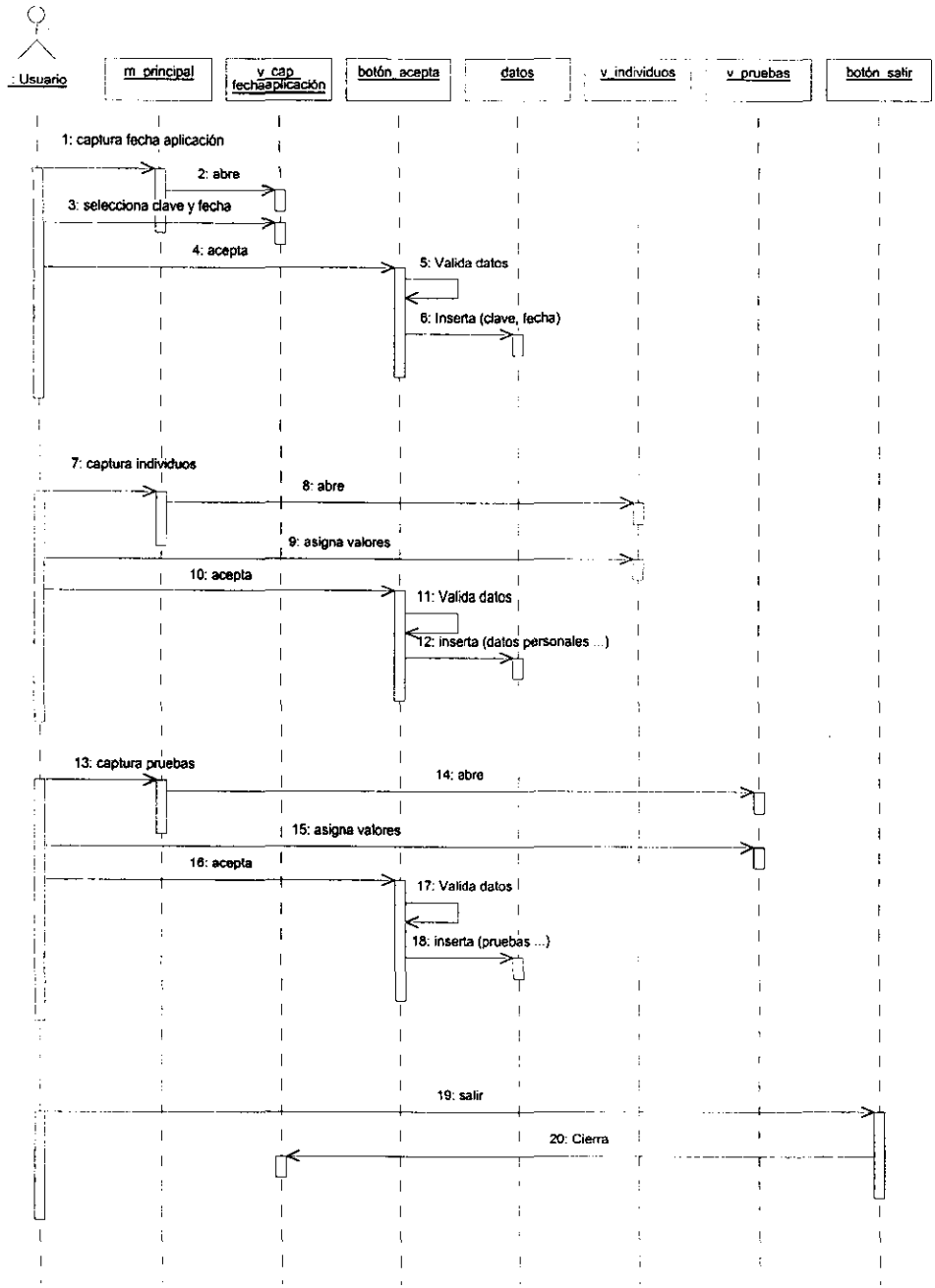


Figura 6.14. Diagrama de Interacción Escenario de Captura

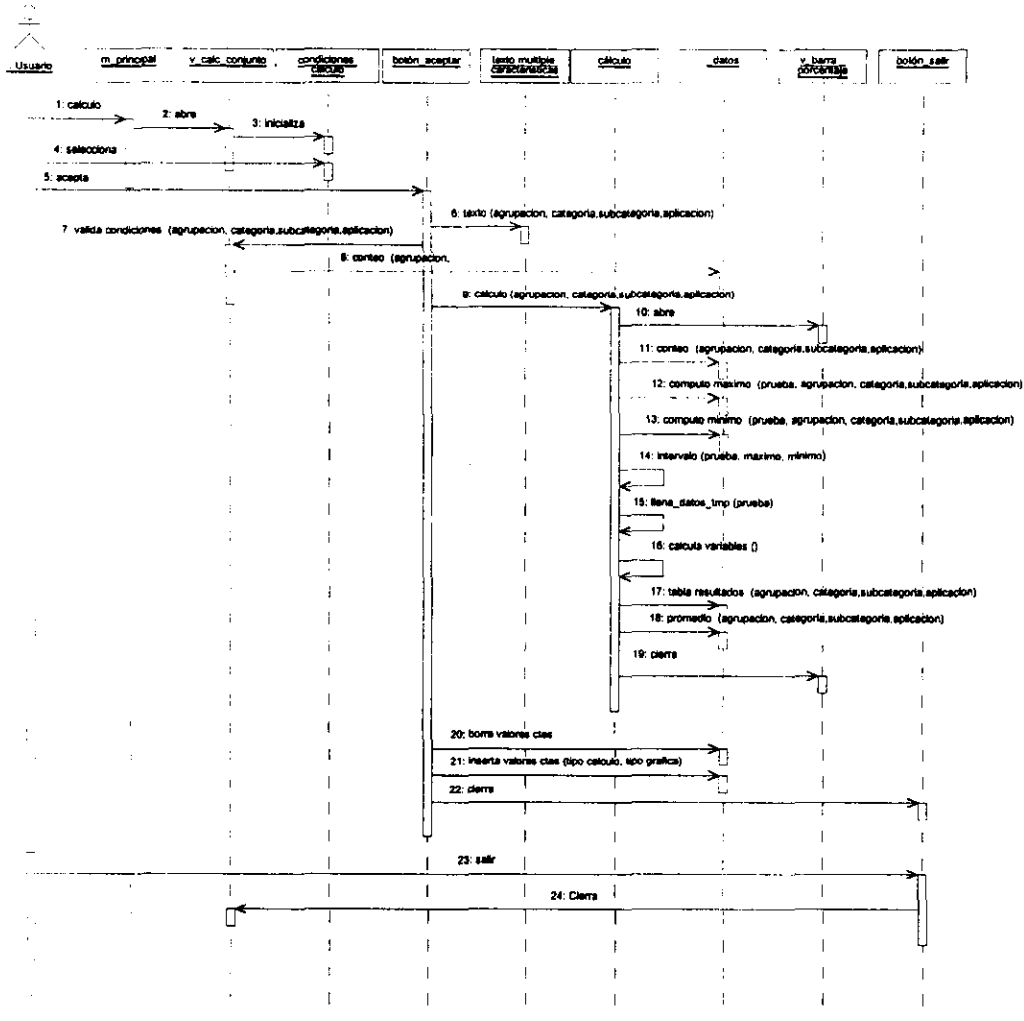


Figura 6.15. Diagrama de Interacción Escenario de Cálculo en Conjunto

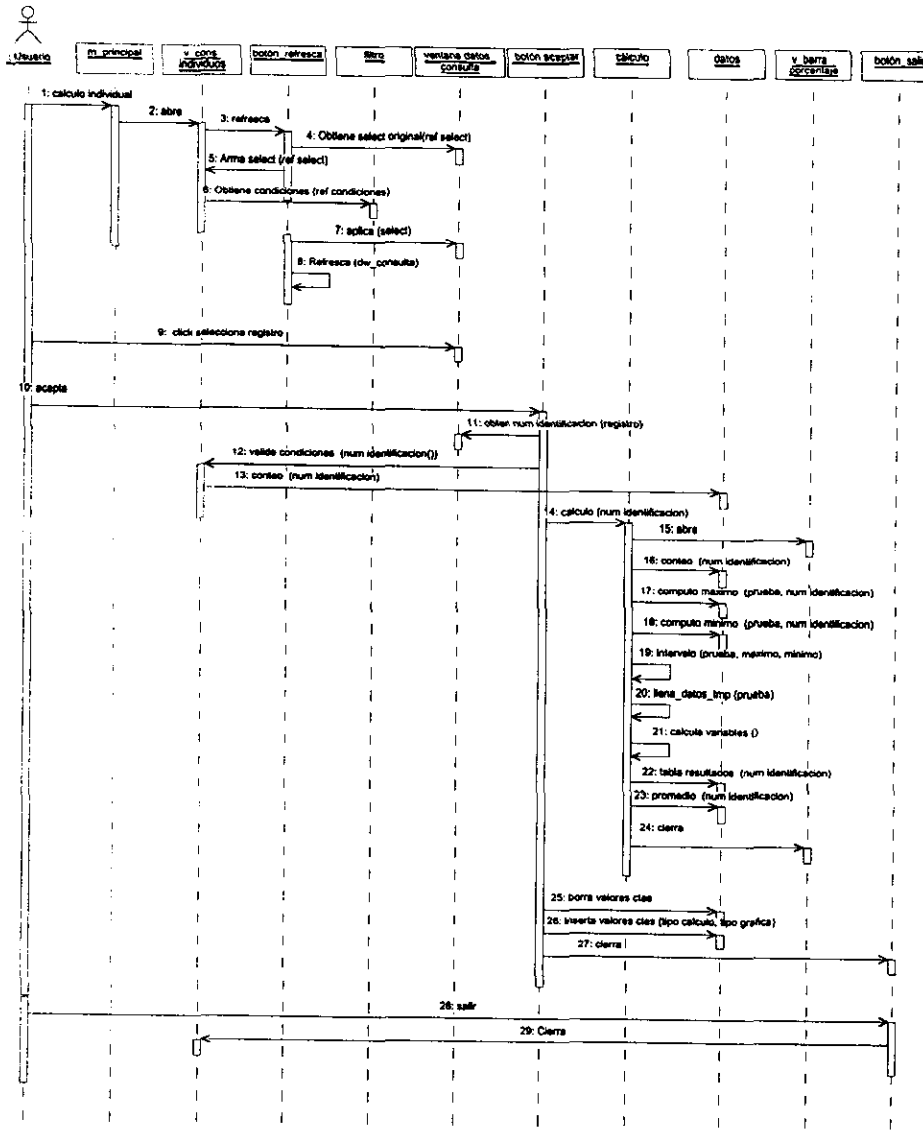


Figura 6.16. Diagrama de Interacción Escenario de Cálculo Individual

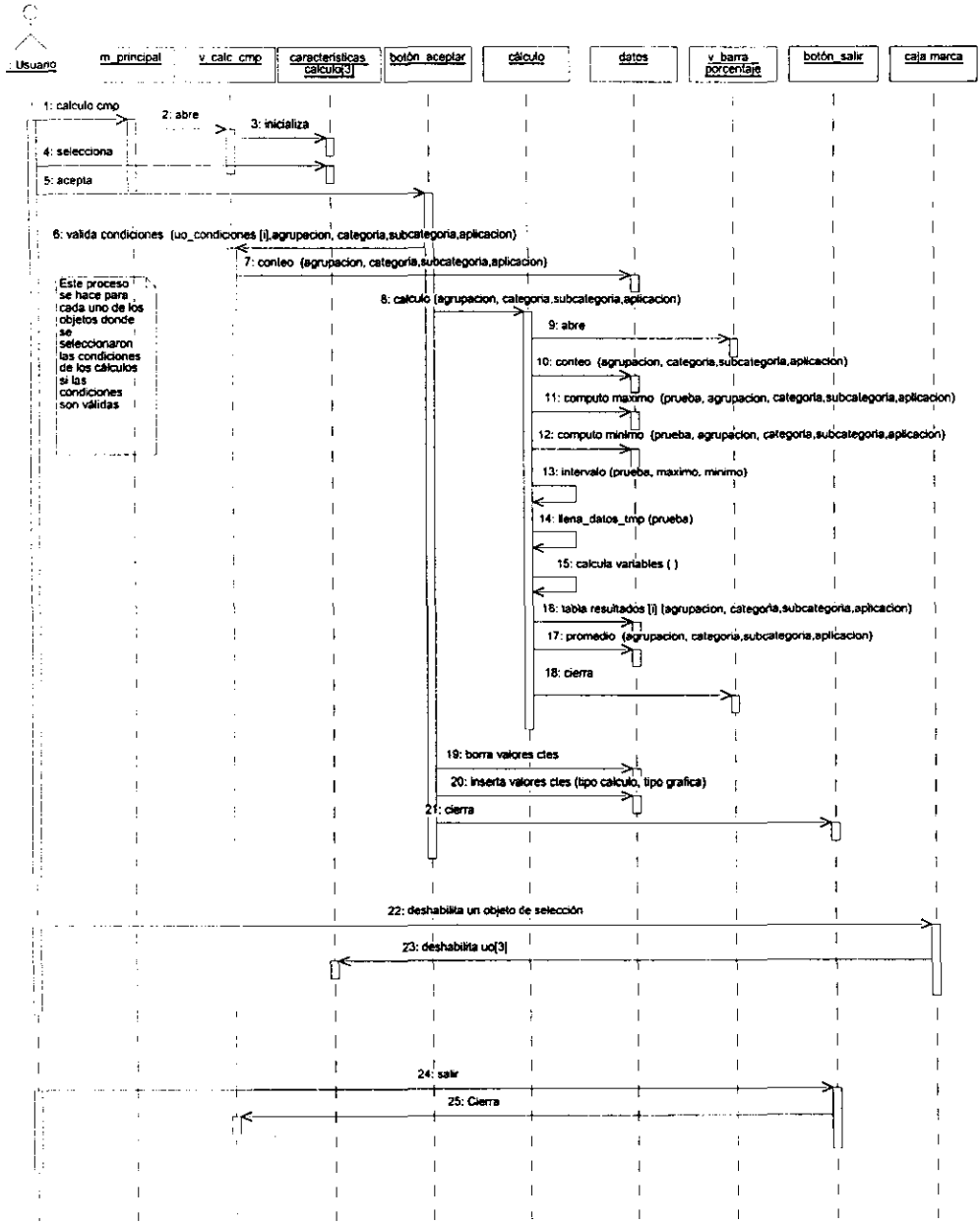


Figura 6.17. Diagrama de Interacción Escenario de Cálculo Comparativo

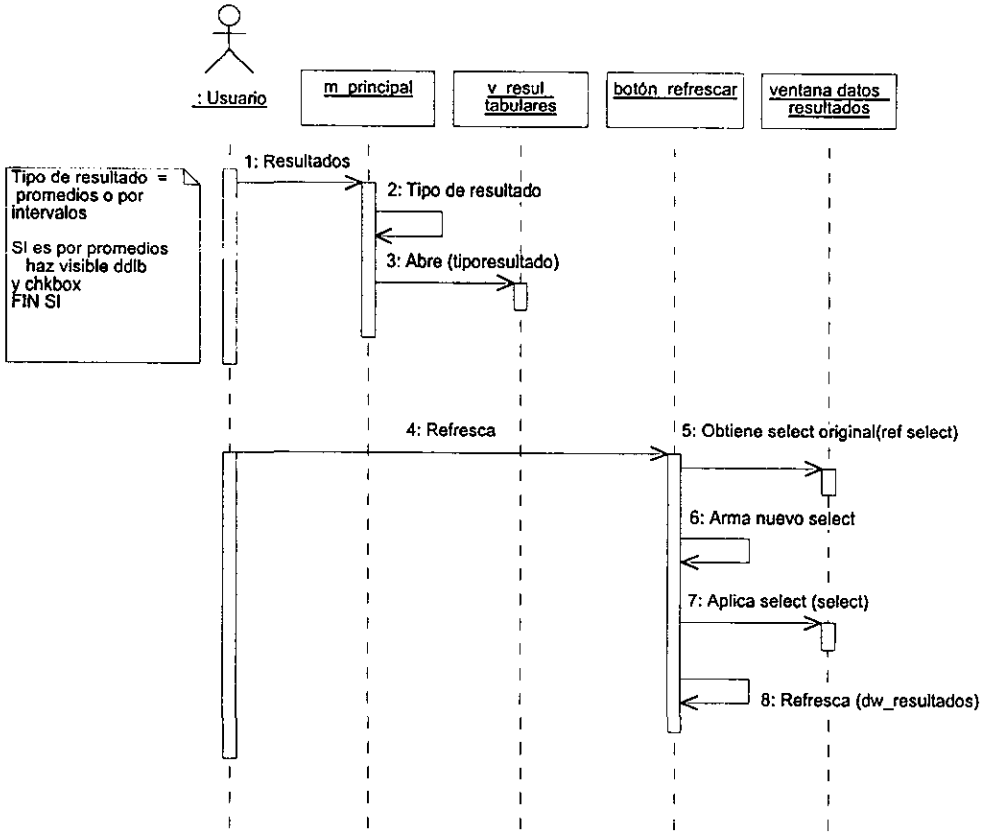


Figura 6.18. Diagrama de Interacción Escenario de Resultados Tabulares a)

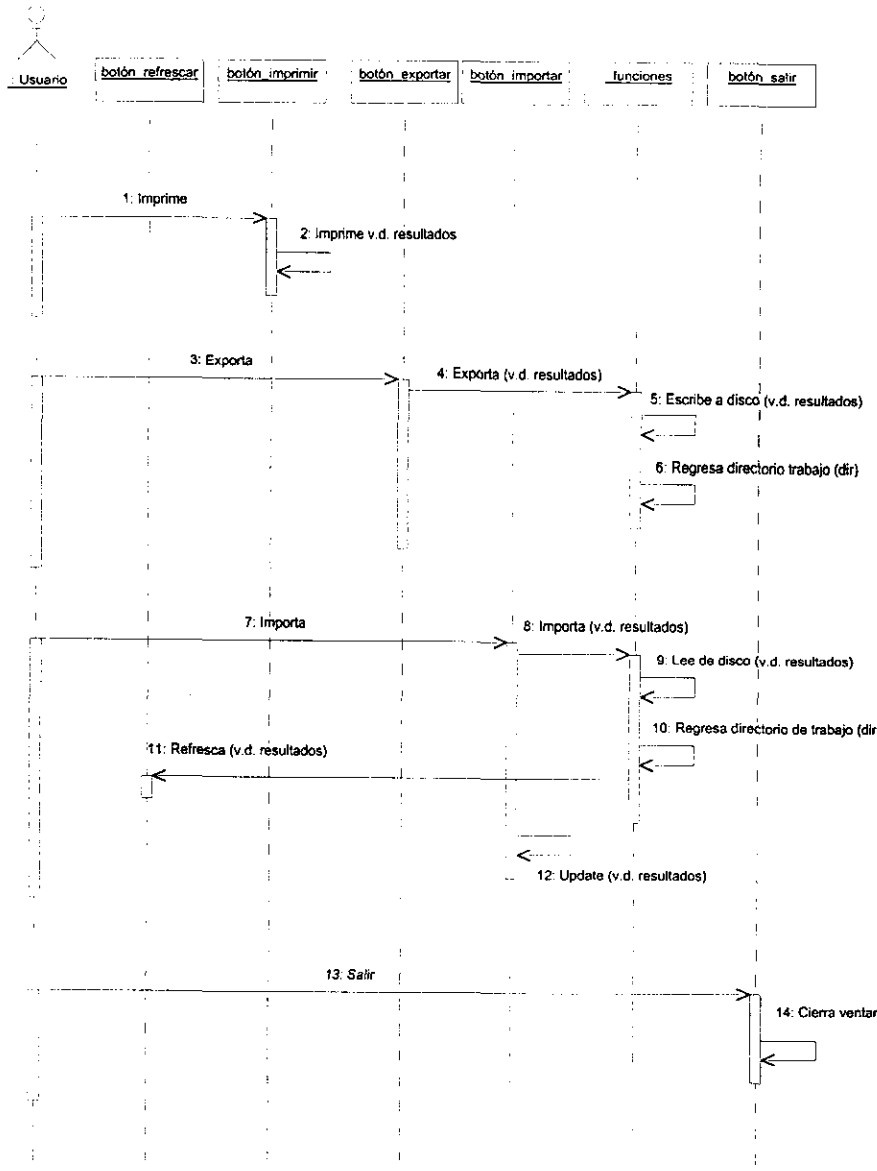


Figura 6.19. Diagrama de Interacción Escenario de Resultados Tabulares b)

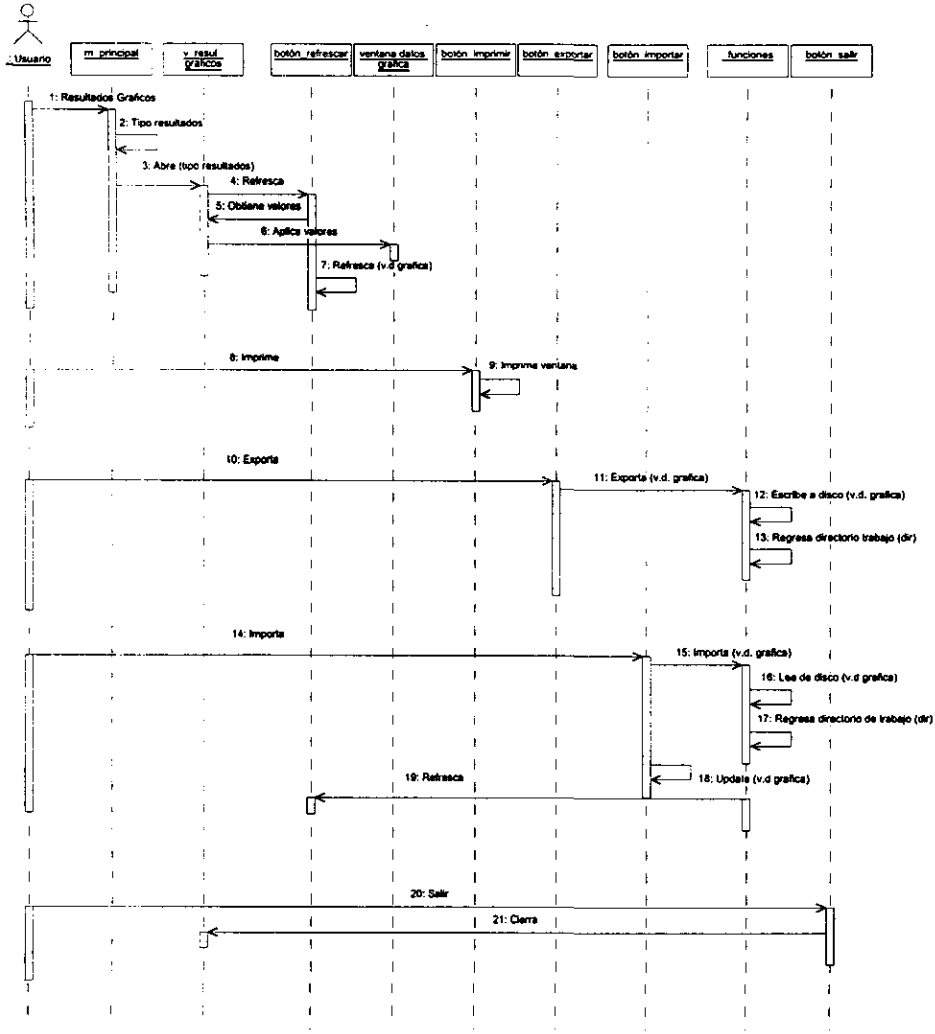


Figura 6.20. Diagrama de Interacción Escenario de Resultados Gráficos a)

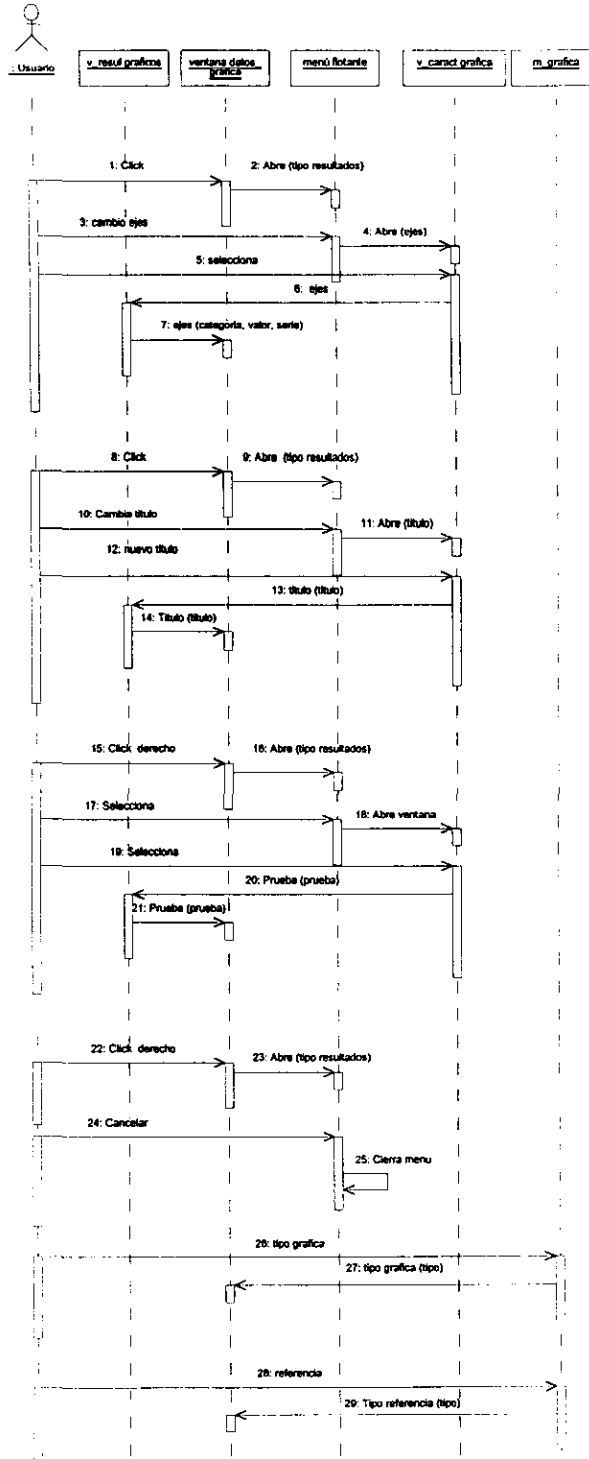


Figura 6.21. Diagrama de Interacción Escenario de Resultados Gráficos b)

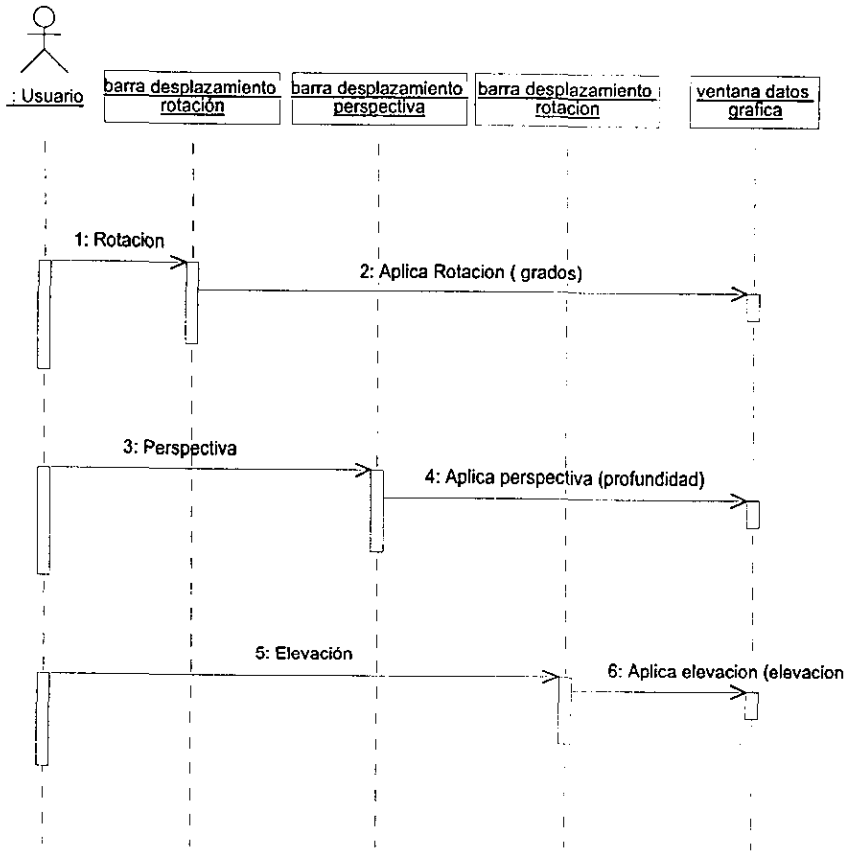


Figura 6.22. Diagrama de Interacción Escenario de Resultados Gráficos c)

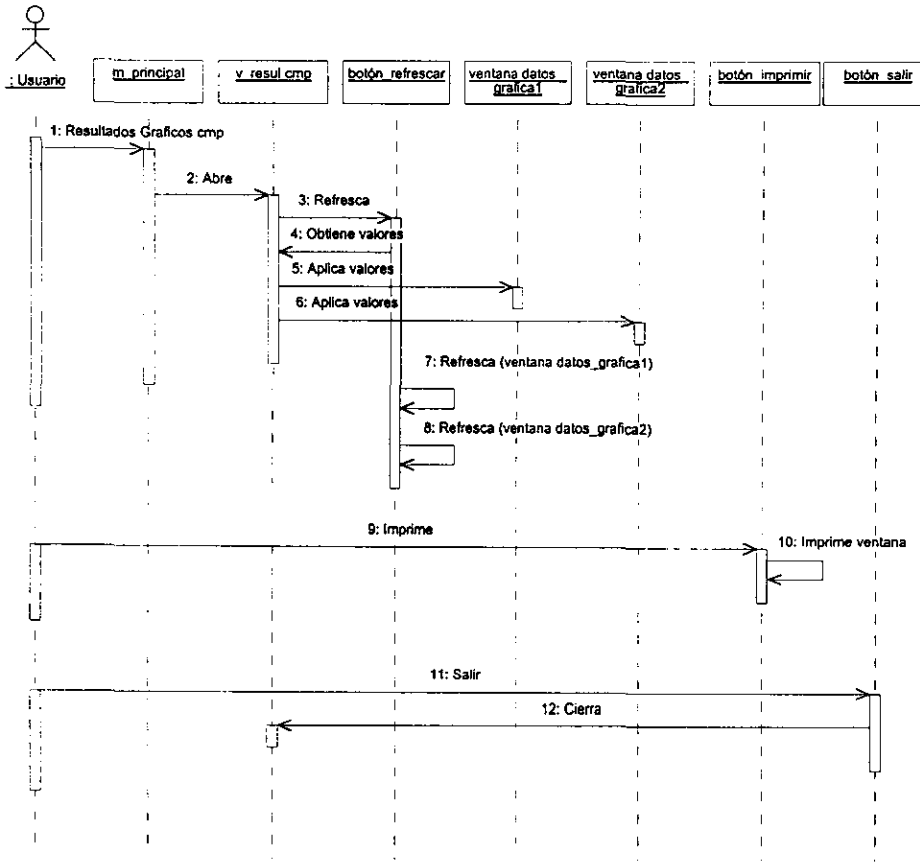


Figura 6.23. Diagrama de Interacción Escenario de Resultados Gráficos Comparativos a)

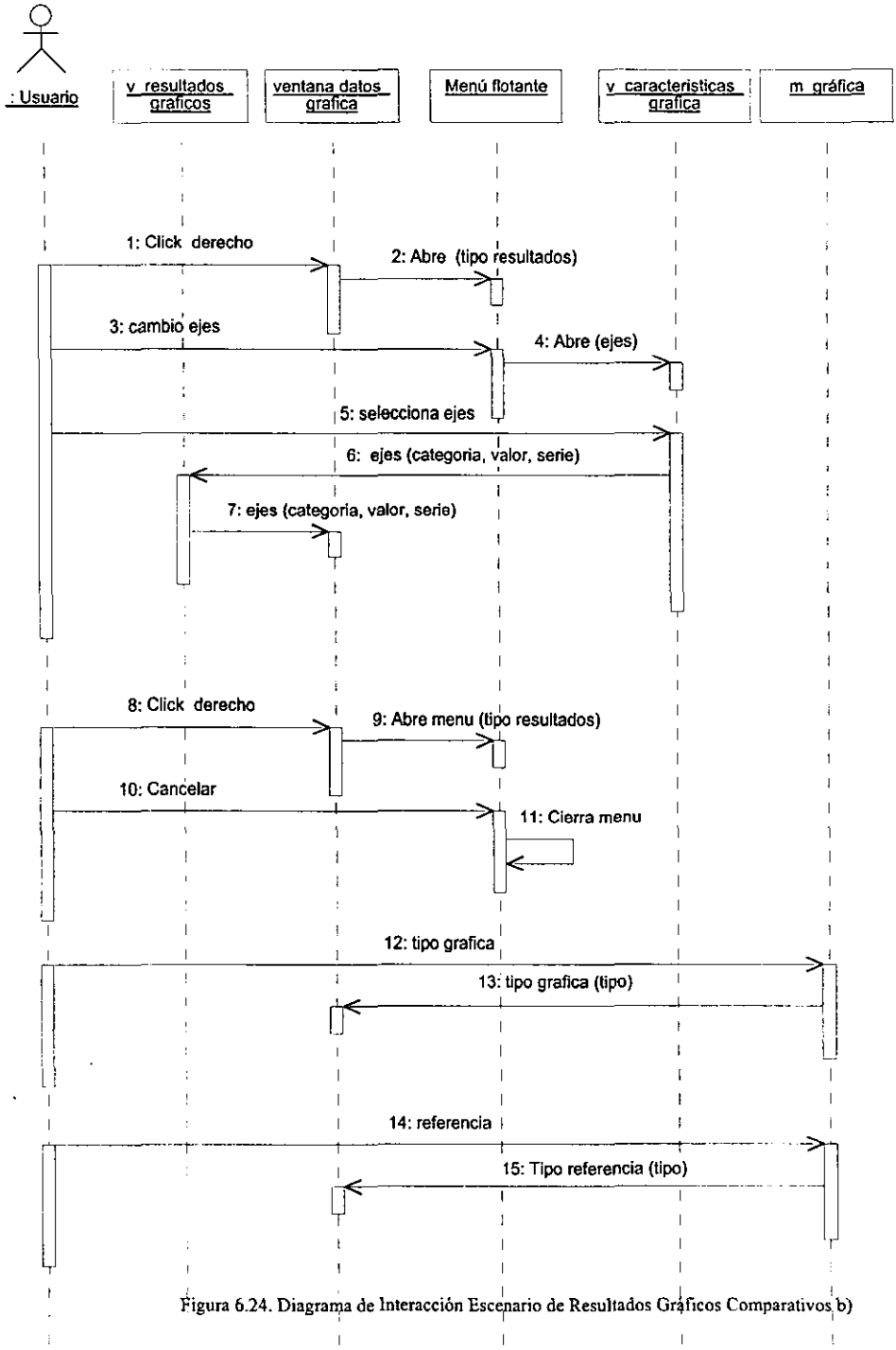


Figura 6.24. Diagrama de Interacción Escenario de Resultados Gráficos Comparativos b)

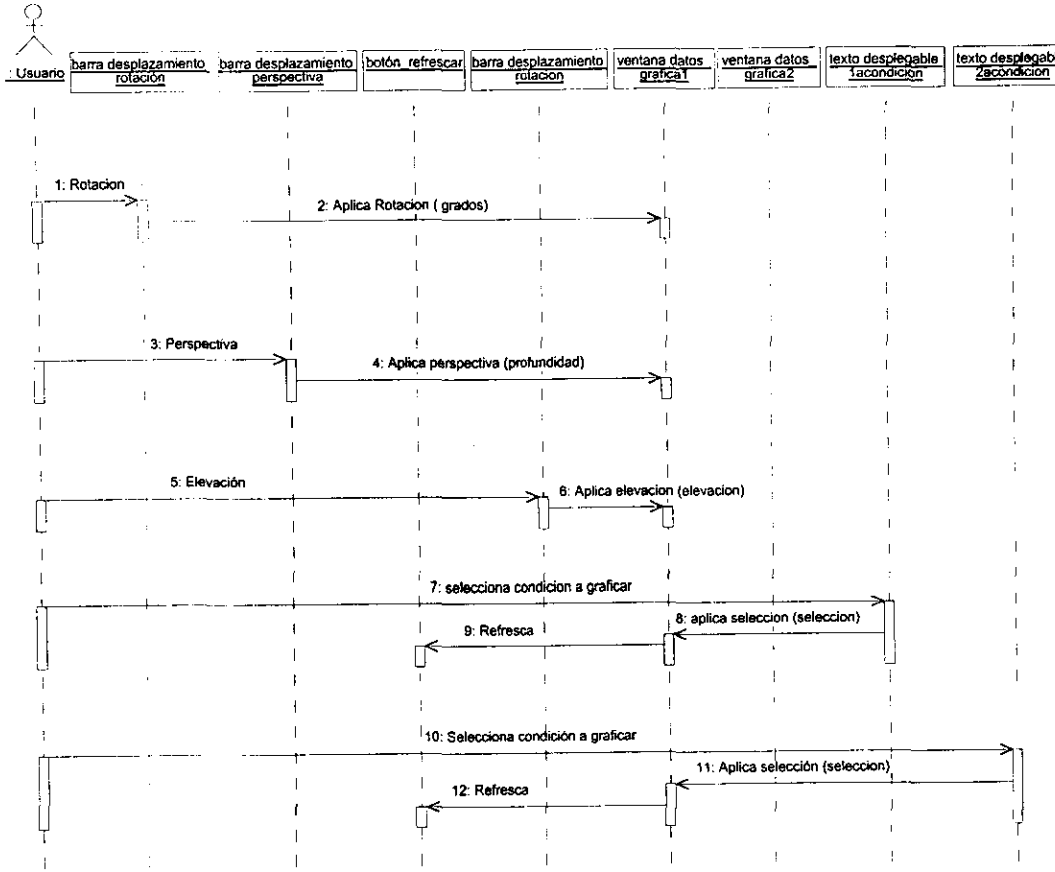


Figura 6.25. Diagrama de Interacción Escenario de Resultados Gráficos Comparativos c)

6.3.2. MODELO DE IMPLANTACIÓN

En este modelo se realiza la codificación de la aplicación de acuerdo al lenguaje seleccionado, para nuestro caso se utilizó PowerBuilder el cual cumple con las características de orientación a objetos como ya se mencionó, de esta forma se codificó el sistema y se concluyó.

Durante la codificación se realizaron hizo la verificación de cada bloque y en conjunto con los que interactúa, es decir, se hicieron las pruebas de caja blanca y caja negra de unidad, también se hicieron pruebas de integración ascendente, estás arrojaron algunos errores y se corrigieron durante el desarrollo. Esto se realizó con la herramienta de "debugg" que proporciona el lenguaje.

Una vez hechas estas pruebas se realizó la prueba del sistema esto con ayuda de personas que iban a utilizar el software, y con unas pruebas realizadas manualmente a un conjunto de individuos, alumnos de la preparatoria 8 de la UNAM, una vez que se tuvieron los resultados se compararon con los obtenidos mediante el sistema y se concluyó que el sistema funciona correctamente arrojando valores correctos, por lo cual podemos decir que las decisiones que se tomen con respecto a los resultados obtenidos mediante el sistema serán objetivas y útiles para los preparadores físicos.

En el Apéndice A se pueden observar los resultados de las pruebas que se obtuvieron siguiendo el proceso descrito en 6.1 Descripción del Problema: Requerimientos. Las pruebas se realizaron utilizando Excel, se muestran los cálculos y las gráficas obtenidas en la prueba manual y se muestran los obtenidos mediante el sistema.

Con respecto a las pruebas de recuperación para el sistema no son relevantes debido a que si por fallos no se lograra cualquier proceso (captura, actualización, consulta, resultados, cálculo, etc.) se podría volver a hacer. Para las pruebas de seguridad se había realizado durante la codificación y se probó la estructura implantada es correcta, sin embargo se depende de los usuarios para hacerla más confiable, es decir, no podemos asegurar la integridad de las claves de acceso de las personas tienen que ser responsables ellas mismas.

Las pruebas de resistencia no son de importancia debido a que por mucha información de individuos o pruebas lo que se utiliza son funciones agregadas y no importa la cantidad de datos, por ejemplo siempre se utiliza obtener el máximo de un conjunto de datos con algunas condiciones, si son pocos o muchos siempre va a entregar un dato. La prueba de sensibilidad no se hizo ya que no hay un conjunto de valores que ocasiones problema para el sistema. La prueba de rendimiento se realizó obteniendo tiempo muy constantes y muy buenos en comparación con los tiempo de llevar a cabo el proceso de forma manual. De forma manual se puede tardar una persona trabajando ocho horas aproximadamente de tres cinco días dependiendo de factores como conocimiento del método, facilidad para realizar operaciones con algún medio. Con el sistema lo que se tarda es aproximadamente 30 segundos promedio, independientemente del número de individuos a evaluar.

6.3.3. RESULTADOS.

En esta sección se va a mostrar mediante matrices los requerimientos del sistema que como se explicó fueron satisfechos por varios escenarios, con respecto a los objetos con los que se implementaron, los objetos a los que se hace referencia están en el modelo de diseño de los escenarios.

Cabe hacer notar que se están tomando los objetos representativos, sin embargo éstos pueden estar compuestos de otros objetos que en conjunto desarrollan la operación o escenario al que se hace referencia.

Existen además algunos requerimientos que son de forma genérica y que así se abordan, por ejemplo, se describió la necesidad de que fuera un sistema que presentara una interfaz fácil de utilizar e interpretar por gente que no necesariamente estuviera relacionada con la computación. Esto se logró desarrollando el sistema para ambiente Windows, por lo tanto todas las ventanas del sistema, así como sus componentes en conjunto cubren ese requerimiento.

También se pedía automatizar un proceso estadístico definido y reducir el tiempo que lleva hacerlo manual, esto se logró con el objeto central que es el que controla el proceso estadístico, con respecto al tiempo de respuesta la verdad es que realizar el proceso manualmente es sumamente tardado, posiblemente días, de esta forma reducirlo no fue tanto problema debido a la capacidad de procesamiento de información de las computadoras, sin embargo se redujo considerablemente.

A continuación se muestran las matrices que se explicaron anteriormente en la tabla 6.1.

OBJETOS / ESCENARIOS	v_portada	v_capt_fechaplicación	v_capt_pruebas	v_capt_individuos	v_cons_individuos	v_calc_conjunto	v_calc_emp
Control de Acceso al sistema	X						
Captura de datos		X	X	X			
Selección de universos variados para análisis					X	X	X
Cálculo estadístico y obtención de resultados							
Manipulación de Resultados Tabulares							
Manipulación de Resultados Gráficos							
Fácil acceso operaciones del sistema							
Funciones varias de S.O.							

OBJETOS / ESCENARIOS	v_resul_tabulados	v_resul_graficos	m_principal	usuario	datos	cálculo	funciones
Control de Acceso al sistema				X			
Captura de datos					X		
Selección de universos variados para análisis							
Cálculo estadístico y obtención de resultados					X	X	
Manipulación de Resultados Tabulares	X						
Manipulación de Resultados Gráficos		X					
Fácil acceso operaciones del sistema			X				
Funciones varias de S.O.							X

Tabla 6.1. Matrices de Resultados

Queda claro que los objetivos y requerimientos que se plantearon al principio se lograron satisfactoriamente, sin embargo analizando el sistema terminado, podemos ver que pueden existir algunas opciones de crecimiento y/o mejoras.

Como se planteó el sistema en un principio se tienen un conjunto de pruebas preestablecidas debido a que son los principales indicadores de la capacidad física, esto implica la forma de tratar las pruebas también ya está establecida, esto del tratamiento de las pruebas se refiere a que hay dos tipos de resultados que arrojan las pruebas, vamos a llamarlo "negativos" y "positivos", esto sólo para diferenciarlos. Así tenemos que los negativos sus características "es mejor hacer menos" esto es por ejemplo, una prueba de velocidad es más rápido el individuo que hace menos tiempo, por el contrario en una prueba de positiva, por ejemplo fuerza es mejor el individuo que hace más.

Aquí podemos observar que el sistema siempre va a hacer el análisis sobre las pruebas ya establecidas y las va a tratar de la manera en que se estableció, por lo tanto si hubiera la necesidad de cambiar las pruebas se debería de modificar el tratamiento, en este sentido pienso que se podría modificar el sistema para dejar una forma de configurar el sistema con respecto a las pruebas a realizar que se van a realizar y como va a ser su tratamiento es decir si son "negativas" o "positivas".

Para tener un análisis completo de la preparación de un deportista es necesario poder analizarlo en otros dos aspectos además del de capacidad física, estos aspecto son psicológico y médico, entonces un posible crecimiento sería el desarrollar esos otros dos módulos para tener una mejor evaluación del deportista.

Existe otro campo el cual podría también ser de utilidad el poder analizar y éste es el de la parte técnica y de resultados, en la cual se podría hacer un análisis sobre algunas variables ya específicas de los deportes, para esta parte quizá se pueda utilizar el mismo procedimiento implementado para el análisis de las pruebas de capacidad física.

Otra alternativa de crecimiento podría ser el crear para alguna institución que contara con varios equipos deportivos a los cuales puedan realizarles las pruebas, crear una base de datos en un servidor y hacer un esquema cliente servidor para poder hacer que más gente tenga posibilidades de utilizar el análisis de las pruebas de capacidad física en beneficio de uno o más equipos deportivos.

6.3.4. DISTRIBUCIÓN DEL SISTEMA

La distribución del sistema se llevará a cabo mediante la entrega los discos de instalación, o un CD, en el manual se incluye un apartado de instalación del sistema y ahí se describe el proceso de instalación..

El CD o los discos de instalación serán generados con Wise Installation System versión 4.0i, este es un programa que nos permite crear compactar los archivos que se van a instalar así como poder modificar algunas opciones de Windows. El código para generar el archivo de instalación se muestra en el apéndice B

Una vez instalado se dará un curso de capacitación en la utilización del sistema, realmente es un sistema muy fácil de emplear debido a que así fue diseñado para que gente no está involucrada en mucho con la computación lo pudiera utilizar, este curso sería de 8 horas.

Además del curso de capacitación en el sistema se entregará un manual el cual cubre toda la funcionalidad del sistema, y pueden utilizar para cualquier duda de utilización, el manual se muestra en el apéndice C

CONCLUSIONES

Para poder desarrollar el sistema que cumpliera con el objetivo planteado se utilizaron varios elementos que se describieron a lo largo del trabajo de los cuales se puede concluir lo siguiente.

El ciclo de vida del software orientado a objetos, nos indica que una gran parte del tiempo es gastada en la fase del diseño y una más pequeña en la implantación, este tiempo que se invierte en diseño se puede ver recompensado con la generación de código reutilizable, una estructura fácilmente extendible y mantenible.

Podemos decir que la programación estructurada difiere de la programación orientada a objetos en la forma de ver un mismo conocimiento. La programación estructurada está orientada a ver ¿cómo? se solucionarán las cosas, es decir, a los procedimientos.

La programación orientada a objetos se basa en ¿qué? hará un módulo, de esta forma la programación orientada a objetos modela los sistema como un conjunto de objetos que interactúan entre sí, y además puede estar directamente relacionado con la realidad.

Sin embargo se puede decir que la programación orientada a objetos está cimentada en la programación estructurada, ya que para descomponer un problema se necesita hacer estructuralmente para poder ver cómo se ordenarán los eventos. Es decir, la codificación de los métodos de un objeto serán codificados de forma estructural con sentencias bien estructuradas de bifurcación y control, en lugar de la sentencia GOTO.

Dentro de las ventajas que se tienen al utilizar la programación orientada a objetos podemos mencionar que el sistema es fácil de mantener, fácil de extender, es adaptable, permite la reutilización que implica menor tiempo de desarrollo, y se puede repartir el trabajo entre varios desarrolladores definiendo claramente los objetos.

Para realizar el desarrollo del sistema se siguió la metodología de Jacobson, podemos decir que uno de los conceptos más importantes de esta metodología, es el de los escenarios o casos de uso. Mediante los escenarios se pueden identificar todos los requerimientos de usuario, pero además durante toda la metodología se siguen utilizando a través de los diferentes modelos con lo que se consigue la rastreabilidad de los objetos a través de los diferentes modelos.

Se comprobó que esta metodología funciona, al menos para sistemas de características similares, además considero que es buena porque permite mediante los casos de uso modularizar al nivel que se quiera el desarrollo para tener un control y poder identificar que módulos son los más importantes, de esta forma poder llevar a cabo un desarrollo incremental de acuerdo a las necesidades principales. Sus diagramas son fáciles de entender y desarrollar, los diagramas están vinculados muy estrechamente ya que para avanzar al siguiente se necesita uno previo, con esto se hace fácil de seguir la metodología.

En la metodología se utilizan tres tipos de esquemas en el modelo de análisis para representar a los objetos:

- de "interfaz": corresponden a los objetos que interactúan con el mundo exterior al sistema,
- de "entidad": éstos modelan la información que debe permanecer en el sistema durante un tiempo determinado,
- de "control": éstos representan aquella funcionalidad que no está íntimamente ligada a ninguno de los otros dos tipos de objetos como control de secuencias,

de esta forma se obtiene un sistema más adaptable ya que por lo general la mayor parte de los cambios a los sistemas es en su funcionalidad y en su interfaz y mediante esta metodología se logran aislar muy bien estas partes de tal forma que los cambios sólo afectan a pocos objetos.

Esta metodología nos permite determinar de antemano el comportamiento que tendrán los objetos y la relación que tienen con los demás objetos antes de implementarlos, también se pueden obtener todos los beneficios mencionados anteriormente como ventajas de la programación orientada a objetos.

Se escogió el lenguaje Power Builder, ya que soporta conceptos de orientación a objetos como el de objeto, encapsulación, clase, polimorfismo y herencia.

Este lenguaje cumple con algunas de las características como familiaridad ya que es el que utilizo en mi trabajo, es para plataforma Windows, tiene presencia ya que si existen material de lectura y tutoriales, es posible el acceso a código existente, y tiene un buen ambiente de programación, cuenta con un depurador suficientemente bueno.

Como ya se mencionó, se desarrolló el sistema siguiendo la metodología mencionada anteriormente y para el caso específico de este trabajo podemos decir que se logró el objetivo que se planteó en un principio que fue automatizar un análisis estadístico para poder evaluar unas pruebas de capacidad física, dando así la posibilidad a la gente encargada de la preparación física de algún grupo de deportistas, el poder tener un panorama objetivo de la preparación física de su grupo y poder tomar decisiones objetivas para el trabajo físico posterior, lográndose reducir el tiempo del procesamiento de la información de forma por demás considerable.

Mediante este sistema se les da la posibilidad a hacer un análisis estadístico de un grupo de deportistas en aproximadamente 30 segundos independiente del número de personas a evaluar, lo cual es muy poco en comparación con el tiempo que se lleva hacer el proceso manualmente, que era aproximadamente de 5 días para hacer la evaluación de 60 personas. Además de que se pueden realizar cálculos de subgrupos de la misma información debido a los niveles descritos como agrupación, categoría, subcategoría, y también se puede hacer el análisis de un individuo, además de dar la posibilidad de almacenar una gran cantidad de información para poder reproducir cualquier cálculo en cualquier momento en el mismo tiempo.

El sistema fue desarrollado en ambiente Windows, por lo que da la posibilidad de que gente con conocimientos básicos o mínimos en computación lo pudieran operar, que era otra de las metas.

Podemos decir que se pueden tomar en cuenta los aspectos mencionados en el capítulo 6 en la parte de resultados para hacer mejoras posteriores al sistema.

BIBLIOGRAFÍA

- Timothy Budd, AN INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING, edición con correcciones,_ Addison-Wesley Publishing Company, Oregon State University, abril 1991.

- Ivar Jacobson /Magnus Christerson /Patrik Jonsson / Gunnar Övergaard, OBJECT-ORIENTED SOFTWARE ENGINEERING A USE CASE DRIVEN APRPROACH, 4th printing , Addison-Wesley, ESSEX Inglaterra,1998.

- Pressman S. Roger, INGENIERÍA DE SOFTWARE UN ENFOQUE PRÁCTICO, 3a edición, McGraw-Hill, España, 1993.

- Grady Booch, ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS CON APLICACIONES, 2ª edición en español, Addison-Wesley/Diaz de Santos 1996.

- Ernesto Peñaloza Romero, APUNTES, FUNDAMENTOS DE PROGRAMACIÓN, 2a edición, UNAM, ENEP Aragón, 1996.

- Herbert Schildt, TURBO C/C++ 3.1 MANUAL DE REFERENCIA, traducido de la 2a edición en inglés, McGraw-Hill, Inc., España, 1994.

- BUILDING OBJECT-ORIENTED APPLICATIONS IN POWER BUILDER STUDENT GUIDE, Sybase Inc, Powersoft, nov 1996.

- Smith M. and Tockey, S, AN INTEGRATED APPROACH TO SOFTWARE REQUIEREMENTS DEFINITION USING OBJECTS, Seattle WA, Boeing Commercial Airplane Support Division.

- R. Wirfs-Brock / B. Wilkerson / L. Wiener, DESIGNING OBJECT-ORIENTED SOFTWARE, 4th printing , Prentice-Hall, Englewood, NJ,1990.

- Valdés Rangel Francisco, MANUAL DE VALORACION DE LA CAPACIDAD FISICA PARA LA EDUCACION PRIMARIA, Edición de la DGENAMDF, México, 1996.

- Augusto Pila Teleña, EVALUACIÓN DE LA EDUCACIÓN FÍSICA Y LOS DEPORTES. Editorial Augusto Pila Teleña. Madrid, España, 1981.

- Avalos Falcon Eduardo, ESTADISTICA EDUCATIVA,Edición privada. ENSM, México 1978.

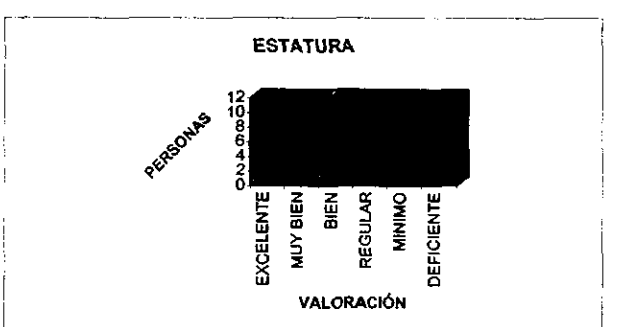
APENDICE A

Comparación de Resultados Obtenidos Manualmente y Mediante el Sistema

num_identificacion	estatura		
507			CM
505			1.78
521			cm
509			1.5
523			OSC
504			0.28
516			#intervalos
506			13
512			INT
515			0.021
511			C
510			0.540540541
503			C2
539			0.292184076
513			Ms
546			1.6435
544			M
501			1.654851351
518			S
514			0.073495069
540			1/2S
502			0.036747535
542			
520			
517			
530			
541			
545			
543			
522			
533			
535			
537			
534			
538	1.5		
532	1.5		
531	1.5		

1	2	6	12	72
2	4	5	20	100
3	3	4	12	48
4	4	3	12	36
5	5	2	10	20
6	0	1	0	0
7	4	0	0	0
8	3	-1	-3	3
9	5	-2	-10	20
10	1	-3	-3	9
11	3	-4	-12	48
12	0	-5	0	0
13	3	-6	-18	108

		VALORACIÓN		PORCENTAJE	FACTOR
1.728346421		EXCELENTE		21.62%	pos
1.691598886	1.728346421	MUY BIEN		13.51%	pos
1.618103817	1.691598886	BIEN		32.43%	pos
1.581356282	1.618103817	REGULAR		13.51%	neg
1.544608748	1.581356282	MINIMO		10.81%	neg
	1.544608748	DEFICIENTE		8.11%	neg
			37	100.00%	

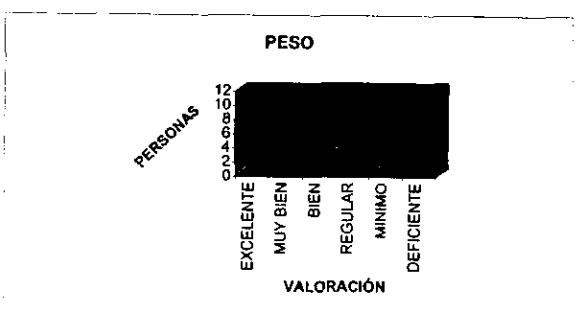


APENDICE A

num_identificacion	peso		CM
509			70
507			cm
523			44
544			OSC
513			26
515			#intervalos
510			13
506			INT
522			2
521			C
520			0.513513514
517	60		C2
516	60		0.263696129
511			Ms
512			55
518			M
543			56.02702703
501			S
505			6.71620262111843
545			1/2S
542			3.35810131055921
530			
504			
539			
502			
514			
503			
541			
538			
534			
532			
531			
537			
535			
540			
546			
533	44		

1		3	7	21	147
2		1	6	6	36
3		1	5	5	25
4		3	4	12	48
5		3	3	9	27
6		2	2	4	8
7		3	1	3	3
8		7	0	0	0
9		1	-1	-1	1
10		3	-2	-6	12
11		7	-3	-21	63
12		2	-4	-8	32
13	46	1	-5	-5	25

		VALORACIÓN	NUMEROS	PORCENTAJE	FACTOR
62.74322965		*EXCELENTE*		18.92%	pos
59.38512834	62.74322965	*MUY BIEN*		16.22%	pos
52.66892572	59.38512834	*BIEN*		29.73%	pos
49.31082441	52.66892572	*REGULAR*		24.32%	neg
45.9527231	49.31082441	*MINIMO*		8.11%	neg
	45.9527231	*DEFICIENTE*		2.70%	neg
			37	100.00%	



APENDICE A

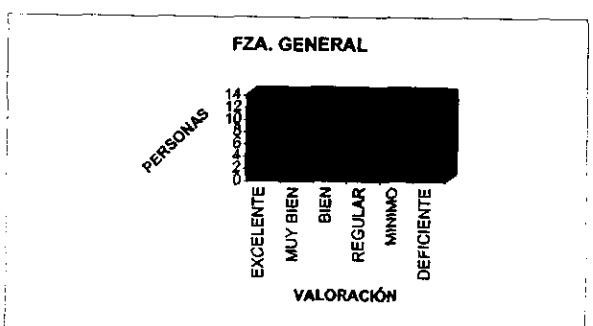
APENDICE A

num_identificacion	fza_general
518	
511	
502	
507	
521	
509	
501	
520	
515	
514	33
541	32
522	32
516	32
513	32
533	
517	
510	
512	
504	
542	
505	
503	
535	
531	
546	
545	
544	
506	
539	
534	
523	
530	
540	
543	
538	
537	8
532	6

CM	
54	
cm	
6	
OSC	
48	
#intervalos	
13	
INT	
3.692	
C	
0.594594595	
C2	
0.353542732	
Ms	
26.31	
M	
28.50524324	
S	
12.86011299476520	
1/2S	
6.43005649738260	

1	4	7	28	196
2	1	6	6	36
3	1	5	5	25
4	2	4	8	32
5	1	3	3	9
6	5	2	10	20
7	4	1	4	4
8	4	0	0	0
9	2	-1	-2	2
10	5	-2	-10	20
11	4	-3	-12	36
12	2	-4	-8	32
13	2	-5	-10	50

41.36535624		*EXCELENTE*		16.22%	pos
34.93529974	41.36535624	*MUY BIEN*		8.11%	pos
22.07518675	34.93529974	*BIEN*		37.84%	pos
15.64513025	22.07518675	*REGULAR*		21.62%	neg
9.215073751	15.64513025	*MINIMO*		10.81%	neg
	9.215073751	*DEFICIENTE*		5.41%	neg
			37	100.00%	

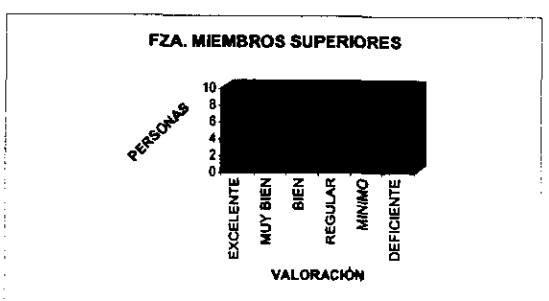


num_identificacion	fza_miembros_sup
502	
501	
503	
518	
511	
512	
507	
513	
521	
514	
516	30
509	30
520	27
522	
506	
531	
515	
504	
505	
543	
539	
523	
540	
517	
510	
532	
535	
534	
546	
533	
530	4
537	3
545	3
544	3
542	3
541	1
538	1

CM	49
cm	1
OSC	48
#intervalos	13
INT	3.692
C	2.459459459
C2	6.048940833
Ms	10.234
M	19.31432432432430
S	14.85739996542630
1/2S	7.42869998271315

1	1	10	10	100
2	4	9	36	324
3	2	8	16	128
4	1	7	7	49
5	2	6	12	72
6	3	5	15	75
7	2	4	8	32
8	0	3	0	0
9	2	2	4	8
10	2	1	2	2
11	6	0	0	0
12	5	-1	-5	5
13	7	-2	-14	28

		VALORACIÓN		PORCENTAJE	CONTROL
34.17172429		EXCELENTE		21.62%	pos
26.74302431	34.17172429	MUY BIEN		13.51%	pos
11.88562434	26.74302431	BIEN		16.92%	pos
4.456924359	11.88562434	REGULAR		27.03%	neg
-2.971775624	4.456924359	MINIMO		18.92%	neg
	-2.971775624	DEFICIENTE		0.00%	neg
			37	100.00%	



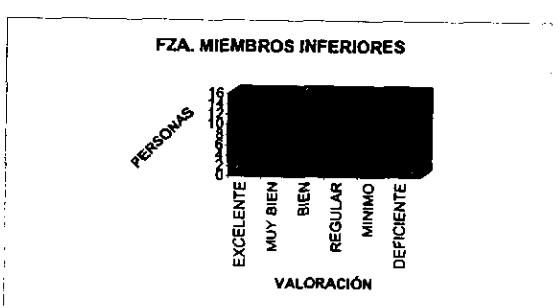
APENDICE A

num_identificacion	fza_miembros_inf	
521	[redacted]	
509		
523		
546		
516		
514		
545		
513		
507		
502		
542		
512		
520		
539		
522		
517		
537		
515		
532		
501		
518		
510		
503		
544		
543		
533		
511		
505		
540		
535		
534		
531		
530		
538		
541		
504		
506	[redacted]	

CM	105
cm	1
OSC	104
#intervalos	13
INT	8
C	0.567567568
C2	0.322132944
Ms	37
M	41.54054054054050
S	19.01719648
1/2S	9.50859823883070

1	[redacted]	2	8	16	128
2		0	7	0	0
3		0	6	0	0
4		0	5	0	0
5		1	4	4	16
6		0	3	0	0
7		7	2	14	28
8		8	1	8	8
9		5	0	0	0
10		9	-1	-9	9
11		4	-2	-8	16
12		0	-3	0	0
13	9	1	-4	-4	16
	[redacted]			[redacted]	

		VALORACIÓN	FRECUENCIA	PORCENTAJE	FACTOR
60.55773702		EXCELENTE	[redacted]	8.11%	pos
51.04913878	60.55773702	MUY BIEN		16.22%	pos
32.0319423	51.04913878	BIEN		43.24%	pos
22.52334406	32.0319423	REGULAR		21.62%	neg
13.01474582	22.52334406	MINIMO		8.11%	neg
	13.01474582	DEFICIENTE	[redacted]	2.70%	neg
			37	100.00%	



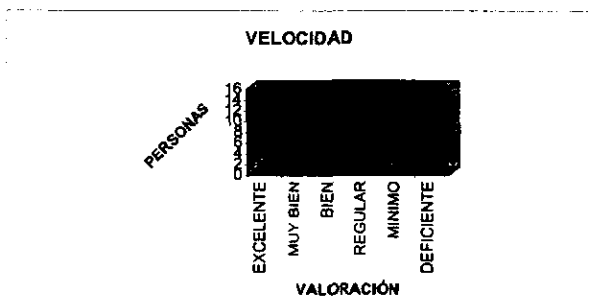
APENDICE A

num_identificacion	velocidad
506	
515	
516	
507	
542	
517	
513	
521	
511	
522	
509	
544	
514	
504	
541	
512	
510	
539	
520	
518	
502	
501	
503	
505	
545	
535	
523	
543	
538	
530	
531	
546	
537	
532	
534	
540	
533	-17.8

CM	-8.5
cm	-17.8
OSC	9.3
#intervalos	13
INT	0.715
C	-0.513513514
C2	0.263696129
Ms	-11.7175
M	-12.08466216216220
S	2.276992605
1/2S	1.138496303

1	1	4	4	16
2	1	3	3	9
3	14	2	28	56
4	3	1	3	3
5	4	0	0	0
6	2	-1	-2	2
7	3	-2	-6	12
8	1	-3	-3	9
9	1	-4	-4	16
10	4	-5	-20	100
11	0	-6	0	0
12	2	-7	-14	98
13	1	-8	-8	64

		VALORACIÓN		PORCENTAJE	FACTOR
-9.807669557		*EXCELENTE*		5.41%	pos
-10.94616586	-9.807669557	*MUY BIEN*		40.54%	pos
-13.22315846	-10.94616586	*BIEN*		27.03%	pos
-14.36165477	-13.22315846	*REGULAR*		8.11%	neg
-15.50015107	-14.36165477	*MINIMO*		10.81%	neg
	-15.50015107	*DEFICIENTE*		8.11%	neg
			37	100.00%	



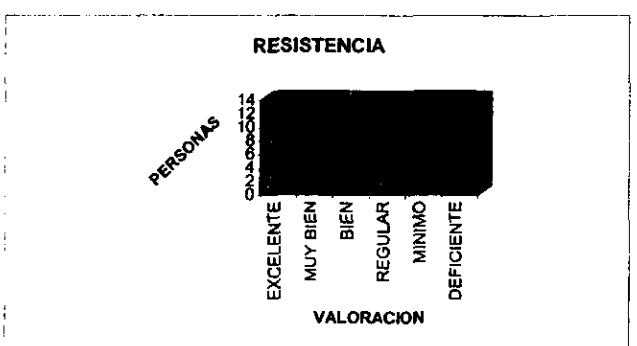
APENDICE A

num_identificacion	resistencia	
518		
506		
514		
521	1350	
509	1350	
522	1305	
512	1305	
507	1300	
516	1295	
513	1295	
505	1290	
517	1265	
503	1250	
502		
504		
530		
511		
520		
537		
533		
539		
545		
515		
510		
534		
531		
544		
535		
546		
543		
541		
538		
542		
540		
523		
501		
532	100	

CM	
2175	
cm	
100	
OSC	
2075	
#intervalos	
13	
INT	
159.615	
C	
-0.21621622	
C2	
0.04674945	
Ms	
1137.5025	
M	
1102.99115	
S	
301.789385	
1/2S	
150.894693	

1		1	6	6	36
2		0	5	0	0
3		0	4	0	0
4		1	3	3	9
5		1	2	2	4
6		10	1	10	10
7		7	0	0	0
8		10	-1	-10	10
9		5	-2	-10	20
10		1	-3	-3	9
11		0	-4	0	0
12		0	-5	0	0
13		1	-6	-6	36

1404.78053		*EXCELENTE*		8.11%	pos
1253.88584	1404.78053	*MUY BIEN*		24.32%	pos
952.096456	1253.88584	*BIEN*		37.84%	pos
801.201763	952.096456	*REGULAR*		18.92%	neg
650.30707	801.201763	*MINIMO*		8.11%	neg
	650.30707	*DEFICIENTE*		2.70%	neg
			37	100.00%	

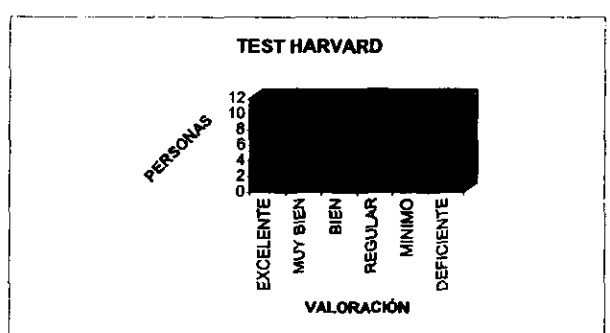


APENDICE A

num_identificacion	test_harvard	
523		CM
509		-80
517		cm
505		-200
521		OSC
518		120
514		#intervalos
512		13
511		INT
507		9.23
520		C
501		-1.135135135
522		C2
502		1.288531775
516		Ms
515		-121.535
513		M
510		-132.0122973
504		S
503		32.34326311
545		1/2S
506	12	16.17163155
530		
543		
540		
544		
546		
539		
537		
535		
534		
531		
533		
541	80	
538	40	
542	20	
532	-200	

1	3	4	12	48
2	1	3	3	9
3	7	2	14	28
4	2	1	2	2
5	8	0	0	0
6	1	-1	-1	1
7	2	-2	-4	8
8	2	-3	-6	18
9	6	-4	-24	96
10	0	-5	0	0
11	1	-6	-6	36
12	0	-7	0	0
13	4	-8	-32	256

-99.6690342		*EXCELENTE*		10.81%	pos
-115.840666	-99.6690342	*MUY BIEN*		24.32%	pos
-148.183929	-115.840666	*BIEN*		29.73%	pos
-164.35556	-148.183929	*REGULAR*		21.62%	neg
-180.527192	-164.35556	*MINIMO*		2.70%	neg
	-180.527192	*DEFICIENTE*		10.81%	neg
			37	100.00%	



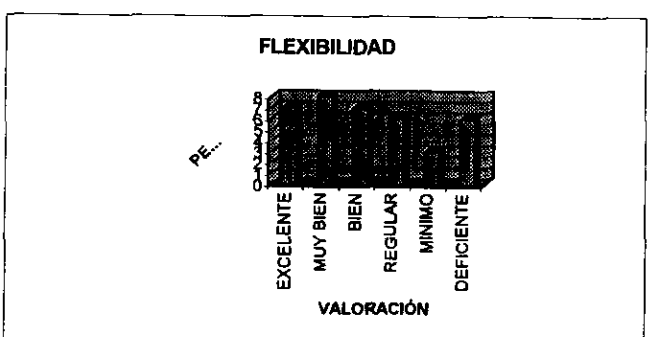
APENDICE A

num_identificacion	flexibilidad
511	0
541	
539	
535	
534	
545	
513	
538	
544	
543	
542	
516	
507	
522	12
515	12
514	11
520	10
518	10
502	
521	
546	
512	
537	
531	
530	
517	
540	
532	
501	
505	
503	2
533	1
523	1
510	1
506	1
504	1
509	0.9

CM	19
cm	0.9
OSC	18.1
#intervalos	13
INT	1.392
C	0.135135135
C2	0.018261505
Ms	8.56
M	8.748108108
S	5.108511984
1/2S	2.554255992

			E	D	FD	FD2
1			1	7	7	49
2			0	6	0	0
3			5	5	25	125
4			1	4	4	16
5			6	3	18	54
6	12.5	18.640	3	2	6	12
7			2	1	2	2
8			4	0	0	0
9			0	-1	0	0
10			4	-2	-8	16
11			2	-3	-6	18
12			2	-4	-8	32
13	2.298	0.904	7	-5	-35	175
			37 N		5 SFD	499 SFD2

		VALORACIÓN	# ALUMNOS	PORCENTAJE	FACTOR
13.8566201		*EXCELENTE*		18.92%	pos
11.3023641	13.8566201	*MUY BIEN*		21.62%	pos
6.19385212	11.3023641	*BIEN*		18.92%	pos
3.63959612	6.19385212	*REGULAR*		16.22%	neg
1.08534013	3.63959612	*MINIMO*		8.11%	neg
	1.08534013	*DEFICIENTE*	6	16.22%	neg
			37	100.00%	



APENDICE A

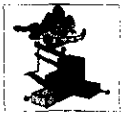


Tabla de Intervalos

1. ESTATURA	mayor que: 1.7283	1.EXCELENTE	8	21.6216
1. ESTATURA	mayor que: 1.6916 y, menor o igual que: 1.7283	2.MUY BIEN	5	13.5135
1. ESTATURA	mayor que: 1.6181 y, menor o igual que: 1.6916	3.BIEN	12	32.4324
1. ESTATURA	mayor que: 1.5814 y, menor o igual que: 1.6181	4.REGULAR	5	13.5135
1. ESTATURA	mayor que: 1.5446 y, menor o igual que: 1.5814	5.MINIMO	4	10.8108
1. ESTATURA	menor o igual que: 1.5446	6.DEFICIENTE	3	8.1081
TOTALES			37	100.0000
2. PESO	mayor que: 62.7432	1.EXCELENTE	7	18.9189
2. PESO	mayor que: 59.3851 y, menor o igual que: 62.7432	2.MUY BIEN	6	16.2162
2. PESO	mayor que: 52.6689 y, menor o igual que: 59.3851	3.BIEN	11	29.7297
2. PESO	mayor que: 49.3108 y, menor o igual que: 52.6689	4.REGULAR	9	24.3243
2. PESO	mayor que: 45.9527 y, menor o igual que: 49.3108	5.MINIMO	3	8.1081
2. PESO	menor o igual que: 45.9527	6.DEFICIENTE	1	2.7027
TOTALES			37	100.0000
3. FUERZA GENERAL	mayor que: 41.3654	1.EXCELENTE	6	16.2162
3. FUERZA GENERAL	mayor que: 34.9353 y, menor o igual que: 41.3654	2.MUY BIEN	3	8.1081
3. FUERZA GENERAL	mayor que: 22.0752 y, menor o igual que: 34.9353	3.BIEN	14	37.8378
3. FUERZA GENERAL	mayor que: 15.6451 y, menor o igual que: 22.0752	4.REGULAR	8	21.6216
3. FUERZA GENERAL	mayor que: 9.2151 y, menor o igual que: 15.6451	5.MINIMO	4	10.8108

Tabla de Intervalos

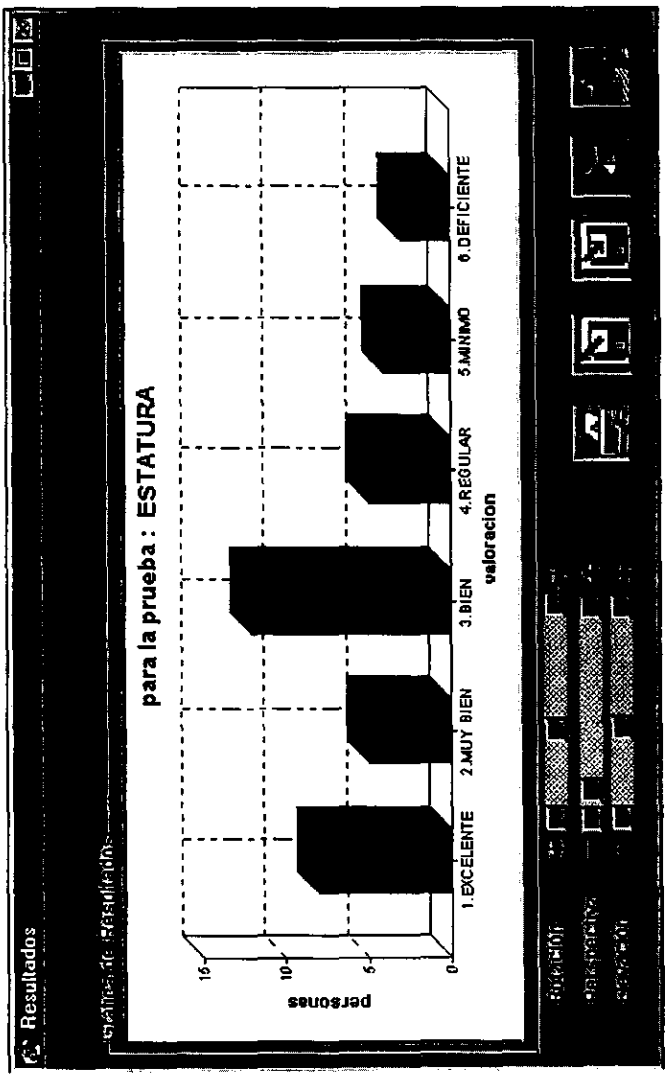
3. FUERZA GENERAL	menor o igual que: 9.2151	6.DEFICIENTE	2	5.4054
TOTALES :			37	100.0000
4. FZA. M. SUPERIORES	mayor que: 34.1717	1.EXCELENTE	8	21.6216
4. FZA. M. SUPERIORES	mayor que: 26.743 y, menor o igual que: 34.1717	2.MUY BIEN	5	13.5135
4. FZA. M. SUPERIORES	mayor que: 11.8856 y, menor o igual que: 26.743	3.BIEN	7	18.9189
4. FZA. M. SUPERIORES	mayor que: 4.4569 y, menor o igual que: 11.8856	4.REGULAR	10	27.0270
4. FZA. M. SUPERIORES	mayor que: -2.9718 y, menor o igual que: 4.4569	5.MINIMO	7	18.9189
4. FZA. M. SUPERIORES	menor o igual que: -2.9718	6.DEFICIENTE	0	0.0000
TOTALES :			37	100.0000
5. FZA. M. INFERIORES	mayor que: 60.5577	1.EXCELENTE	3	8.1081
5. FZA. M. INFERIORES	mayor que: 51.0491 y, menor o igual que: 60.5577	2.MUY BIEN	6	16.2162
5. FZA. M. INFERIORES	mayor que: 32.0319 y, menor o igual que: 51.0491	3.BIEN	16	43.2432
5. FZA. M. INFERIORES	mayor que: 22.5233 y, menor o igual que: 32.0319	4.REGULAR	8	21.6216
5. FZA. M. INFERIORES	mayor que: 13.0147 y, menor o igual que: 22.5233	5.MINIMO	3	8.1081
5. FZA. M. INFERIORES	menor o igual que: 13.0147	6.DEFICIENTE	1	2.7027
TOTALES :			37	100.0000
6. VELOCIDAD	menor que: 9.8077	1.EXCELENTE	2	5.4054
6. VELOCIDAD	mayor o igual que: 9.8077 y, menor que: 10.9462	2.MUY BIEN	15	40.5405
6. VELOCIDAD	mayor o igual que: 10.9462 y, menor que: 13.2232	3.BIEN	10	27.0270
6. VELOCIDAD	mayor o igual que: 13.2232 y, menor que: 14.3617	4.REGULAR	3	8.1081

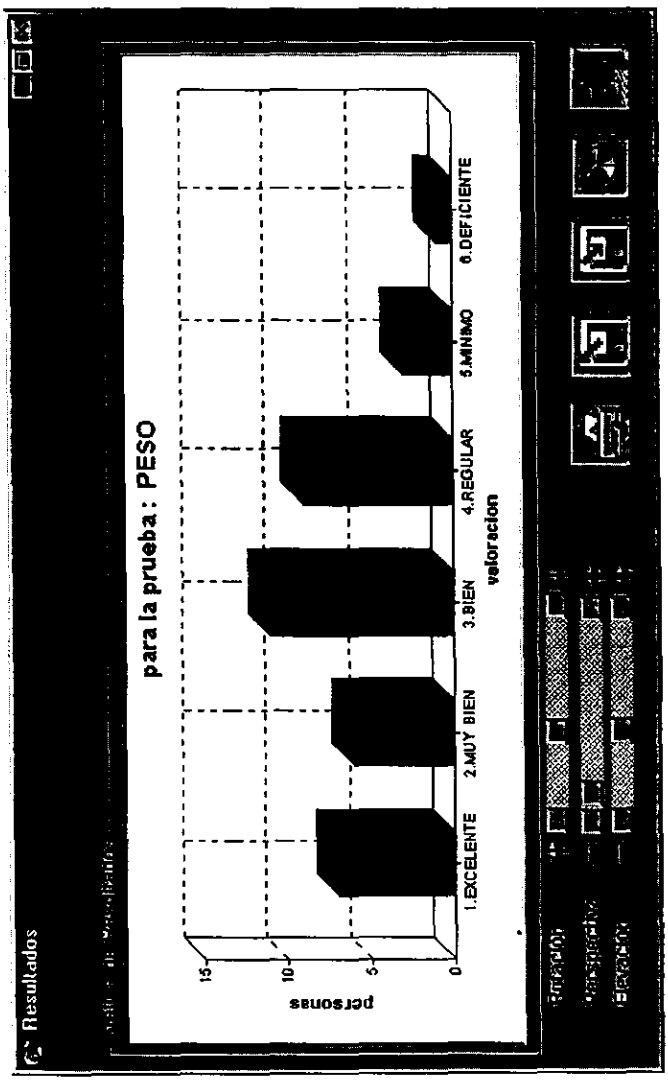
Tabla de Intervalos

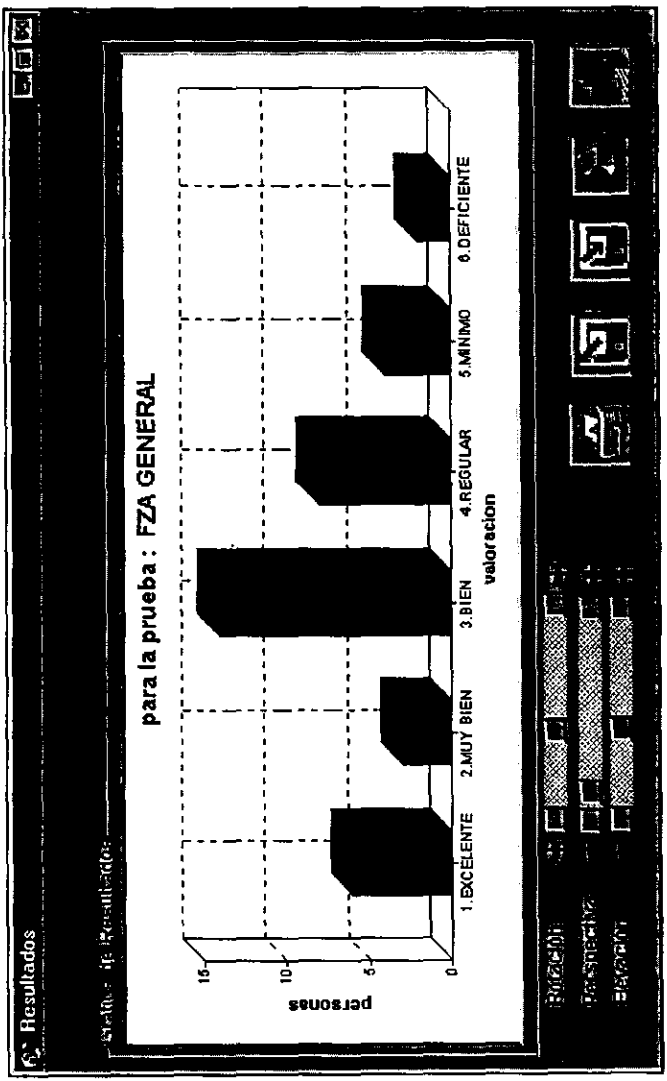
6. VELOCIDAD	mayor o igual que: 14.3617 y, menor que: 15.5002	5.MINIMO	4	10.8108
6. VELOCIDAD	mayor o igual que: 15.5002	6.DEFICIENTE	3	8.1081
TOTALES			37	100.0000
7. RESISTENCIA	mayor que: 1404.7805	1.EXCELENTE	3	8.1081
7. RESISTENCIA	mayor que: 1253.8858 y, menor o igual que: 1404.7805	2.MUY BIEN	9	24.3243
7. RESISTENCIA	mayor que: 952.0965 y, menor o igual que: 1253.8858	3.BIEN	14	37.8378
7. RESISTENCIA	mayor que: 801.2018 y, menor o igual que: 952.0965	4.REGULAR	7	18.9189
7. RESISTENCIA	mayor que: 650.3071 y, menor o igual que: 801.2018	5.MINIMO	3	8.1081
7. RESISTENCIA	menor o igual que: 650.3071	6.DEFICIENTE	1	2.7027
TOTALES			37	100.0000
8. TEST HARVARD	menor que: 99.669	1.EXCELENTE	4	10.8108
8. TEST HARVARD	mayor o igual que: 99.669 y, menor que: 115.8407	2.MUY BIEN	9	24.3243
8. TEST HARVARD	mayor o igual que: 115.8407 y, menor que: 148.1839	3.BIEN	11	29.7297
8. TEST HARVARD	mayor o igual que: 148.1839 y, menor que: 164.3556	4.REGULAR	8	21.6216
8. TEST HARVARD	mayor o igual que: 164.3556 y, menor que: 180.5272	5.MINIMO	1	2.7027
8. TEST HARVARD	mayor o igual que: 180.5272	6.DEFICIENTE	4	10.8108
TOTALES			37	100.0000
9. FLEXIBILIDAD	mayor que: 13.8566	1.EXCELENTE	7	18.9189
9. FLEXIBILIDAD	mayor que: 11.3024 y, menor o igual que: 13.8566	2.MUY BIEN	8	21.6216
9. FLEXIBILIDAD	mayor que: 6.1939 y, menor o igual que: 11.3024	3.BIEN	7	18.9189

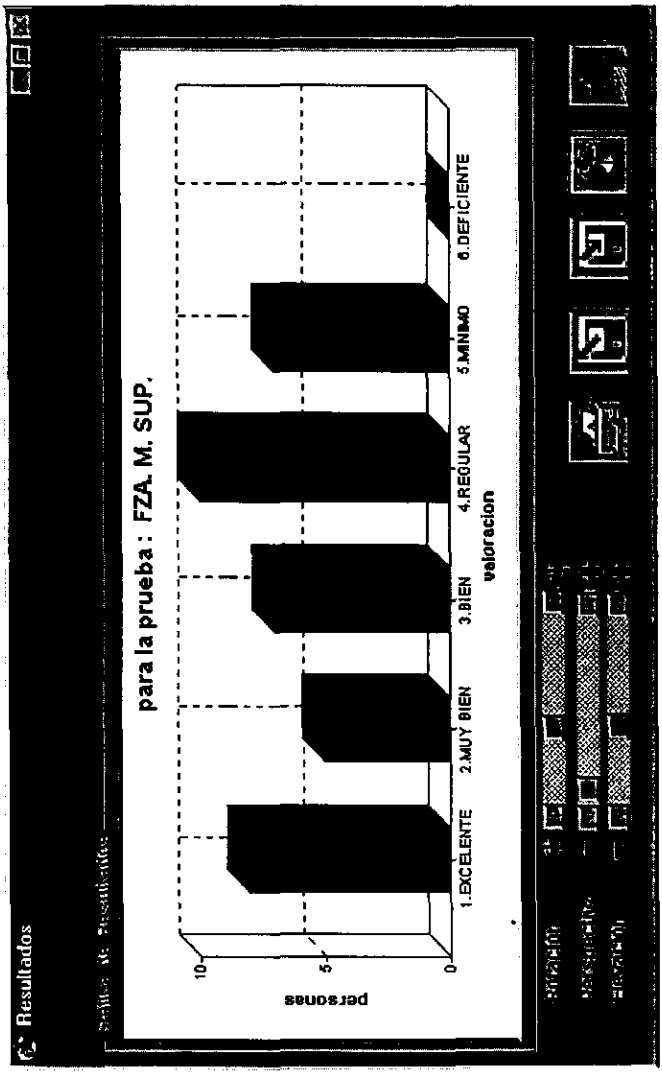
Tabla de Intervalos

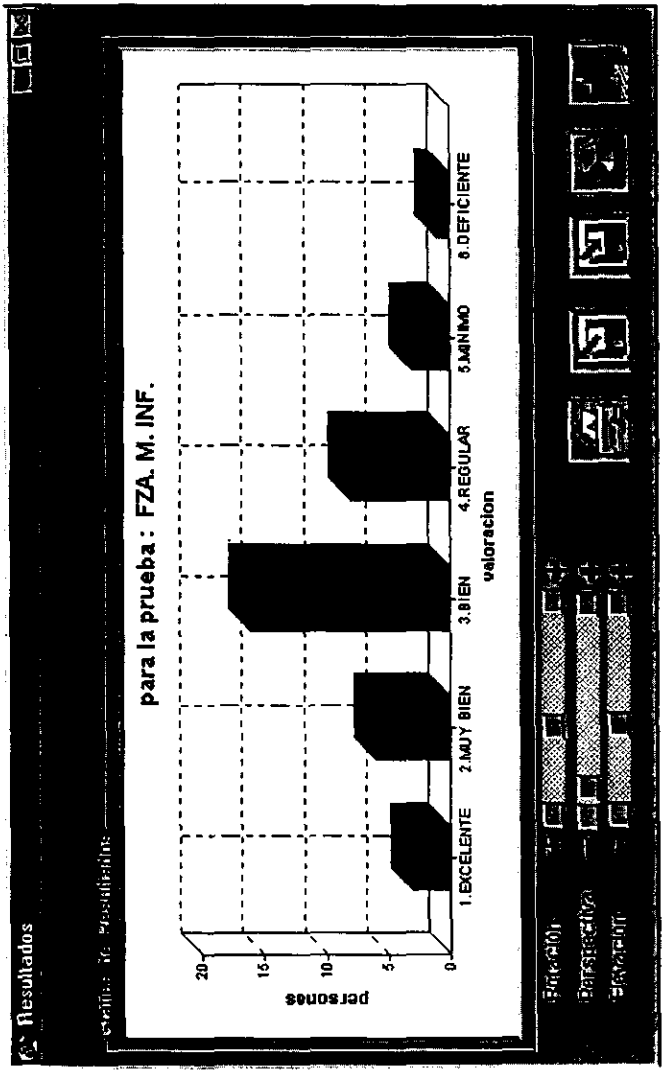
9. FLEXIBILIDAD	mayor que: 3.6396 y, menor o igual que: 6.1939	4.REGULAR	6	16.2162
9. FLEXIBILIDAD	mayor que: 1.0853 y, menor o igual que: 3.6396	5.MINIMO	3	8.1081
9. FLEXIBILIDAD	menor o igual que: 1.0853	6.DEFICIENTE	6	16.2162
TOTALES			37	100.0000

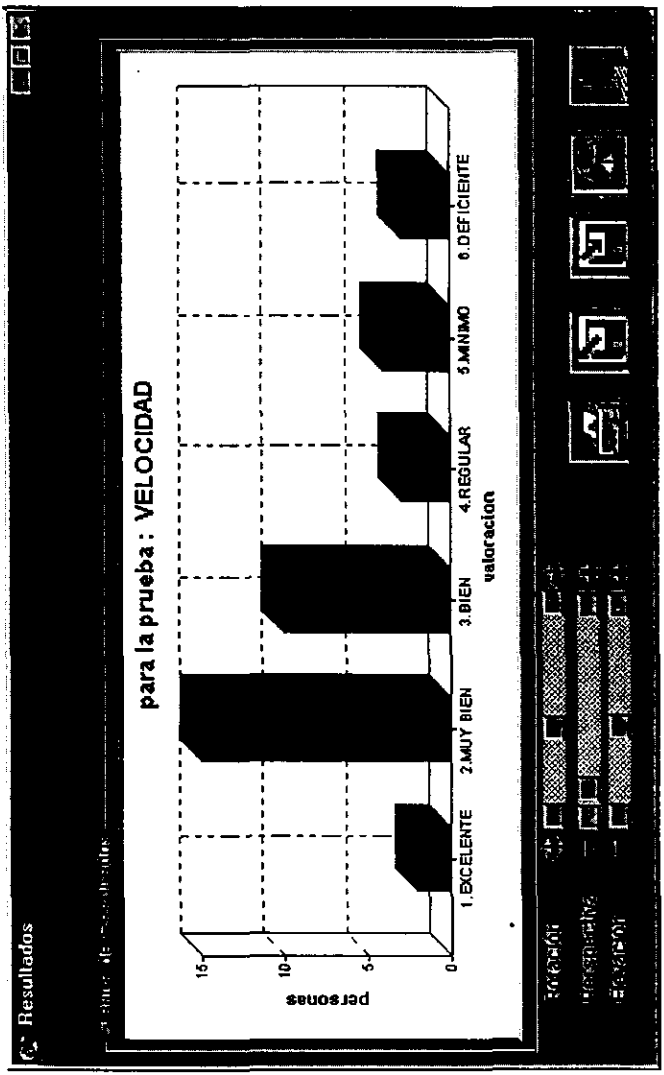

Resultados
para la prueba : ESTATURA
personas
15
10
5
0
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion

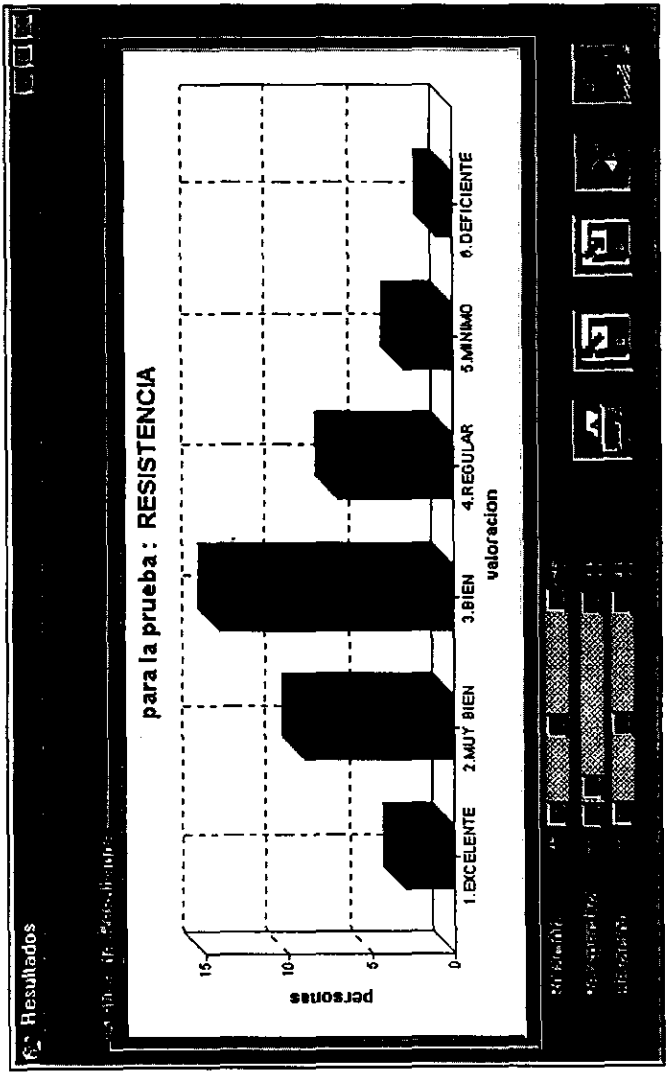

Resultados
para la prueba: PESO
personas
15
10
5
0
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion

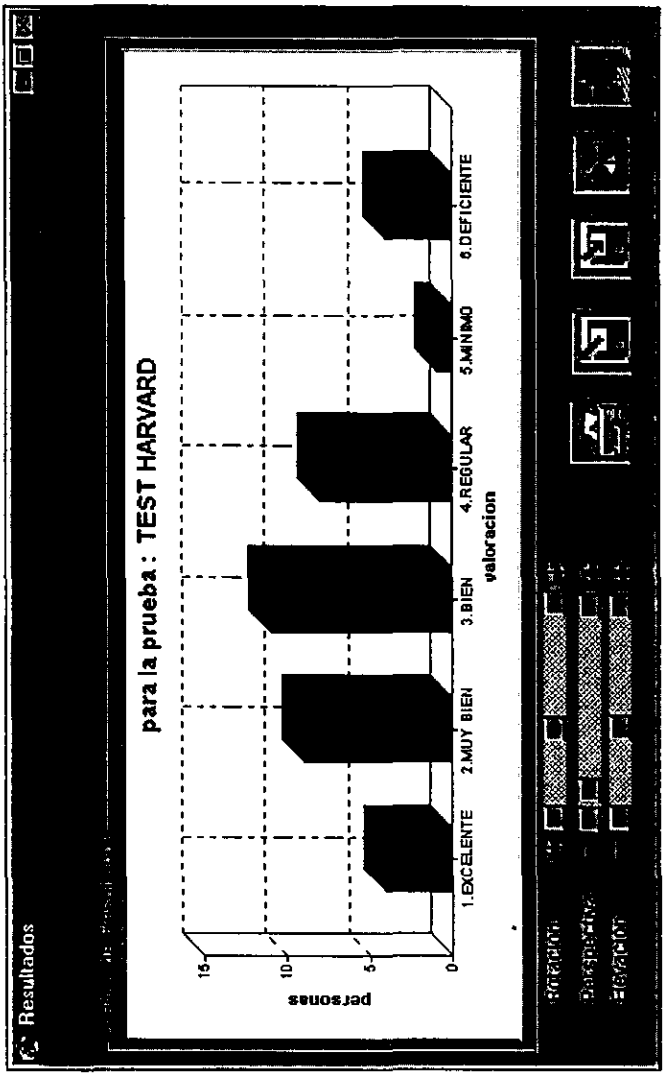

Resultados
para la prueba : FZA GENERAL
personas
15
10
5
0
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion

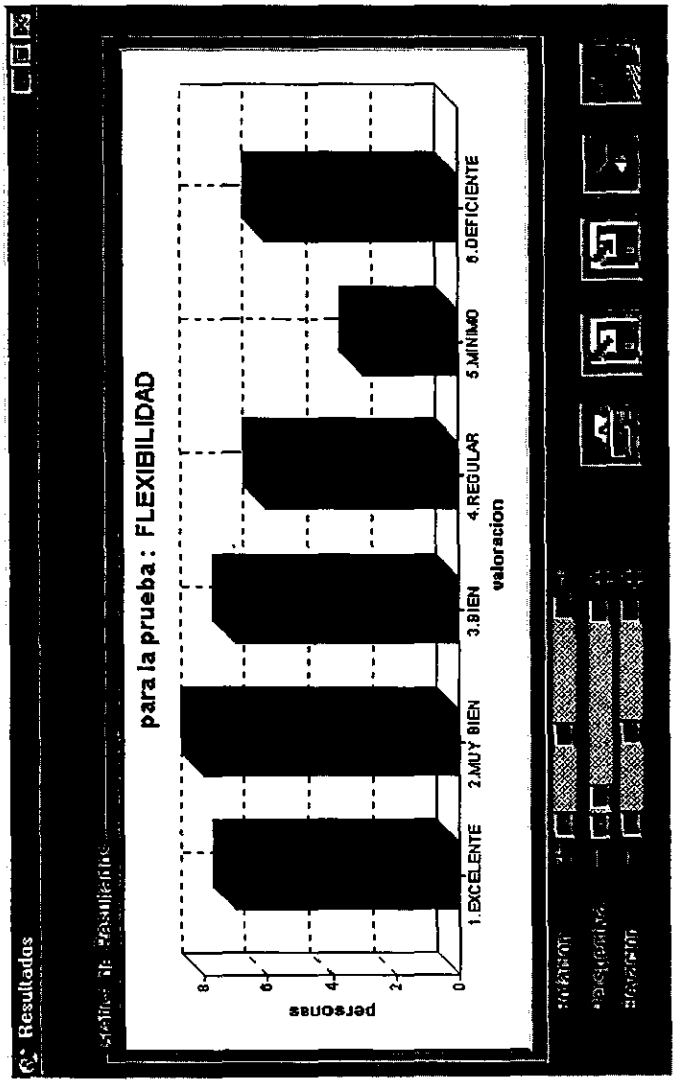

Resultados
para la prueba : FZA. M. SUP.
personas
10
5
0
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion


Resultados
para la prueba : FZA. M. INF.
personas
20
15
10
5
0
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion


Resultados
para la prueba : VELOCIDAD
personas
15
10
5
0
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion


Resultados
para la prueba: RESISTENCIA
personas
15
10
5
0
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion


Resultados
para la prueba : TEST HARVARD
personas
15
10
5
0
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion


Resultados
para la prueba: FLEXIBILIDAD
personas
0
2
4
6
8
1.EXCELENTE
2.MUY BIEN
3.BIEN
4.REGULAR
5.MINIMO
6.DEFICIENTE
valoracion

APENDICE B

Código de la Generación de Discos de Instalación

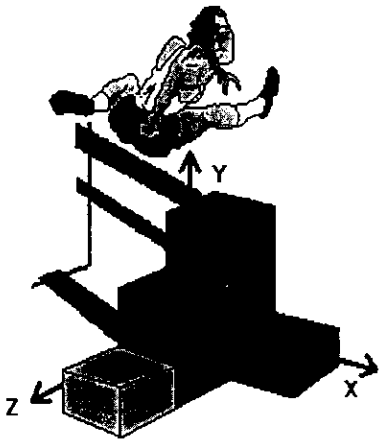
```
Display Graphic C:\Pces\intalacion\portadainstal.bmp (Scale)
X-Position: 0  Y-Position: 0
Display Message "Instalación de PCES "
Text: Inicio Instalación
Rem ************************Definición de variables
Check If Directory not writable %SYS% Start Block
   Set Variable SYS to %WIN%
End Block
Set Variable NUMBER to C:\Pces\Image\numeros
Set Variable IMAGE to C:\Pces\Image
Set Variable MAINDIR to C:\Pces
Set Variable WIN to C:\windows\system
Set Variable ESPACIO to 0
Check free disk space
Rem ****************Distribución de archivos
Install File C:\Pces\Pces.db to %MAINDIR%\Pces.db
Description: Base de datos del sistema
Install File C:\Pces\pces.ini to %MAINDIR%\pces.ini
Description: archivo de configuración del sistema
Install File C:\Pces\pces_dw.pbd to %MAINDIR%\pces_dw.pbd
Description: Librería del sistema
Install File C:\Pces\pces_uo.pbd to %MAINDIR%\pces_uo.pbd
Description: Librería del sistema
Install File C:\Pces\pces.exe to %MAINDIR%\pces.exe
Description: programa principal
Install File C:\Dscheme\Pcfg32.dll to %WIN%\Pcfg32.dll
Description: dll cfg
Rem ******************dll's de pb5.03
Install File C:\Dscheme\dllsist\pb5i32dk\Pbwss050.dll to %MAINDIR%\Pbwss050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbwsc050.dll to %MAINDIR%\Pbwsc050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbtyp050.dll to %MAINDIR%\Pbtyp050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbsmi050.dll to %MAINDIR%\Pbsmi050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbshr050.dll to %MAINDIR%\Pbshr050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbrtf050.dll to %MAINDIR%\Pbrtf050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbrte050.dll to %MAINDIR%\Pbrte050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbrtc050.dll to %MAINDIR%\Pbrtc050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbroi050.dll to %MAINDIR%\Pbroi050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbreg32.exe to %MAINDIR%\Pbreg32.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pboss050.dll to %MAINDIR%\Pboss050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbosc050.dll to %MAINDIR%\Pbosc050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbodb050.dll to %MAINDIR%\Pbodb050.dll
Description: dll sistema
```

```
Install File C:\Dscheme\dllsist\pb5i32dk\Pbnps050.dll to %MAINDIR%\Pbnps050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbnpc050.dll to %MAINDIR%\Pbnpc050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbitxt50.dll to %MAINDIR%\Pbitxt50.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbidbf50.dll to %MAINDIR%\Pbidbf50.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbhlp050.dll to %MAINDIR%\Pbhlp050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbdwe050.dll to %MAINDIR%\Pbdwe050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbdse050.dll to %MAINDIR%\Pbdse050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbdpb050.dll to %MAINDIR%\Pbdpb050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbbgr050.dll to %MAINDIR%\Pbbgr050.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\pb5i32dk\Pbcmp050.dll to %MAINDIR%\Pbcmp050.dll
Description: dll sistema
Rem ******************sqlany
Install File C:\Dscheme\dllsist\sqlany\Wodbc.hlp to %MAINDIR%\Wodbc.hlp
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Wtr50t.dll to %MAINDIR%\Wtr50t.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Wod50t.dll to %MAINDIR%\Wod50t.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Wl50ent.dll to %MAINDIR%\Wl50ent.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Sqlactrs.dll to %MAINDIR%\Sqlactrs.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Rtdsk50.exe to %MAINDIR%\Rtdsk50.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Isql.exe to %MAINDIR%\Isql.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbwrite.exe to %MAINDIR%\Dbwrite.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbvalid.exe to %MAINDIR%\Dbvalid.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbupgrad.exe to %MAINDIR%\Dbupgrad.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbunload.exe to %MAINDIR%\Dbunload.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbtran.exe to %MAINDIR%\Dbtran.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbtl50t.dll to %MAINDIR%\Dbtl50t.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbsvmn50.exe to %MAINDIR%\Dbsvmn50.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbstop.exe to %MAINDIR%\Dbstop.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbstart.exe to %MAINDIR%\Dbstart.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbshrink.exe to %MAINDIR%\Dbshrink.exe
Description: dll sistema
```

```
Install File C:\Dscheme\dllsist\sqlany\Dblog.exe to %MAINDIR%\Dblog.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbl50t.dll to %MAINDIR%\Dbl50t.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbinit.exe to %MAINDIR%\Dbinit.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbinfo.exe to %MAINDIR%\Dbinfo.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbextf50.dll to %MAINDIR%\Dbextf50.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbexpand.exe to %MAINDIR%\Dbexpand.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dberase.exe to %MAINDIR%\Dberase.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbeng50.exe to %MAINDIR%\Dbeng50.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbcollat.exe to %MAINDIR%\Dbcollat.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\sqlany\Dbbackup.exe to %MAINDIR%\Dbbackup.exe
Description: dll sistema
Rem ********************odbc
Rem ================= Se renombran los odbc de windows/system, si es que existen
Check If File exists %SYS%\odbc32.dll Start Block
   Rename %SYS%\odbc32.dll to odbc32.old
End Block
Check If File exists %SYS%\odbccp32.dll Start Block
   Rename %SYS%\odbccp32.dll to odbccp32.old
End Block
Check If File exists %SYS%\odbcint.dll Start Block
   Rename %SYS%\odbcint.dll to odbcint.old
End Block
Install File C:\Dscheme\dllsist\odbc\Odbcint.dll to %MAINDIR%\Odbcint.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\odbc\Odbctrac.dll to %MAINDIR%\Odbctrac.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\odbc\Odbcinst.hlp to %MAINDIR%\Odbcinst.hlp
Description: dll sistema
Install File C:\Dscheme\dllsist\odbc\Odbcinst.dll to %MAINDIR%\Odbcinst.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\odbc\Odbccr32.dll to %MAINDIR%\Odbccr32.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\odbc\Odbccp32.dll to %MAINDIR%\Odbccp32.dll
Description: dll sistema
Install File C:\Dscheme\dllsist\odbc\Odbcad32.exe to %MAINDIR%\Odbcad32.exe
Description: dll sistema
Install File C:\Dscheme\dllsist\odbc\Odbc32.dll to %MAINDIR%\Odbc32.dll
Description: dll sistema
Rem ================ Se actualiza el registro
If System Has Windows 95 Shell Interface Start Block
   Rem ================ Se registra SQL ANYWHERE
   Registry Key SOFTWARE\ODBC\ODBC.INI\Sybase SQL Anywhere 5.0 = %MAINDIR%\WOD5
   Registry Key SOFTWARE\ODBC\ODBCINST.INI\ODBC DRIVERS = Installed
   Registry Key SOFTWARE\ODBC\ODBCINST.INI\Sybase SQL Anywhere 5.0 = %MAINDIR%\
   Registry Key SOFTWARE\ODBC\ODBCINST.INI\Sybase SQL Anywhere 5.0 = %MAINDIR%\
```

```
End Block
Rem ******************crea grupo de programas
Get Registry Key Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folde
Set Variable GROUP to \SPORTSOFTWARE (Append)
Check If File or Directory doesn't exist %GROUP% Start Block
   Create Directory %GROUP%
End Block
Create Shortcut from %MAINDIR%\Pces.exe to %GROUP%\PCES.lnk
Rem ****************Modifica Registry
Registry Key SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\PCES.EXE = %MA
Registry Key SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\PCES.EXE = C:\
Rem ********************** Se registra la base de datos local en el registry
Registry Key SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources = Sybase SQL Anywhere 5.0
Registry Key SOFTWARE\ODBC\ODBC.INI\Pces = yes
Registry Key SOFTWARE\ODBC\ODBC.INI\Pces = C:\Pces\Pces.db
Registry Key SOFTWARE\ODBC\ODBC.INI\Pces = Pces
Registry Key SOFTWARE\ODBC\ODBC.INI\Pces = C:\Pces\WOD50T.DLL
Registry Key SOFTWARE\ODBC\ODBC.INI\Pces = sql
Registry Key SOFTWARE\ODBC\ODBC.INI\Pces = c:\Pces\dbeng50.exe
Registry Key SOFTWARE\ODBC\ODBC.INI\Pces = dba
Display Message "Instalación de PCES "
Text: Fin de Instalación
Custom Graphic
```

Manual de Usuario



Physical Capacity Evaluation System

Contenido

Capitulo I.
Introducción

En los tiempos actuales se ha creado la necesidad de tener un control sobre la información en todos los ámbitos. El deporte no es la excepción.

Para poder lograr un mejor resultado deportivo, tener mejores atletas, y estar mejor capacitados, es necesario medir y cuantizar, esto se hace estableciendo algo que medir, es decir algunas pruebas que sean aplicadas a un grupo de gentes para su posterior evaluación.

El "Physical Capacity - Evaluation System" ("PC-ES") realiza a grandes rasgos, un análisis estadístico de los resultados obtenidos de las pruebas, para poder tener una evaluación de alguna agrupación con el objetivo detectar deficiencias y poder atarcarlas a tiempo y efectivamente.

El "PC-ES" tiene una base de datos en la cual almacena la información necesaria para proporcionar un esquema robusto, consistente, y flexible de almacenamiento de información. En esencia se guardan en la base de datos: a un número de individuos pertenecientes a una o varias agrupaciones que a su vez se pueden dividir en categorías y subcategorías, de este modo tenemos que el "PC-ES" nos sirve, no sólo para una agrupación sino para varias simultáneamente.

También se almacenan los resultados de las pruebas obtenidas por los individuos pertenecientes a alguna agrupación en alguna categoría y/o subcategoría. Esto nos da la flexibilidad de almacenar varias pruebas realizadas por los individuos a lo largo del tiempo y a lo largo de su paso por las distintas categorías si es que pertenecen a una misma agrupación, o a través de su paso por distintas agrupaciones.

Cada individuo va a realizar las pruebas en un cierto momento, a este se le caracteriza de alguna manera, de tal forma que cada prueba realizada por un grupo de individuos en cierto momento pertenece a una caracterización, pudiendo tener bastantes momentos o caracterizaciones para realizar las pruebas, dando así la posibilidad de tener un registro de pruebas realizadas.

Con esta base de datos podemos entonces tener flexibilidad suficiente para tener un control sobre la información de la capacidad física de alguna agrupación.

El "PC-ES" está desarrollado para una plataforma de Windows 95, esto es está desarrollado de forma que lo pueda utilizar no un experto en sistemas sino cualquier gente con conocimientos básicos de computación. Cumple con las características de interfaz de Windows, es decir manejo de ventanas, *menús, botones, etc.*

Configuración

La configuración mínima necesaria para el sistema es una PC compatible con procesador PENTIUM, 8 MB de memoria, DK, velocidad de 100MHz, Monitor a color UVGA 800 x 600.

Capitulo II.
Entrada y salida del Sistema

La entrada al sistema se realiza mediante un login y un password, estos serán proporcionados cuando se instale la versión del "Physical Capacity – Evaluation System". Se permiten tres intentos de introducir la clave de acceso si son fallidos no se accede al sistema. Una vez adentro del sistema se puede cambiar el password y dar de alta otras claves de acceso que vayan a poder accesar a la información así como dar de baja las que no tengan porque verla. Esto se describe posteriormente cuando se expliquen las utilerías del "Physical Capacity – Evaluation System".. En la figura 2.1 se muestra la pantalla de presentación donde se pide la clave de acceso.

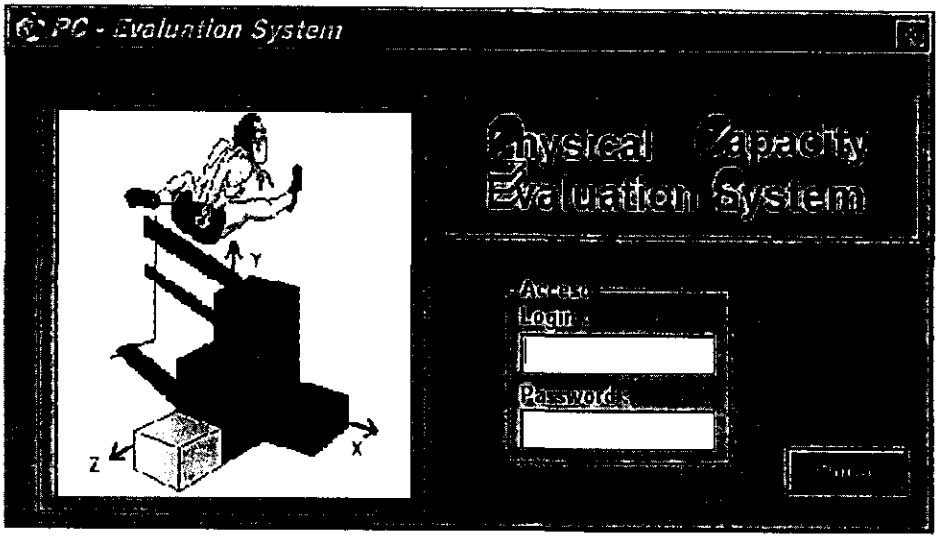


Figura 2.1. Pantalla de presentación PC-ES

Para salir del sistema se puede hacer mediante la opción del menú o bien a través de su ícono de acceso rápido, en la figura 2.2 se muestran las dos opciones.

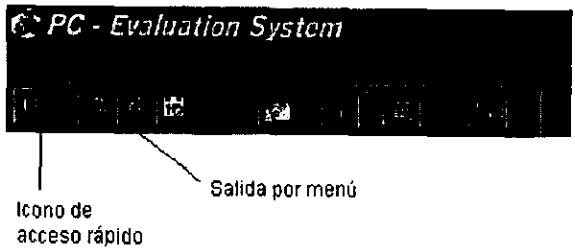


Figura 2.2. Salida del sistema

Capitulo III.
Captura de los Datos

Toda la captura se realiza a partir de el menú de captura. Para poder describir la captura de los datos es necesario plantear el esquema general de almacenamiento de datos. En el esquema del PC-ES tenemos que se almacenan separadamente los datos personales de los individuos, y las fechas de aplicación caracterizadas con una clave. De esta forma se pueden tener ***32000*** individuos diferentes almacenados. De esta misma manera se pueden tener ***m*** caracterizaciones de aplicaciones.

Estas dos formas de almacenamiento tienen forma de identificar individualmente a cada uno de los registros almacenados esto se denomina llave. La llave para donde se almacenan los individuos es un numero secuencial del 00001 hasta el 32000. En donde se almacenan las aplicaciones es la clave de caracterización. De este modo no puede haber registros repetidos.

Sin embargo hasta aquí nuestro esquema no está completo ya que nos falta tener un lugar donde almacenar los resultados obtenidos de las pruebas realizadas. Para dar origen a este sitio es necesario relacionar tanto las aplicaciones con los individuos, de esta forma tenemos que la llave de donde se almacenan los resultados de las pruebas es el número secuencial (de algún individuo) y la clave de aplicación (de alguna aplicación). Esto significa que cada registro que se almacena aquí, corresponde a un individuo x que realizó una prueba y en algún momento. Ver la figura 3.1.

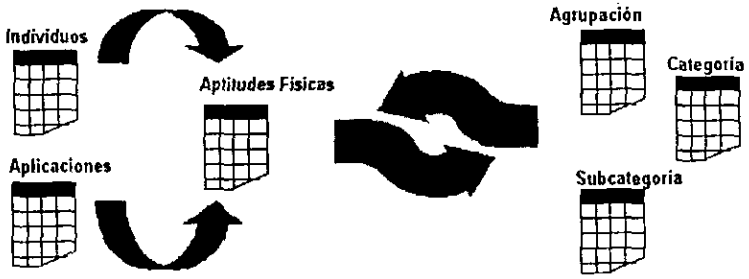


Figura 3.1. Esquema de almacenamiento de datos del PC-ES

Este esquema fue diseñado para que nos permita almacenar gran información, ya que de esta forma podemos almacenar varias pruebas realizadas por un mismo individuo a lo largo del tiempo relacionando al individuo con la aplicación correspondiente.

Existen algunos datos que son generales al sistema que nos sirven para hacer separaciones por grupos específicos de individuos y de las pruebas realizadas por ellos. Estos datos como se usan en varios lados se han establecido como catálogos. De esta forma uno puede definir en un catálogo una gran cantidad de datos que se vayan a utilizar y así en base a esa información obtener otra.

Los catálogos que se han definido son:

- Catálogo de agrupaciones
- Catálogo de categorías
- Catálogo de subcategorías.

Estos catálogos nos van dejando ve diferentes niveles de abstracción. Pueden existir ***32000*** agrupaciones, que pueden tener a su vez ***32000*** categorías y que también pueden tener ***32000*** subcategorías. La ventaja de manejar de esta forma la información es que así se puede ir agrupando la información tan generalmente o específicamente como se quiera.

Como se puede observar mucha de la clasificación de la información va a estar en función de los catálogos ya que nos interesa consultar información de una agrupación , categoría y subcategoría específicas. Por esta razón es indispensable que los catálogos tengan datos porque si no, no existe la forma de asociar individuos a una agrupación, categoría y subcategoría.

Catálogos

Como ya se expuso, los catálogos van a ser de uso general a lo largo del sistema y son forman parte de otra información, por esto se debe de introducir información primero en ellos para que a partir de eso poder introducir la demás información.

Como los catálogos son una parte importante se les tiene que dar un mantenimiento, esto consiste en mantenerlos actualizados. Por lo tanto existen opciones especiales para dar mantenimiento a los catálogos que consiste en dar altas, y bajas principalmente. A continuación se describe como dar mantenimiento a los catálogos.

En la figura 3.2 se muestra la opción en el menú para poder accesar a la pantalla de mantenimiento de los menús.

Figura. 3.2. Opción del menú para mantenimiento de los catálogos.

Una vez entrando a cualquiera de los catálogos se presenta la pantalla correspondiente, todas las pantallas tiene la misma funcionalidad sólo cambian los datos mostrados. Por esta razón se explicará sólo una.

La ventana muestra automáticamente los datos contenidos en el catálogo, por lo tanto la primera vez que se use el PC-ES estos estarán vacíos.

La ventana se muestra en la figura 3.3. En la ventana existen cinco botones de izquierda a derecha son :

- Botón de impresión
- Botón de consulta
- Botón de inserción
- Botón de borrado
- Botón de salir

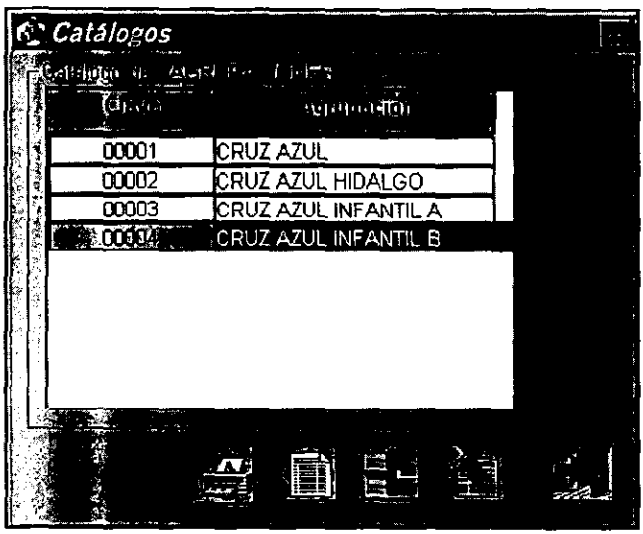


Figura 3.3. Pantalla de mantenimiento de Catálogos

El botón de impresión nos sirve para imprimir el catálogo si se requiere. El botón de consulta se utiliza para refrescar la pantalla de consulta podría usarse cuando se hace una baja o una alta en el catálogo, para estar seguros de que ya existe en la base de datos. El botón de salir cierra la ventana.

La ventana de mantenimiento de catálogos tiene algunas características, como algunas otras del PC-ES. Si se quiere seleccionar un registro se le da doble click con el mouse. Si un registro está seleccionado y se selecciona otro se quitará lo seleccionado del primero y si se quiere quitar lo seleccionado a un registro que esté seleccionado se le da doble click con el mouse. Otra característica es que se puede arrastrar la ventana de consulta hacia los botones y es el mismo efecto que si los seleccionáramos. Para arrastrar la

ventana se tiene que apretar el botón derecho del mouse sobre el cuadro de afuera de la ventana y aparecerá un ícono que nos indica que se está arrastrando la información contenida en la ventana. Ver la figura 3.6.

Se dejaron dos botones porque su explicación requiere más detalle. Para la inserción de un nuevo registro al catálogo se presiona el botón correspondiente o se arrastra la ventana de información hacia el. Se insertará un registro vacío y estará seleccionado con el mouse se debe de presionar en la casilla de la clave introducir el número consecutivo siguiente (es recomendable) u otro que no exista, el formato de los números se dá automáticamente no es necesario introducir un "00005" sólo "5".Con el mouse o con la tecla tab se puede pasar a la siguiente casilla e introducir el dato correspondiente ya sea agrupación, categoría o subcategoría si ya está correcto se presiona la tecla enter y se guarda automaticamente, mostrándose en seguida en la ventana de información, si es que no hubo un error. Si este es el caso aparece una ventana de error con el mensaje correspondiente. Este proceso se muestra en la figura 3.4.

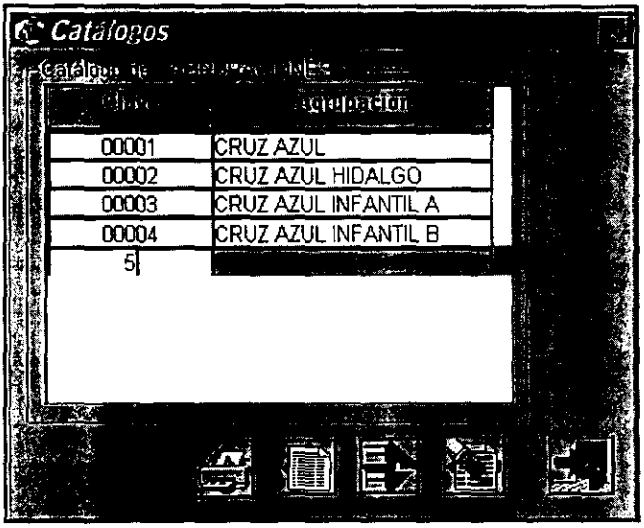


Figura 3.4. Proceso de inserción en un Catálogo.

Para el proceso de borrado de un registro se usa el botón de borrado, a la función proporcionada por este se puede accesar de las dos formas mencionadas anteriormente. Es necesario que se encuentre un registro seleccionado si no se borrará ningún registro y se mandará un mensaje de error (Ver la figura 3.6). Cuando se ha seleccionado un registro y se utiliza el botón aparece un mensaje de confirmación (ver la figura 3.7). En las figuras siguientes se muestra el proceso de borrado de un registro

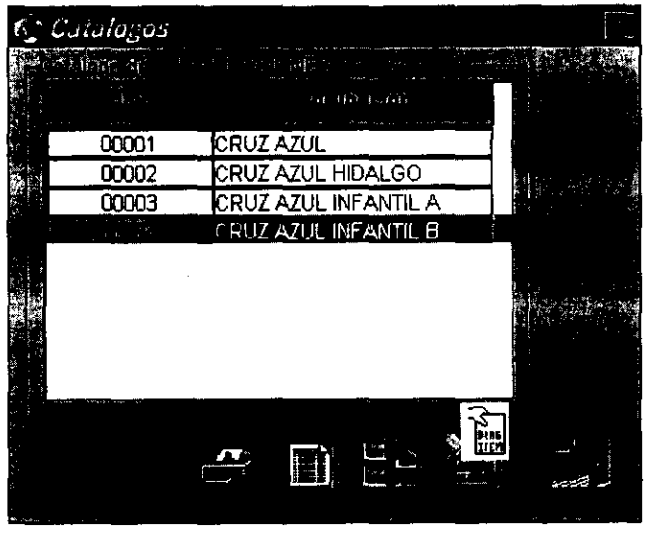


Figura 3.5. Borrado de un registro

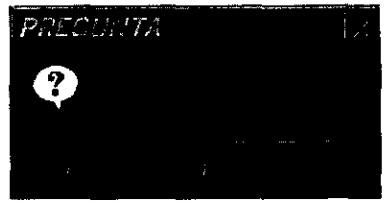


Figura 3.6. Mensaje de confirmación.

Figura 3.7. Mensaje cuando no se ha seleccionado ningún registro.

Si se quiere borrar un registro que se esté utilizando par la demás información se mostrará un mensaje como el siguiente y no se borrará.

Figura 3.6 Mensaje que se muestra al borrar un registro que se está utilizando

El botón de salir es el único que no acepta que se arrastre la información ya que no tiene sentido. Por lo tanto para salir de la ventana de mantenimiento de catálogos se necesita presionar el botón de salir.

Captura de Individuos

Una vez que se tiene capturada la información en los catálogos, implica que ya se tienen definidas las agrupaciones , categorías y subcategorías en las cuales se van a clasificar a los individuos. Es entonces cuando podemos hacer la captura de los datos personales.

Estos datos nos van a servir para identificar a un individuo de los demás, como ya se mencionó cada individuo va a tener una clave que va a ser

un número con el cual se le identificará a lo largo del sistema. La pantalla de captura de los datos personales (ver la figura 3.7) tiene dos botones uno de los cuales es el de aceptar y otro el de salir.

La pantalla de captura está dividida en dos partes la primera es correspondiente a los datos personales del individuo, está del lado izquierdo. La segunda correspondiente a los datos de la agrupación a la que pertenece el individuo, esta del lado derecho. En ésta última se muestra la información que se introdujo previamente en los catálogos y de la cual se tiene que seleccionar la correcta. Se presenta en cajas de texto desplegables.

Aquí es necesario hacer notar que la agrupación, categoría y subcategoría que se le asignan a un individuo son en un cierto momento, y que pueden ser modificadas a lo largo del tiempo.

Para guardar la información introducida en la pantalla de captura se debe de presionar el botón de aceptar. Cuando se presiona se realizan algunas validaciones antes de guardar la información como que no exista la clave asignada y que se tengan los campos mínimos para poder guardarlos. Estas validaciones se hacen en todas las pantallas de captura.

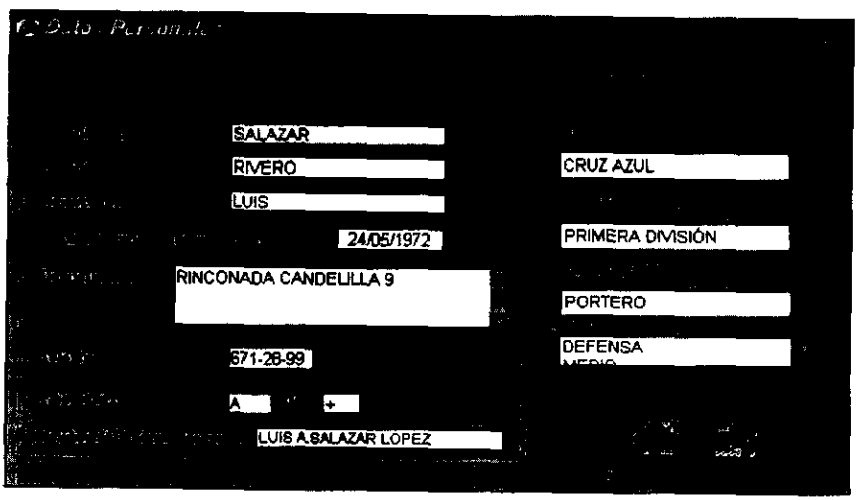


Figura 3.7. Pantalla de captura de Individuos.

Los campos mínimos para aceptar un individuo son :

- Numero de Identificación (Clave)
- Apellido Paterno
- Apellido Materno
- Nombre
- Fecha Nacimiento

- Agrupación
- Categoría
- Subcategoría

En los últimos tres si no se seleccionan se asume que es el que aparece en la caja de texto. Nunca deberán estar vacías las cajas de texto ya que previamente se deberán llenar los catálogos Si falta alguno de los primeros cinco aparecerá un mensaje como el que sigue indicando cual o cuales faltan.

Figura 3.8. Mensaje de error si faltan datos.

Si el número que se quiere introducir ya ha sido asignado previamente también aparece un mensaje de error indicándolo. Ver la figura 3.9

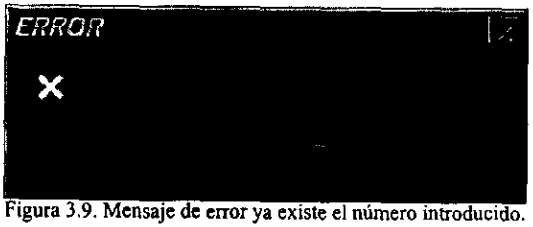


Figura 3.9. Mensaje de error ya existe el número introducido.

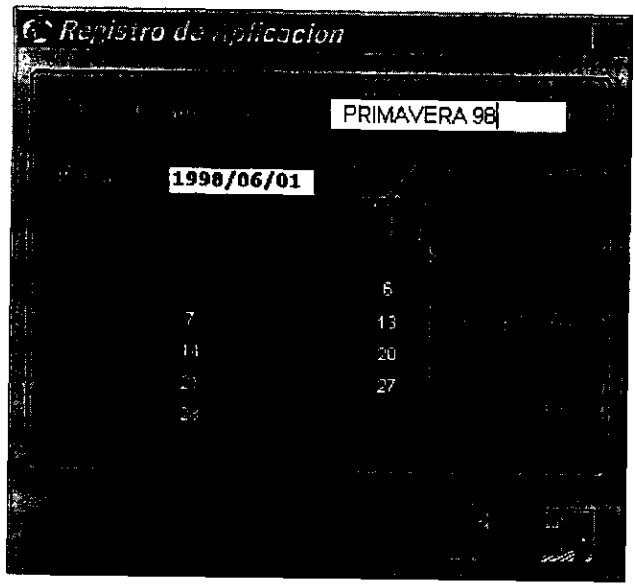
Si se logran introducir los datos correctamente se limpiará la pantalla dando la posibilidad de introducir algún individuo más. De esta forma se seguirá operando hasta que se presione el botón de salir. Las demás pantallas de captura operan de la misma manera, es decir, se salen hasta que se presiona el botón de salir y mientras no se presione se permite seguir capturando de forma continua los elementos.

Las diferentes cajas de texto tiene un orden de llenado que se da por defoult, para seguir este orden es necesario que al terminar de llenar alguna de las cajas de texto se presione la tecla tab, si se quiere llenar en algún orden especial o alguna caja de texto para corregirla se debe de seleccionar DICHA caja con el mouse y llenar o cambiar el contenido.

Captura de Fechas de Aplicación

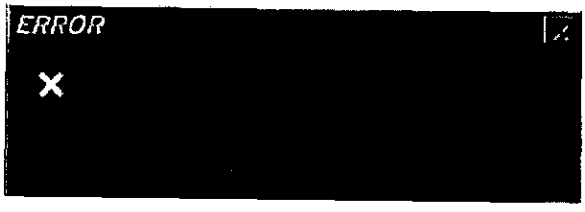
Para realizar la captura de una fecha de alguna aplicación es necesario introducir dos datos y los dos son obligatorios, el primero es una clave alfanumérica que va a identificar a una aplicación de pruebas específica. El otro dato es una fecha, para proporcionar la fecha se muestra un calendario en el cual se puede seleccionar con el mouse la fecha de la aplicación. Para cada aplicación debe de existir sólo una fecha en la cual se realizó aunque los datos no se capturen en su totalidad en esa fecha. La fecha es para referir el tiempo en que se realizaron las pruebas y no cuando se introdujeron los datos al sistema.

Es importante decir que si no se selecciona ninguna fecha se pone automáticamente la que se muestra , (la fecha de la computadora). En el calendario se pueden elegir fechas de diferentes meses y años, Para avanzar y/o retroceder en años se presionan la tecla de la flecha externa (grande) que esta en el calendario, si se quiere avanzar y/o retroceder en meses se presiona la tecla de la fecha interna (pequeña). Para estas teclas para la derecha se avanza y para la izquierda se retrocede. Esto se esquematiza en la figura 3.10.



3.10. Manejo de calendario.

Esta ventana como las otras de captura tiene dos botones uno de aceptar y uno de salir, al igual que las otras ventana si se acepta se valida que se haya llenado la caja de la clave de aplicación y permite seguir capturando fechas de aplicación, si no se ha capturado la clave de la aplicación manda un error como el siguiente.



3.11. Error si no se ha introducido la clave de la aplicación.

La fecha siempre es correcta y como ya se mencionó si no se selecciona se pone la de la fecha de la máquina. El botón de salir cierra la ventana

Captura de Pruebas Físicas

La pantalla de captura de pruebas físicas opera de manera similar a la de captura de individuos, esta pantalla tiene tres secciones principales una donde se define la llave para guardar los datos, esta se localiza en la parte superior derecha, aquí se encuentran dos cajas de texto donde se pide el número de identificación de un individuo y la clave de la aplicación a la cual se van a asociar esos valores de las pruebas. Estos datos son muy importantes ya que son los que nos dan la posibilidad de vincular a las fechas de aplicación con los individuos y generando de esta forma un único registro de resultados de pruebas para esa llave. La calve de la aplicación se debe de seleccionar la correcta ya que en la caja de texto se muestran de forma desplegable todas las existentes. Si no se selecciona alguna se pone la que esté mostrada.

La otra sección se encuentra debajo de la primera, aquí se definen los datos correspondientes a la agrupación, categoría y subcategoría correspondiente a un individuo, en estas cajas de texto se muestran los datos correspondientes a un individuo capturados en la pantalla de datos personales, sin embargo se pueden modificar por cualquiera de los valores existentes que se muestran al desplegar las cajas de texto.

La tercera sección se encuentra del lado izquierdo, esta sección se refiere a los resultados obtenidos en las pruebas de valoración y a su vez se divide en dos. La parte de arriba es para capturar los resultados de las pruebas y la de abajo para algunos comentarios con respecto a ese individuo y a esa fecha de aplicación.

De esta forma tenemos una pantalla con tres secciones y de esas secciones se tienen algunos datos obligatorios como en las otras pantallas de captura. Los datos obligatorios son :

- Número de Identificación
- Clave de Aplicación

- Estatura
- Peso
- Fuerza General
- Fuerza Miembros Superiores
- Fuerza Miembros Inferiores
- Velocidad
- Resistencia
- Test de Harvard
- Flexibilidad

- Agrupación
- Categoría
- Subcategoría

Los últimos tres datos así como en la captura de individuos existen llenos con un valor por defoult, que es el correspondiente a los datos personales capturados previamente. Por lo tanto siempre son válidos. Sin embargo para los valores de los resultados de las pruebas si es necesario que se introduzcan todos si no cuando se selecciona el botón de aceptar se marca un error, indicando cuales son los que faltan, como el siguiente:

3.11. Error indicando la introducción incorrecta de datos

Si existiera un error al introducir el numero de identificación se marcaría un error como el siguiente.

3.12. Error al introducir el número de identificación

Los dos botones son los mismos a las otras pantallas de captura (aceptar y salir) por lo tanto su funcionamiento es igual.

Las pantallas de captura de datos personales de individuos, pruebas físicas y fechas de aplicación pueden ser accionadas también con unos íconos de acceso rápido que se encuentran en el menú. Esto además de la forma tradicional a través del menú en cascada que ya se mostró anteriormente en la figura 3.2. Los íconos de acceso rápido se muestran a continuación :

3.13. Iconos de acceso rápido (individuos, pruebas físicas, fechas de aplicación)

Capitulo IV.
Procesamiento de los Datos

Uno de los objetivos del "PC-ES" es poder dar la flexibilidad para realizar cálculos estadísticos sobre los resultados de las pruebas de valoración de un conjunto de individuos seleccionados por el usuario de forma sencilla.

Esto nos da la posibilidad de ir analizando resultados tan específicamente como se requiera, se puede ir desde lo más general (una agrupación) hasta lo más particular (un individuo).

Para que se pueda lograr el que el usuario seleccione los cálculos a realizar deseados de forma sencilla se dividieron en dos tipos :

- Conjunto
- Individuales

Para poder realizar los cálculos es necesario accesar a las opciones correspondientes del menú estas se muestran en la figura siguiente :

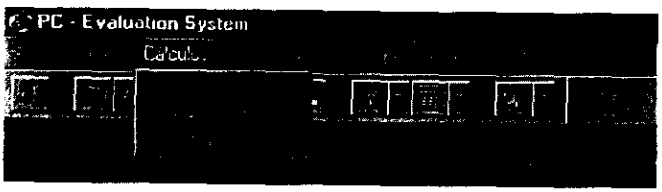


Figura 4.1. Acceso a las opciones para realizar cálculos.

También como son opciones muy frecuentes tienen sus íconos de acceso rápido. Estos se muestran en la figura 4.2.

Figura 4.2. 10conos de acceso rápido (conjunto, individuales, comparativos)

Cálculos en Conjunto

Los cálculos en conjunto nos van a permitir obtener resultados a varios niveles estos pueden ser :

- Agrupación
- Categoría
- Subcategoría
- Clave de Aplicación

Sin embargo puede seleccionarse un conjunto combinado con los niveles mencionados, por ejemplo una agrupación y una subcategoría, una agrupación y una fecha de aplicación, etc.

En la pantalla de captura se nos permite hacer de manera fácil la selección. Si se quiere inhibir algún nivel posible de seleccionar simplemente se marca el cuadro que tiene al lado derecho, esto hace que no se tome en cuenta ese nivel para la selección del grupo. Todos los niveles que no se encuentren deshabilitados *se tomarán en cuenta* en la selección con el valor que tengan. Los valores posibles de los niveles son mostrados cuando se selecciona la caja de texto desplegable y éstos van a corresponder a datos introducidos previamente. Ver figura 4.3.

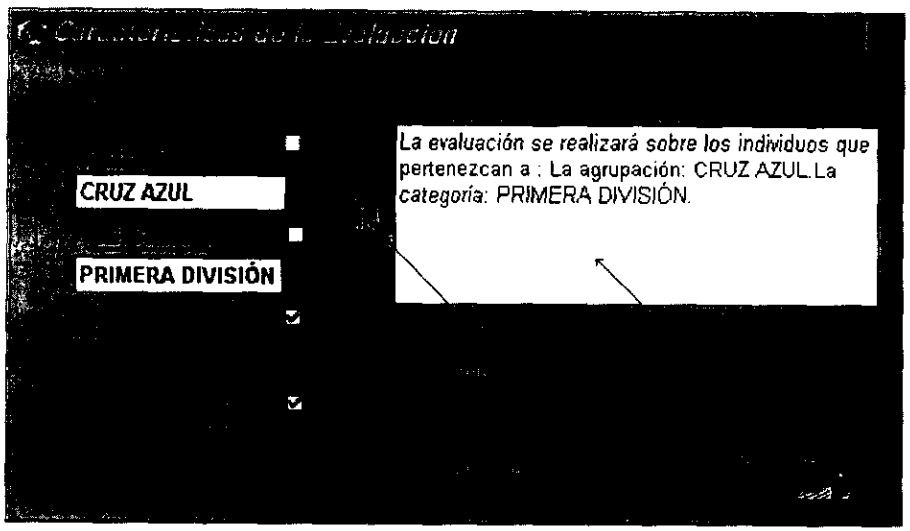
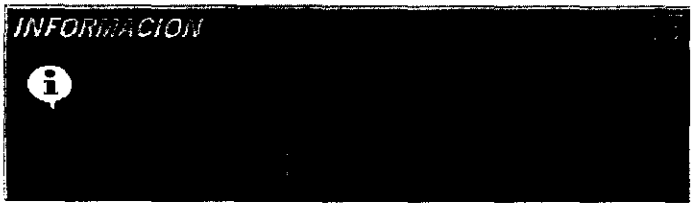


Figura 4.3. Pantalla de cálculos en conjunto.

En la parte derecha de la pantalla se encuentra una caja de texto donde se puede arrastrar la información de los distintos niveles (el arrastrado de información se lleva a cabo igual que en los catálogos) y se dará una descripción textual del conjunto elegido sobre el cual se desarrollarán los cálculos. Esto sirve para cuando no se está seguro del conjunto o para revisar antes de ejecutar los cálculos. Si se está seguro de la información a procesar se presiona el botón de aceptar, la caja de texto del lado derecho se llena con los datos correspondientes y se empiezan los cálculos.

Cuando se presiona el botón de aceptar se hace una validación muy importante, se checa que existan para las condiciones seleccionadas del cálculo que existan al menos cuatro registros sobre los cuales se va a realizar el cálculo, esto es porque si no serían muy pocos registros y mientras más registros se tengan los resultados son más representativos. Si no se cumpliera con esta restricción se manda un mensaje como el siguiente :



4.4. Mensaje que se manda cuando no hay suficientes registros para hacer el cálculo

El porcentaje de avance de los cálculos se muestra en una barra como la de la figura 4.5. Una vez iniciado el proceso de los datos no es posible detenerlo. Cuando se llega al 100% de la realización de los cálculos la ventana se cierra para poder analizar los resultados. Los resultados se guardan en el mismo lugar por lo tanto siempre que se haga un cálculo nuevo los resultados anteriores se borran. El botón de salir cierra la ventana si es que no se hizo ningún cálculo.

4.5. Barra de porcentaje de avance en los cálculos

Cálculos Individuales

Por otro lado los cálculos individuales nos van a permitir obtener resultados del desempeño de una persona con respecto a si mismo, es decir sólo hay un nivel. Para que este proceso se pueda llevar a cabo es necesario que existan al menos tres registros de resultados de pruebas para un individuo por las mismas razones que describieron para los cálculos en conjunto. También se realiza la validación y si no se cumple el mensaje es el siguiente :

4.6. Mensaje que se manda cuando no hay suficientes registros para hacer el cálculo

Otra validación que se realiza en los cálculos individuales es cuando se presiona el botón de aceptar y no se ha seleccionado ningún registro para hacer el cálculo. Si esto sucede se muestra un mensaje de error indicándolo como los anteriores errores.

La pantalla de cálculos individuales cambia radicalmente de la de cálculos en conjunto. Esto se debe a que en esta pantalla no se trata de identificar por medio de condiciones a un grupo a analizar, sino que se selecciona una sola persona. Por lo tanto la ayuda que se da es para ir filtrando toda la información de los individuos hasta encontrar al buscado.

Por esta razón, la pantalla se divide en dos secciones; una donde se muestra la información de los individuos y otra donde se dan las condiciones de consulta. La consulta que se da de entrada, es la de todos los individuos que hayan realizado alguna prueba, en las condiciones tenemos que se puede buscar respecto a un campo mostrado en la consulta, con un operador y un valor determinado.

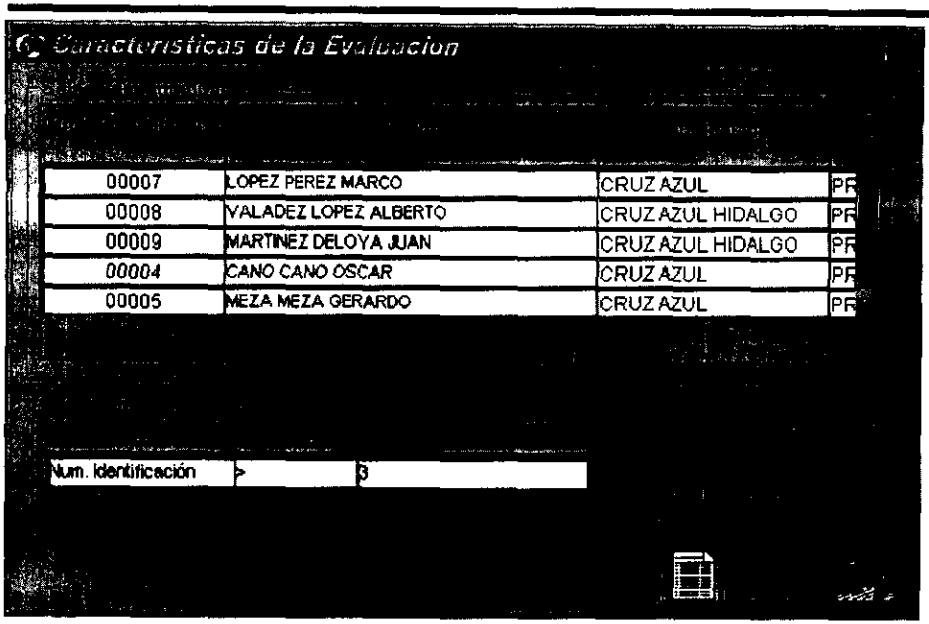


Figura 4.7. Pantalla de cálculos individuales.

A continuación se muestra una tabla donde se dan los campos utilizables para la condición, los operadores válidos para cada campo y los valores usados.

COLUMNAS	OPERADORES	VALORES
Núm. Identificación	=, >, <, ENTRE	Numérico
Apellido Paterno	ENTRE , COMO	Alfanumérico
Apellido Materno	ENTRE , COMO	Alfanumérico
Nombre	ENTRE , COMO	Alfanumérico
Agrupación	ENTRE , COMO	Alfanumérico
Categoría	ENTRE , COMO	Alfanumérico
Subcategoría	ENTRE , COMO	Alfanumérico

Tabla 4.1. Columnas, operadores y valores en la consulta de individuos.

Con estos campos y condiciones se puede filtrar la información para encontrar al individuo a analizar. Por ejemplo si queremos analizar al individuo que tiene el número de identificación igual a diez, sólo se pone en las cajas de texto seleccionando las cosas correctas :

Núm. Identificación = 10

Para los campos que son alfanuméricos se utilizan dos operadores principalmente y tienen un uso especial estos son : **"ENTRE"** y **"COMO"**, el operador entre se utiliza también en los campos numéricos y nos sirve para poder abarcar varios valores. Regresando al ejemplo anterior supongamos que se quiere buscar aun individuo que tiene un numero de identificación entre doce y quince, esto se obtendría con el operador entre:

*Núm. Identificación **ENTRE** 10,11,12,13,14,15*

Aquí se mostrarían los cinco registros, si fuera para un campo alfanumérico una condición quedaría de la siguiente forma :

*Nombre **ENTRE** ERNESTO,SAUL,LUIS,JESUS*

El otro operador es el de "COMO" este operador es muy útil cuando no se conoce completamente la cadena de caracteres a buscar. Para este operador se puede usar un carácter de control adicional este es "**%**" Por ejemplo si se quiere buscar una cadena que se sabe que termina en DES, la condición puede quedar como sigue :

*Apellido Paterno **COMO** %DES*

Si iniciara con CA y no se supiera que caracteres hay después quedaría como sigue :

*Apellido Paterno **COMO** CA%*

De esta forma se filtra la información hasta que se encuentra la adecuada o se reduce el conjunto mostrado, de tal forma que se pueda identificar al individuo buscado para realizar los cálculos. La sección de condiciones de búsqueda tiene varias peculiaridades que describirán a continuación:

- Los campos de columnas y los operadores sólo se seleccionan ya que se muestran en cajas de texto desplegables. EL valor si se tiene que poner. Ver figura 4.9.

- Se pueden tener varias condiciones de búsqueda y no solamente una, estas actuarían ligándolas con una conjunción "y", es decir, se mostrarán los registros que cumplan con la primera condición y la segunda y la tercera y...

- Para poder poner más condiciones se pone el cursor en algún campo de la última condición o simplemente presionar con el puntero del mouse un título de algún campo de la caja de condiciones de búsqueda. Posteriormente se presiona la tecla de la flecha hacia abajo (↓), en este momento se inserta otro renglón para poder poner otra condición.

- Para borrar una condición, es necesario colocar el cursor en un campo de la condición o renglón a borrar y presionar la tecla de suprimir (**Supr**), en este momento se borrará el campo.

- Una vez seleccionada la(s) condición(es) , para actualizar la consulta se pueden seguir dos caminos uno es el de presionar la tecla (**enter**) y automáticamente se actualizará, el otro camino es presionar el botón de actualización (el de la extrema izquierda).

En la figura 4.9 se muestran varias condiciones de búsqueda solicitadas.

Al seleccionar las condiciones se van validando, si se escoge una condición no válida (campos a los que no se les puede aplicar un cierto operador) se indicará por medio de un mensaje como el que se muestra en la figura siguiente.

Figura 4.8. Error de incompatibilidad entre campos y operadores de la condición.

Para seleccionar un individuo se da doble click sobre el registro seleccionado, éste cambiará de color, pudiendo iniciar los cálculos.

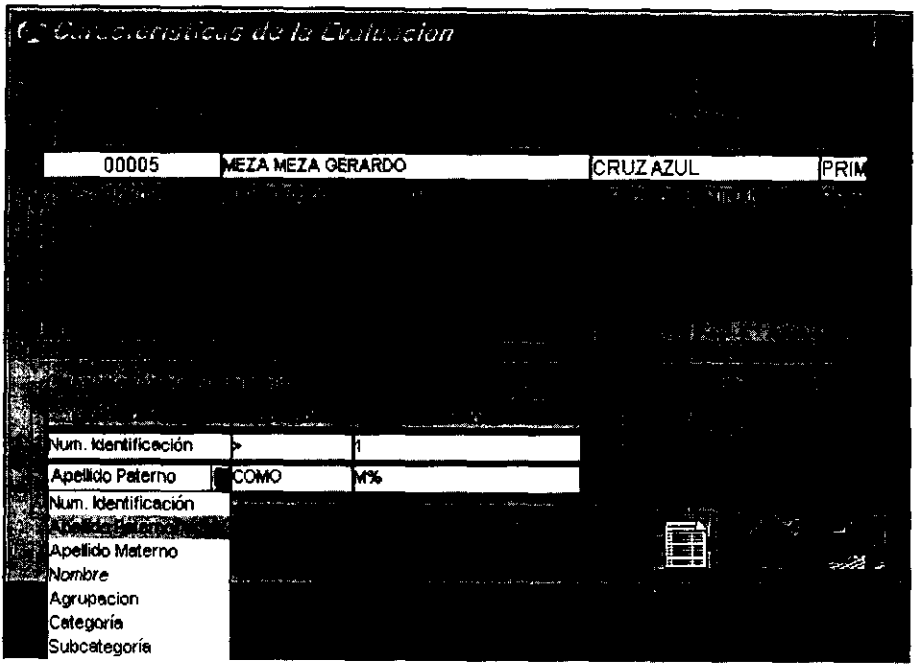


Figura 4.9. Selección con doble click de un registro. Varias condiciones de búsqueda

La pantalla tiene tres botones el de consultar, el de aceptar y el de salir. El de consultar actualiza la consulta con las condiciones de búsqueda seleccionadas. El de aceptar hace las validaciones y posteriormente los cálculos, mostrando el grado de avance en una ventana exactamente igual a la mostrada en los cálculos en conjunto en la figura 4.5. El de salir es para cerrar la ventana sin hacer cálculos.

La pantalla de cálculos individuales trabaja de forma similar a las pantallas de consulta, por esto cuando se describan las pantallas de consulta se hará referencia a la de cálculos individuales.

Cálculos Comparativos

Este tipo de cálculos son para poder realizar comparaciones entre dos o tres pruebas realizadas. Las opciones mediante las cuales se pueden realizar los cálculos se seleccionan como en los cálculos en conjunto, de esta forma se pueden tener las comparaciones a varios niveles. La pantalla que se muestra para estos cálculos es la siguiente :

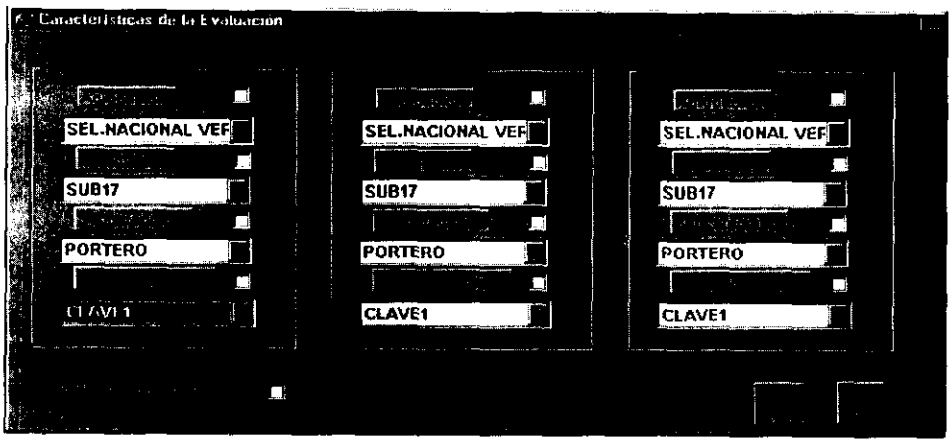


Figura 4.10. Pantalla de selección de condiciones para cálculos comparativos.

En esta pantalla se tiene una opción para cuando se quieren hacer sólo una comparación entre dos pruebas, se marca el "check box" del lado inferior izquierdo y la tercera condición se inhibe, de esta forma se indica al sistema que sólo se meterán dos condiciones para pruebas, en cualquier momento se puede desmarcar la opción y meter la tercera condición para la prueba.

Al presionar el botón de aceptar se hacen algunas validaciones para las condiciones seleccionadas, marcándose los mensajes de error correspondientes señalados en los cálculos en conjunto. Además se indica en que condición existió el error y se suspende el cálculo automáticamente y se pide corregir el error. El mensaje que se mostraría sería el siguiente :

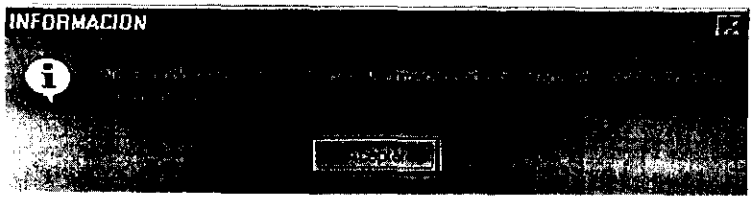


Figura 4.11. Error que se muestra cuando no se pueden realizar cálculos de alguna condición seleccionada

Si no existieron errores se realizan los cálculos mostrándose la barra de avance para cada condición.

Figura 4.12. Barra de avance para cada condición

Capitulo V.
Obtención de Resultados

Los resultados que se pueden obtener en el "PC-ES" son de dos tipos que a su vez se dividen en otros dos :

- Gráficos
- Tabulares

Es conocido por todos que los dos tipos de resultados son necesarios, los tabulares para poder tener un panorama analítico y frío donde se puede identificar claramente con escalas numéricas las deficiencias y aciertos en la preparación física. Los gráficos para poder tener un panorama más claro a nivel global y de forma fácil de comprender para obtener una idea clara de la situación actual de preparación física de un conjunto de individuos.

Resultados Tabulares

Los resultados tabulares que se obtienen del "PC-ES" se dividen en dos tipos :

- Intervalos
- Promedios

Para poder accesar a ellos se puede hacer por medio de el menú en la opción de resultados o por medio de sus íconos de acceso rápido. En las figuras siguientes se muestran las dos opciones.

Figura 5.1. Iconos de acceso rápido de resultados tabulares

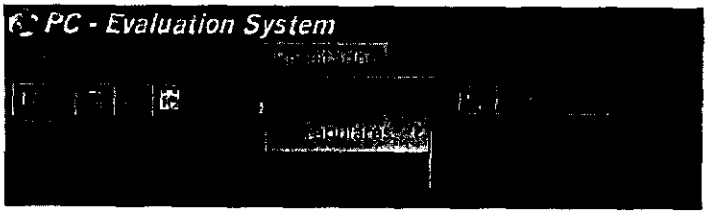
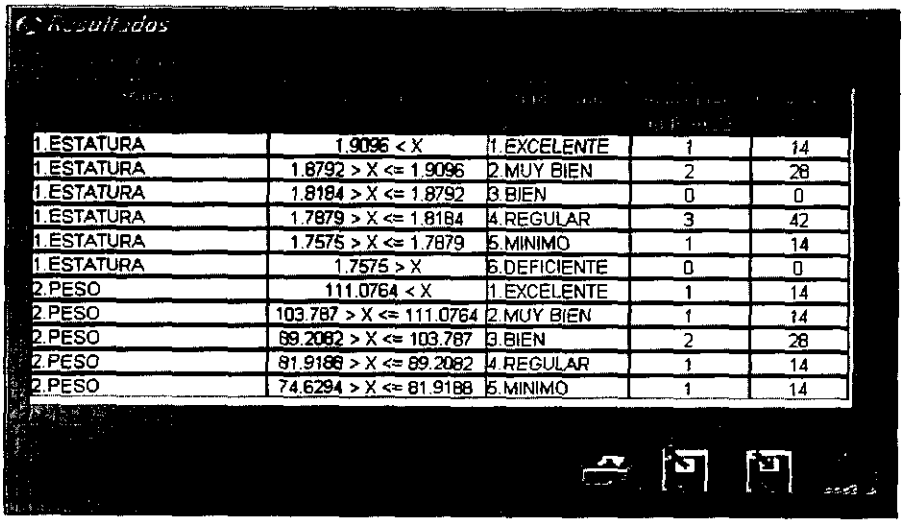


Figura 5.2. Acceso a resultados tabulares por menú.

Los resultados tabulares correspondientes a los intervalos se muestran en una ventana como la siguiente.



1.ESTATURA	1.9096 < X	1.EXCELENTE	1	14
1.ESTATURA	1.8792 > X <= 1.9096	2.MUY BIEN	2	28
1.ESTATURA	1.8184 > X <= 1.8792	3.BIEN	0	0
1.ESTATURA	1.7879 > X <= 1.8184	4.REGULAR	3	42
1.ESTATURA	1.7575 > X <= 1.7879	5.MINIMO	1	14
1.ESTATURA	1.7575 > X	6.DEFICIENTE	0	0
2.PESO	111.0764 < X	1.EXCELENTE	1	14
2.PESO	103.787 > X <= 111.0764	2.MUY BIEN	1	14
2.PESO	89.2082 > X <= 103.787	3.BIEN	2	28
2.PESO	81.9188 > X <= 89.2082	4.REGULAR	1	14
2.PESO	74.6294 > X <= 81.9188	5.MINIMO	1	14

Figura 5.3. Pantalla de resultados tabulares por intervalos

En esta ventana se muestra agrupados por prueba los intervalos que corresponderían a cada una de las valoraciones EXCELENTE, MUY BIEN, BIEN, REGULAR, MINIMO, DEFICIENTE, que a su vez tienen una correspondencia numérica en el mismo orden del 10 al 5, Aquí se muestran cuantas personas coincidieron en el intervalo mostrado en las columnas anteriores es decir que concurrencia se tuvo en un cierto intervalo y se nos indica a que porcentaje corresponde esa concurrencia en cada intervalo.

En base a estos datos se puede obtener un panorama de los cálculos realizados para algunas condiciones (ya sea en conjunto o individualmente) ya que se observa con claridad cuantas personas por prueba están en buenas o malas condiciones.

En esta ventana se muestran cuatro botones de izquierda a derecha son el de impresión; nos sirve para imprimir el reporte como se ve en la pantalla. El de importar; éste nos sirve para leer de un archivo los resultados previamente exportados a éste. El de exportar; éste nos permite exportar la información mostrada a un archivo especificado. Es importante mencionar que los datos exportados en esta opción de resultados por intervalos son los que podrán ser importados en la misma ventana de resultados por intervalos, esto nos sirve por si se quiere guardar en medio magnético algunos resultados por intervalos para posteriormente utilizarlos en excel o para su almacenamiento y posterior consulta. Entrando a la opción de resultados tabulares por intervalos en otra máquina que tenga el "PC-ES" se podrán ver los resultados importándolos de un archivo aunque no se haya generado en la misma máquina.

Para la búsqueda y selección de archivo se muestra la ventana general de Windows para selección de archivos con el directorio de trabajo que es "C:\PCES\" que se muestra a continuación.

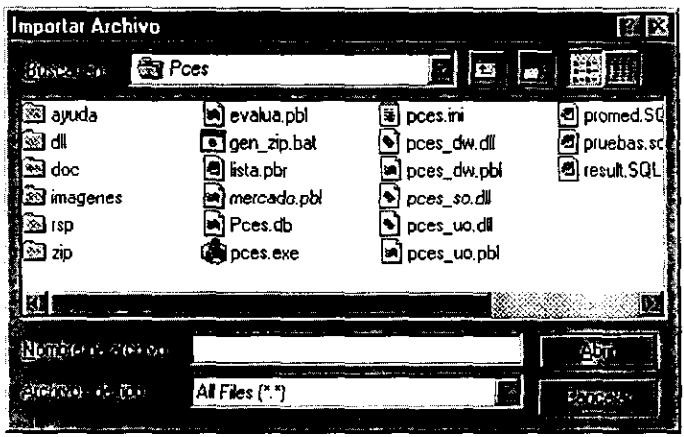
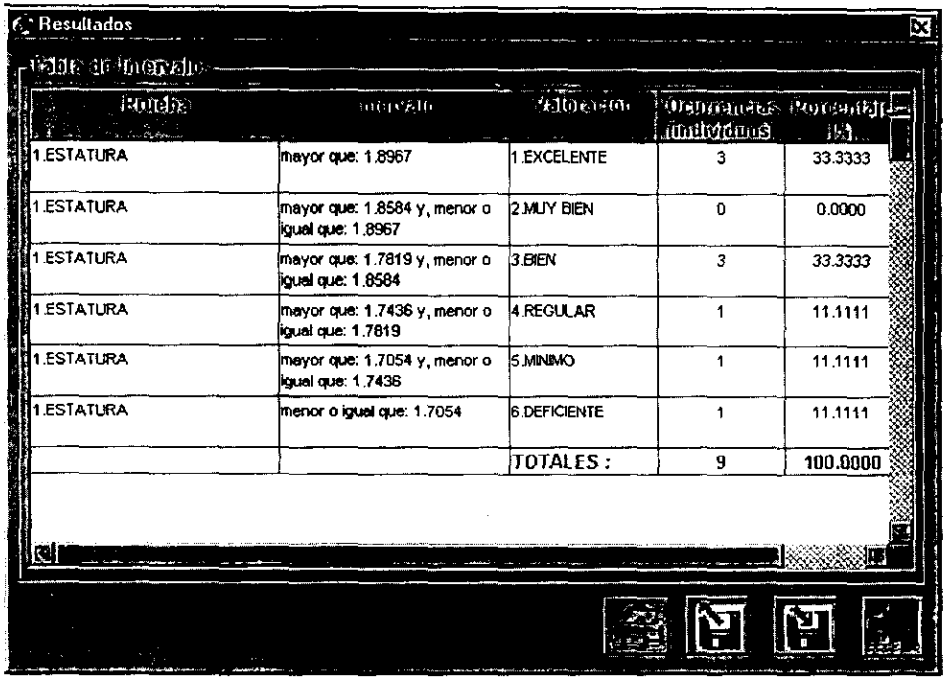


Figura 5.4. Ventana de selección de archivos de Windows

El botón de salir sirve para cerrar la ventana de resultados tabulares por intervalos.

Los resultados tabulares correspondientes a los promedios se muestran en una ventana como la siguiente.



Prueba	Intervalo	Valoración	Ocurrencias (individuos)	Porcentaje (%)
1.ESTATURA	mayor que: 1.8967	1.EXCELENTE	3	33.3333
1.ESTATURA	mayor que: 1.8584 y, menor o igual que: 1.8967	2.MUY BIEN	0	0.0000
1.ESTATURA	mayor que: 1.7819 y, menor o igual que: 1.8584	3.BIEN	3	33.3333
1.ESTATURA	mayor que: 1.7436 y, menor o igual que: 1.7819	4.REGULAR	1	11.1111
1.ESTATURA	mayor que: 1.7054 y, menor o igual que: 1.7436	5.MINIMO	1	11.1111
1.ESTATURA	menor o igual que: 1.7054	6.DEFICIENTE	1	11.1111
		TOTALES :	9	100.0000

Figura 5.5. Pantalla de resultados tabulares por promedios

Esta ventana tiene un funcionamiento similar a la anterior sólo que con alguna funcionalidad adicional. Aquí se muestran varios campos entre ellos se muestra el número de identificación de un individuo, su nombre, la clave de la aplicación y los resultados de las pruebas con valor numérico, se muestra también el promedio individual y el promedio total de la evaluación. Los individuos que se muestran son los que cumplen con las condiciones establecidas para los cálculos realizados por lo tanto si se hizo un cálculo individual sólo aparecerá el nombre de una persona varias veces con las claves de aplicación en las que haya realizado pruebas y los diferentes valores que haya obtenido en las pruebas, así como sus promedios.

Aquí se dan los resultados numéricos para que la gente que los analice pueda interpretarlos y/o utilizarlos como le convenga. Si se quiere ver con más detalle algún registro de un participante en especial bastaría con dar doble click en el registro correspondiente y se mostraría una pantalla de consulta de los datos de ese participante. Esta pantalla se muestra en la figura siguiente.

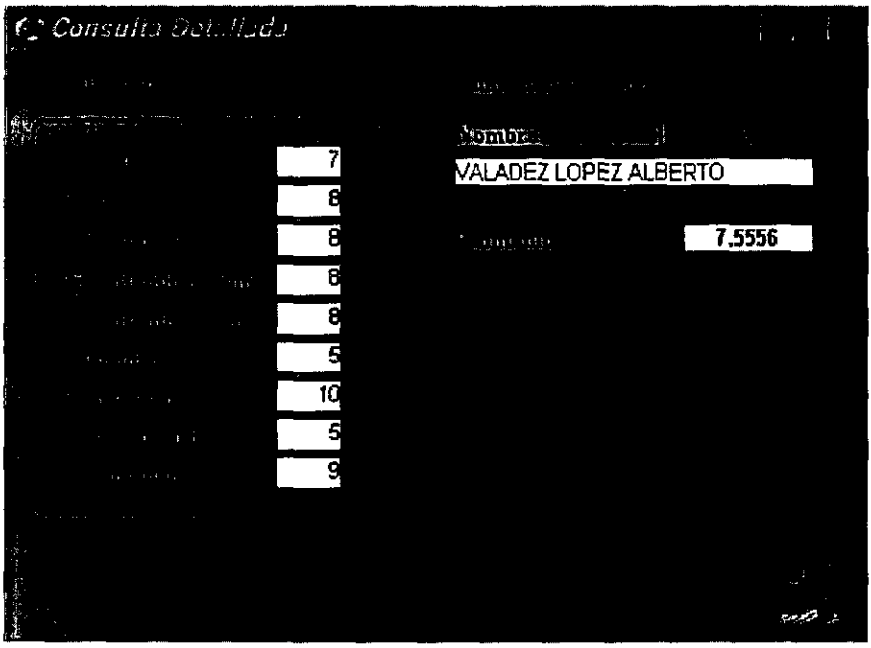


Figura 5.5. Consulta detallada de promedios.

En esta ventana se muestra una sección adicional a la de intervalos, es una caja de texto desplegable donde se puede seleccionar una prueba y unos círculos para seleccionar una opción de ordenamiento, esto nos permite ordenar los resultados de acuerdo a una prueba y de forma ascendente o descendente para su mejor manejo. Una vez seleccionada la prueba y el tipo de ordenamiento se debe de presionar el botón de consultar (el de la extrema izquierda) para afectar los datos mostrados.

Los demás botones funcionan de la misma manera que en la ventana de resultados por intervalos, sólo que al exportar se hace en otro formato y por lo tanto se tiene que importar en esta misma ventana, es decir, en una ventana de resultados por promedios de una máquina que tenga "PC-ES".

En las dos ventanas de resultados tabulares si se logra la exportación e importación correcta de archivos se manda un mensaje para avisar. Si se logró la exportación se manda un mensaje como el siguiente :

Figura 5.6. Mensaje de éxito a la exportación de resultados a un archivo

Si se logra la importación correcta del archivo se manda un aviso como el siguiente :

Figura 5.7. Mensaje de éxito a la importación de un archivo de resultados

Resultados Gráficos

Los resultados gráficos que se obtienen del "PC-ES" se dividen en dos tipos :

- Conjunto
- Por Prueba
- Temporales
- Comparativos

Para poder accesar a ellos se puede hacer por medio de el menú en la opción de resultados o por medio de sus íconos de acceso rápido. En las figuras siguientes se muestran las dos opciones.

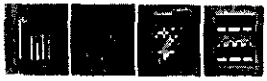


Figura 5.8. Iconos de acceso rápido de resultados gráficos.

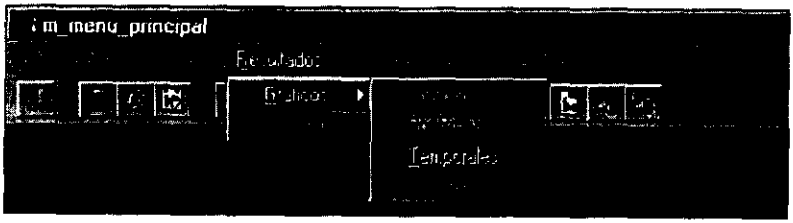


Figura 5.10. Acceso resultados gráficos a través del menú

Los resultados gráficos en conjunto se muestran en una ventana como la mostrada en la figura 5.11. En esta ventana se tienen cuatro botones principales, de izquierda a derecha está el de importar resultados de archivo; este funciona de la misma manera que en los resultados tabulares, así como el de exportar. El de graficar; éste nos sirve para después de haber seleccionado algunas características modificables de la gráfica poder volver a graficar los mismos valores afectando las mencionadas características. El de salir sirve para cerrar la ventana de resultados gráficos. Existen algunos tipos de gráficas los cuales se permite imprimir, cuando se seleccionan alguno de estos aparece al lado izquierdo del último botón uno para poder imprimir la gráfica

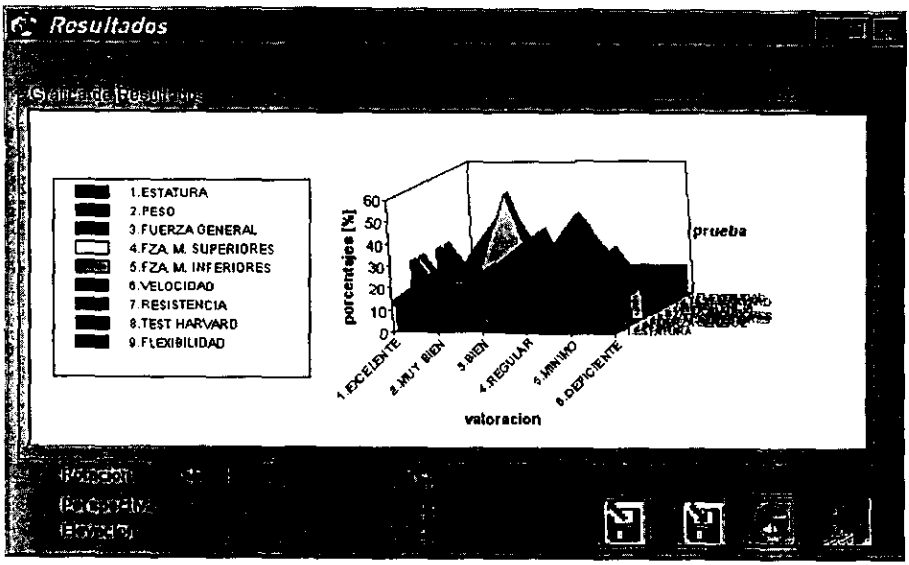


Figura 5.11. Pantalla de resultados gráficos en conjunto.

En los botones de exportar e importar se maneja el mismo formato que los resultados tabulares por intervalos, por lo tanto se podría importar un archivo que hubiese sido exportado de la ventana de resultados tabulares por intervalos. Y de forma contraria exportar uno en resultados gráficos e importarlo en resultados tabulares por intervalos.

La gráfica de resultados tiene algunas características que pueden ser cambiadas. Existen tres grupos de características que se cambian de diferente forma. En el primer grupo se encuentran los cambios que afectan automáticamente a la gráfica en su aspecto, automáticamente porque solo es necesario cambiarlas y se modifica la gráfica.

En este grupo se encuentran la rotación, la perspectiva y la elevación, en las dos últimas hacia la derecha es aumento y hacia la derecha disminución. En la primera se rota a la derecha o a la izquierda. Una restricción de este grupo es que sólo se puede aplicar a gráficas en tercera dimensión. Estas características se localizan en la pantalla de resultados gráficos (en conjunto y por prueba) en la parte inferior izquierda. Se pueden modificar presionando alguna de las flechas que se encuentran en las barras o presionando en alguna parte de la barra fuera del marcador de posición.

El segundo grupo corresponde a los tipos de gráficas, se puede elegir entre quince gráficas distintas, también a la posición de la leyenda, esta puede ser sin leyenda, arriba, abajo, derecha, izquierda. Estas características se pueden modificar en el menú de la ventana de resultados gráficos (abajo del menú principal), estas características se muestran en menús gráficos desplegables. En la figura siguiente se muestra los tipos de gráfica y las distintas posiciones de la leyenda.

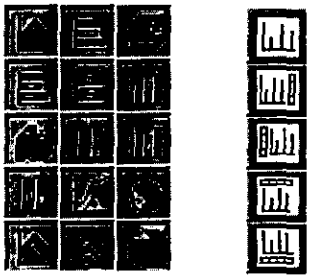


Figura 5.11. Tipos de gráficas y posiciones de leyenda

Una vez desplegados los menús gráficos se debe seleccionar con el mouse algún tipo de gráfica y posición de leyenda. Para que estos cambios tengan efecto se debe de presionar el botón de graficar, en este instante se tomarán en cuenta las modificaciones a las caracteristicas pero con los mismos datos Si se va a importar un archivo es necesario seleccionar antes el tipo de gráfica y posición de la leyenda para que se apliquen al momento de importarse (ya no habría necesidad de presionar el botón de graficar). Una vez importados los datos quedan almacenados en la base de datos local por lo tanto se pueden hacer las modificaciones sobre las características de la gráfica y se trabajará sobre los resultados importados. Si se quisiera trabajar con otros datos sería necesario o importarlos o realizar otro cálculo.

Existe otro grupo de características que se cambian sobre la gráfica de forma similar al grupo descrito anterior , sólo que en lugar de que sea con menú gráfico se hace mediante una ventana de captura. Para acceder a estas caracteristicas es necesario presionar el botón izquierdo del mouse sobre el área de la gráfica, se desplegará un menú flotante como el siguiente.

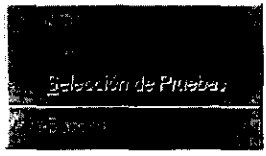


Figura 5.12. Menú flotante

En la opción de ejes se muestra una ventana como la mostrada en la figura 5.13, en la cual nos permite cambiar los conceptos asociados a los ejes de la gráfica por ejemplo, si se tiene el eje de categoría como la valoración, el eje de la serie como pruebas y el eje de valor como personas, se podría cambiar a que fuera el eje de la categoría como las pruebas el de valor como porcentaje y el de la serie como la valoración. Est se hace seleccionando de las cajas de texto los conceptos a asociar en cada eje y presionando el botón de aceptar en la pantalla mostrada en la figura 5.13.

Para ponerle un título a la gráfica es necesario seleccionar la opción en el menú y se desplegará otra ventana parecida a la de los ejes , sólo que en esta se tiene que escribir el título en la caja de texto mostrada para ello. Una vez puesto el título se presiona el botón de aceptar. La pantalla se muestra en la figura 5.14.

La opción de cancelar del menú flotante sólo lo cierra no se afecta ninguna propiedad de la gráfica. Después de que se acepta algún cambio se cierra la ventana y se regresa a la pantalla de resultados gráficos. Es necesario presionar el botón de graficar para que la gráfica se vea afectada.

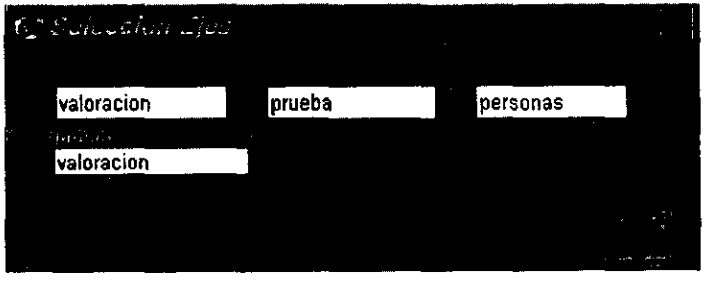


Figura 5.13. Ventana para modificar los conceptos de los ejes de la gráfica.

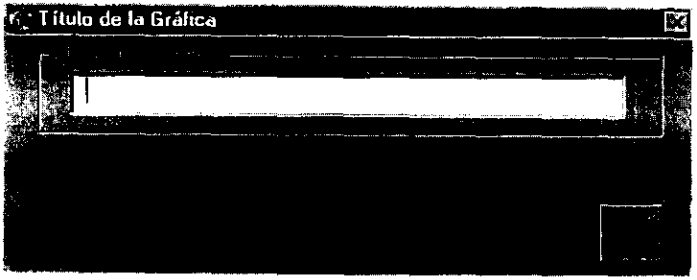


Figura 5.14. Ventana para ponerle título a la gráfica.

Los resultados gráficos por prueba nos permiten un análisis más específico, en estos se nos muestra por prueba su gráfica aislada, por esta razón su gráfica sólo tiene dos ejes el de categoría que son las pruebas y el de valor que son los resultados este puede darse en personas o en porcentaje. La pantalla en la que se muestran estos resultados es la siguiente :

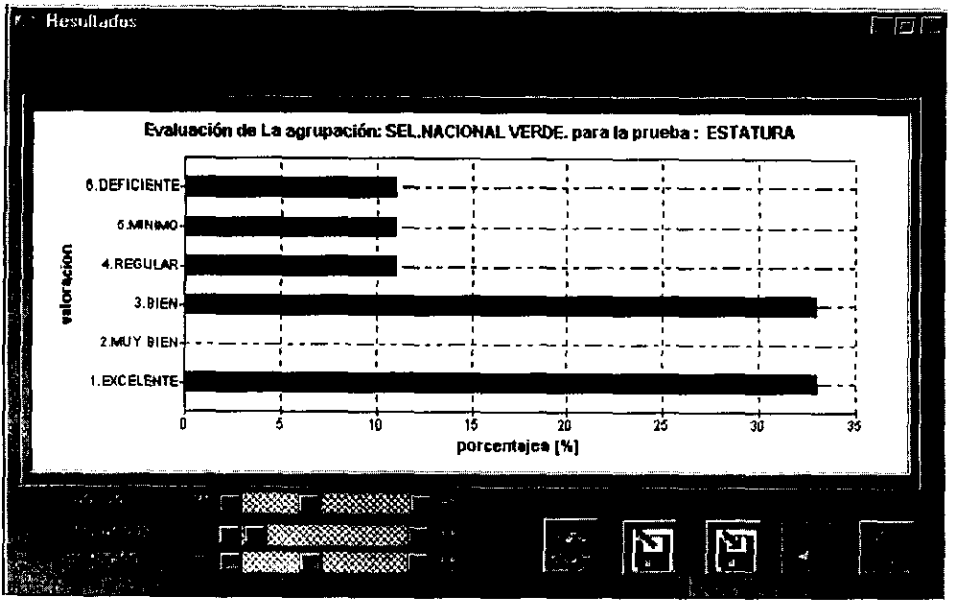


Figura 5.16. Pantalla de resultados gráficos por prueba.

Como se puede observar la pantalla es igual a la de resultados gráficos en conjunto, es fácil deducir que su funcionamiento es igual y así es sólo que tiene sus peculiaridades, las cuales se van a describir a continuación.

En el menú flotante que se desplegaba anteriormente aparecía deshabilitada la opción de selección de pruebas, ya que se mostraban todas las pruebas en conjunto, para la gráfica de resultados por prueba se habilita esta opción en el menú flotante para que puedan escoger la prueba que se quiere graficar. Entonces el menú modificado queda como se muestra en la figura siguiente.

Figura 5.17. Menú flotante

Al habilitar la opción de selección de pruebas, como las demás opciones da lugar a otra ventana donde se realiza la selección, esta ventana tiene una caja de texto desplegable donde se muestran todas las pruebas que pueden ser seleccionadas, seleccionar alguna se debe presionar el botón de aceptar y posteriormente el de graficar para afectar la gráfica. La ventana de selección de pruebas se muestra en la figura 5.18.

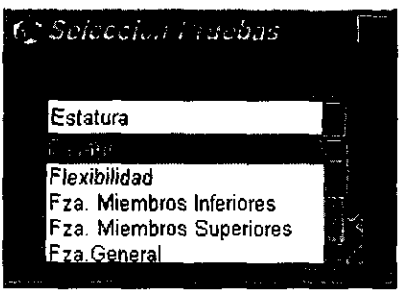


Figura 5.18. Ventana de selección de pruebas

Como ya mencionó las gráficas en este tipo de resultado son en dos ejes por lo tanto el único eje que se permite cambiar es el de valor que ya se dijo que puede ser dado en personas o porcentaje, el otro es fijo ya que corresponde a la valoración de una prueba en específico. Por esta razón la pantalla para cambiar los ejes se modifica deshabilitando los ejes que no se pueden cambiar, quedando como se muestra en la figura 5.20.

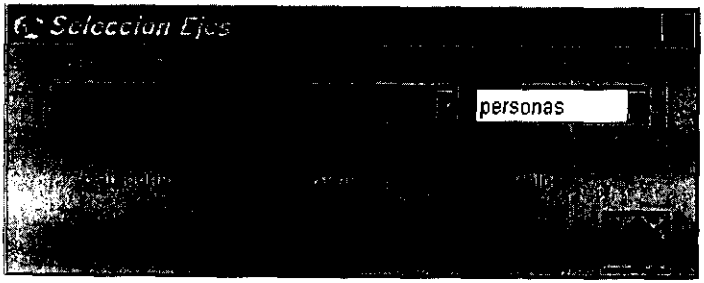


Figura 5.20. Selección de ejes

Como se puede ver la opción de resultados gráficos por prueba es muy importante porque nos permite aislar los resultados de cada una de las pruebas teniendo así un panorama mucho más claro de los resultados generales.

Los resultados temporales son accesibles solamente para cuando se realizan los cálculos individuales y la forma de operar es igual a la pantalla de resultados gráficos por prueba con la excepción de que los ejes cambian con la finalidad de obtener un análisis en el tiempo y al nivel de abstracción que se requiera.

Los resultados comparativos solo se pueden acceder cuando se realiza antes un cálculo comparativo, operan de la misma forma que la pantalla de resultados gráficos en conjunto, pero tiene la particularidad de que no se pueden importar, exportar ni imprimir datos, si se quieren obtener los datos de alguna(s) de las condiciones seleccionadas se deben de realizar los cálculos separadamente. La pantalla de resultados comparativos se muestra a continuación :

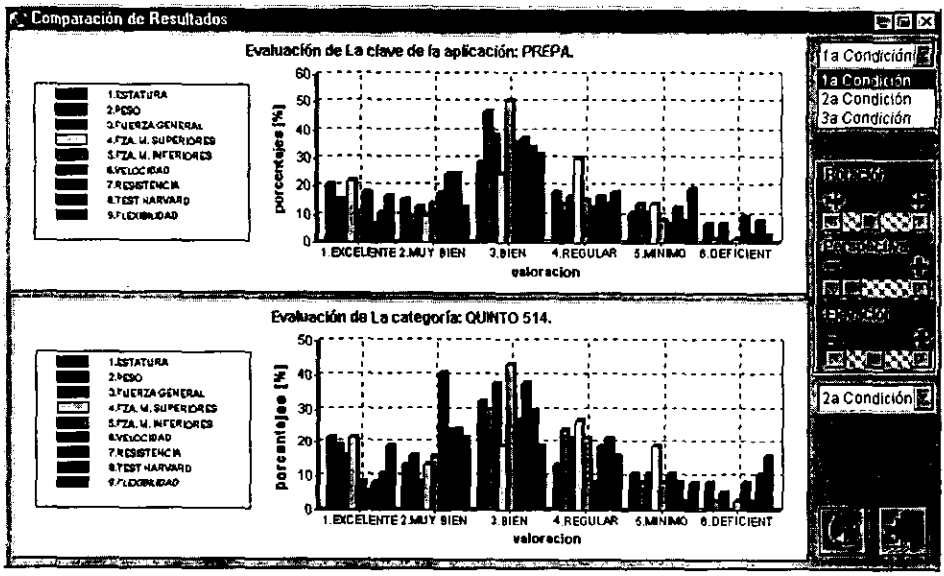


Figura 5.21. Pantalla de resultados comparativos

Como se puede observar esta pantalla aunque su operación es igual a la de resultados en conjunto difiere de aspecto, ya que nos presenta dos gráficas las cuales se pueden comparar directamente, por eso es necesario que se seleccionen al menos dos condiciones en los cálculos comparativos. Ahora bien si se tienen más de dos condiciones y se quiere poner en pantalla la tercera o se quiere cambiar la de abajo hacia arriba y la de arriba hacia abajo, en fin hacer los movimientos adecuados para la comparación, se cuenta con dos cajas de texto desplegable del lado derecho correspondientes una a cada una de las gráficas, mediante ellas se puede cambiar a la gráfica que se quiera para obtener el mejor resultado de la comparación.

Capitulo VI.
Consultas y Mantenimiento

Es necesario en un sistema que maneja tanta información el tener forma de consultarla así como de darle un mantenimiento adecuado para tener siempre confianza en ella. En el "Physical Capacity - Evaluation System" se da la posibilidad de consultar tanto a los individuos como a las pruebas físicas que hayan realizado y también a las fechas de aplicación.

Las consultas que puede dar el sistema son en dos niveles el primero de ellos es algo general y en este se permite filtrar algo de información para localizar un registro en específico. El segundo consiste en una consulta detallada de un registro seleccionado en la primera parte. Es muy similar a cuando se realizaban los cálculos individuales primero se buscaba la gente se seleccionaba y se le hacían los cálculos aquí no se hacen sólo se consulta detalladamente.

Para realizar las consultas es necesario accesar las opciones de consulta a través del menú como se muestra en la figura siguiente :

Figura 6.1. Acceso por menú a las consultas

Individuos

En la consulta de individuos se nos muestra una ventana muy similar a la mostrada en cálculos individuales, en ésta, se puede buscar a una(s) persona(s) por alguna(s) condiciones de búsqueda como se describieron en el capítulo IV. La pantalla opera igual que la que se describió en ese capítulo con algunas adiciones. Ver figura 6.2.

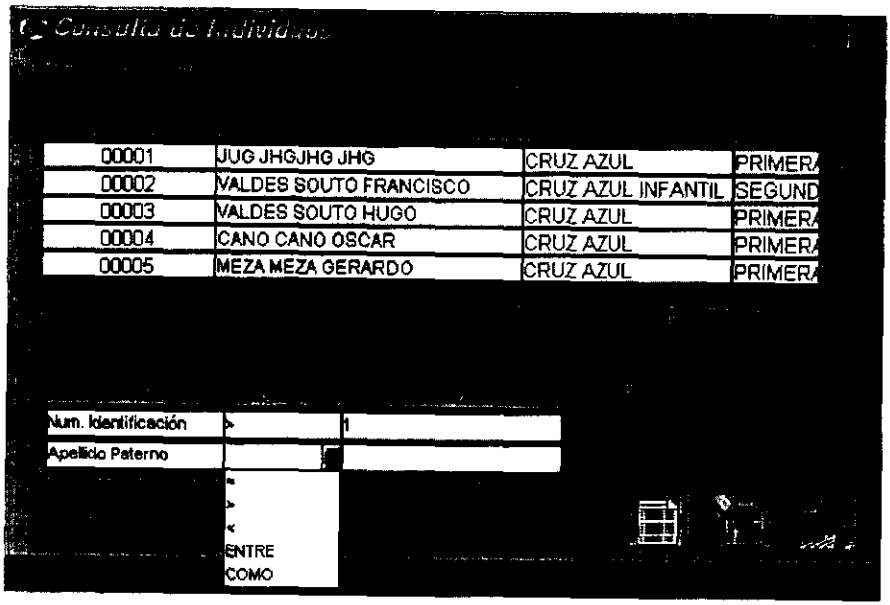


Figura 6.2. Pantalla de consulta de individuos.

Como se pretende dar mantenimiento también en la pantalla se presenta un botón para dar de baja en lugar del de aceptar que se mostraba en la otra pantalla, si se selecciona un registro (en las pantallas de consulta se seleccionan los registros con un click) y se presiona el botón de borrar se borrará el registro, es decir, el individuo y todas las pruebas realizadas por el por lo tanto no quedará huella de su existencia. Hay que tener cuidado cuando se use esta opción, por esto cuando se selecciona se muestra un mensaje de confirmación por que **una vez borrado el registro no hay forma de recuperarlo**, el mensaje de confirmación se muestra en la figura 6.3.

Figura 6.3. Mensaje de confirmación borrado de individuos.

Si no se hubiese seleccionado ningún registro se mostraría un mensaje como el siguiente y no se borraría ningún registro.

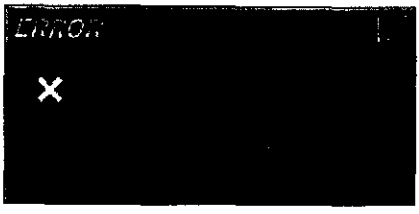


Figura 6.4. Mensaje si no se seleccionó ningún registro.

Si se quiere profundizar en la consulta de algún individuo se da un doble click en el registro que se quiera ver y se desplegará otra ventana en la cual se mostrarán todos los datos referentes al individuo seleccionado. La pantalla de la consulta detallada se muestra a continuación.

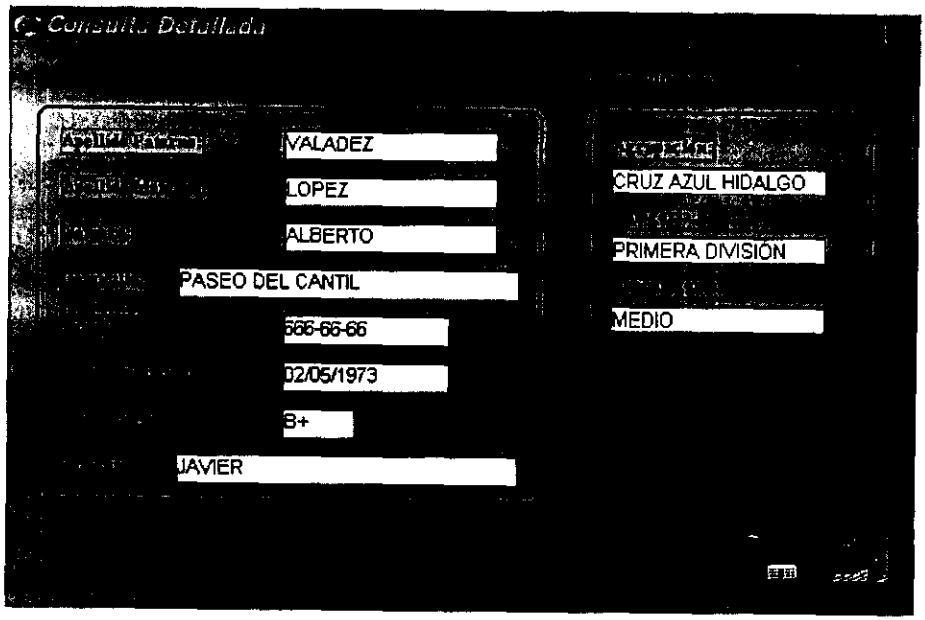


Figura 6.4. Pantalla de consulta detallada de un individuo.

En esta pantalla se tienen dos botones el de actualizar y el de salir, el de salir funciona igual que todos los de las ventanas anteriores, el de actualizar se usa cuando se quiere hacer alguna modificación sobre alguno(s) de los campos modificables, es decir que no sea el número que identifica al individuo. Esto como ya se mencionó con la finalidad de mantener actualizada la información.

Al presionar el botón de actualizar los campos modificables se activarán (se podrán modificar), cuando se seleccionen se pondrán en otro color. En los campos de la agrupación, categoría y subcategoría se mostrarán al seleccionarlos los valores correspondientes de los catálogos como se ha venido haciendo en una caja desplegable para seleccionar el valor adecuado. Cuando se hayan terminado las modificaciones se debe de teclear (**enter**) y se realizarán los cambios. Cuando no se ha seleccionado el botón de actualizar no se permite modificar los campos, si se presionó el botón y no se van a hacer modificaciones solo se teclea (**enter**).

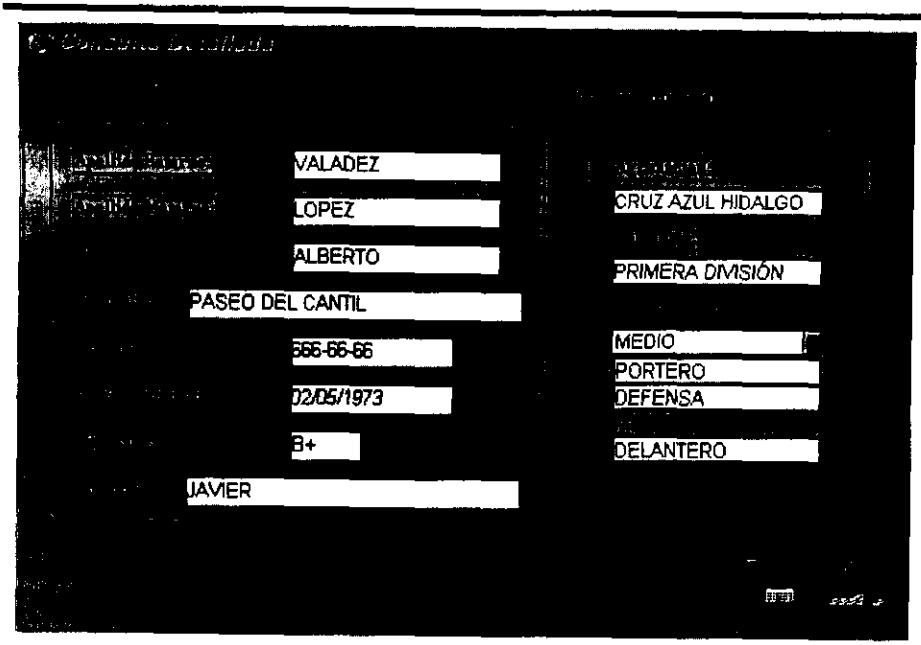


Figura 6.5. Modificación en la pantalla de consulta detallada de individuos

Si por algún motivo no se haya podido hacer la actualización de los datos se mostrará un mensaje indicándolo como el que sigue.

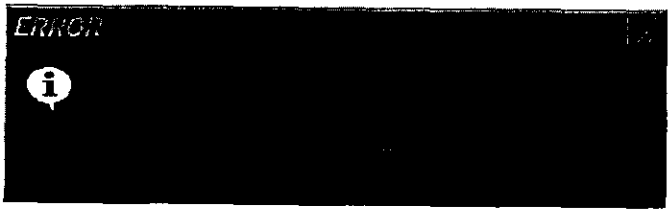


Figura 6.6. Mensaje cuando no se pueden actualizar los datos.

Pruebas Físicas

El proceso de consulta de las pruebas físicas es muy similar al de individuos, de hecho la pantalla tiene las mismas características, sólo que se muestran campos correspondientes a las pruebas físicas y las condiciones de búsqueda se ponen en función de esos campos.

Los campos que se muestran son : número de identificación, nombre, clave de la aplicación y fecha de aplicación. Dependiendo del tipo de valor de los campos, se hacen validaciones con respecto a los operadores que se pueden utilizar y se mandan mensajes de error como los descritos en el capítulo IV. En la siguiente figura se muestra la pantalla de consulta de pruebas físicas. Posteriormente se muestra la tabla 6.1 correspondiente a las columnas, sus operadores y valores válidos.

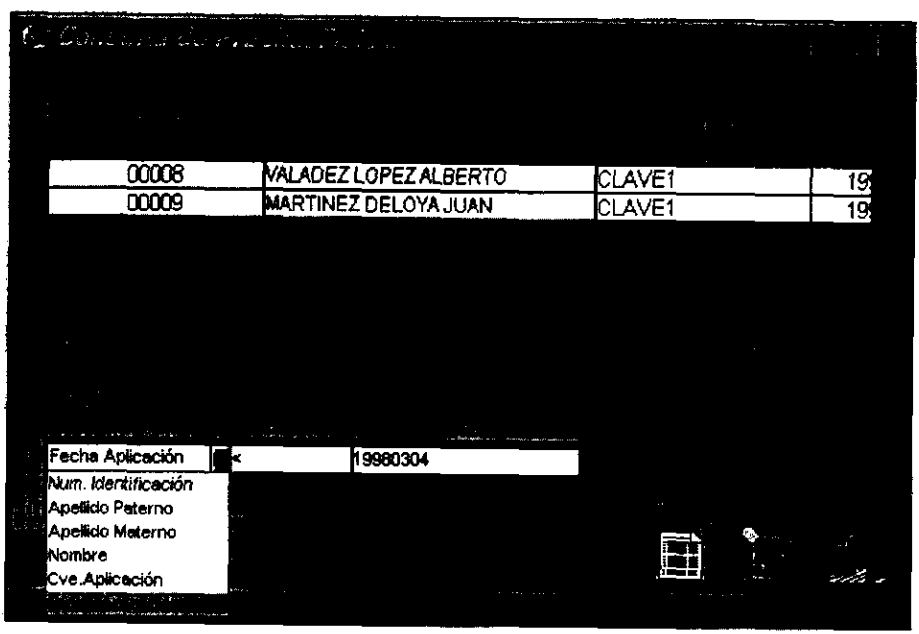


Figura 6.7. Pantalla de consulta de Pruebas Físicas.

COLUMNAS	OPERADORES	VALORES
Núm. Identificación	=, >, <, ENTRE	Numérico
Nombre	ENTRE , COMO	Alfanumérico
Clave Aplicación	ENTRE , COMO	Alfanumérico
Fecha Aplicación	=, >, <, ENTRE	Fecha

Tabla 6.1. Columnas, operadores y valores en la consulta de pruebas físicas.

En esta pantalla como en las otras que permiten condiciones de búsqueda, se pueden poner varias y presionando (**enter**) o el botón de consultar se actualiza la consulta.

El botón de borrar cuando se presiona borra lo referente a los resultados obtenidos en una prueba por un individuo (los especificados en el renglón seleccionado). Se sigue el mismo proceso de validación, es decir si no se seleccionó algún renglón se muestra el mensaje de la figura 6.4. Si hubo un registro seleccionado se muestra el mensaje de validación siguiente:

Figura 6.8. Mensaje de confirmación de borrado de una prueba.

Una vez que se ha localizado el registro con las condiciones de búsqueda se puede hacer una consulta detallada sobre el, presionando con doble click sobre el registro elegido se accede a la pantalla de la consulta

detallada, esta funciona de la misma forma que la de consulta detallada de individuos es decir también se pueden realizar modificaciones a los valores introducidos, sólo que aquí los campos no modificables son 4, el número de identificación, la clave de la aplicación, nombre y fecha de aplicación.

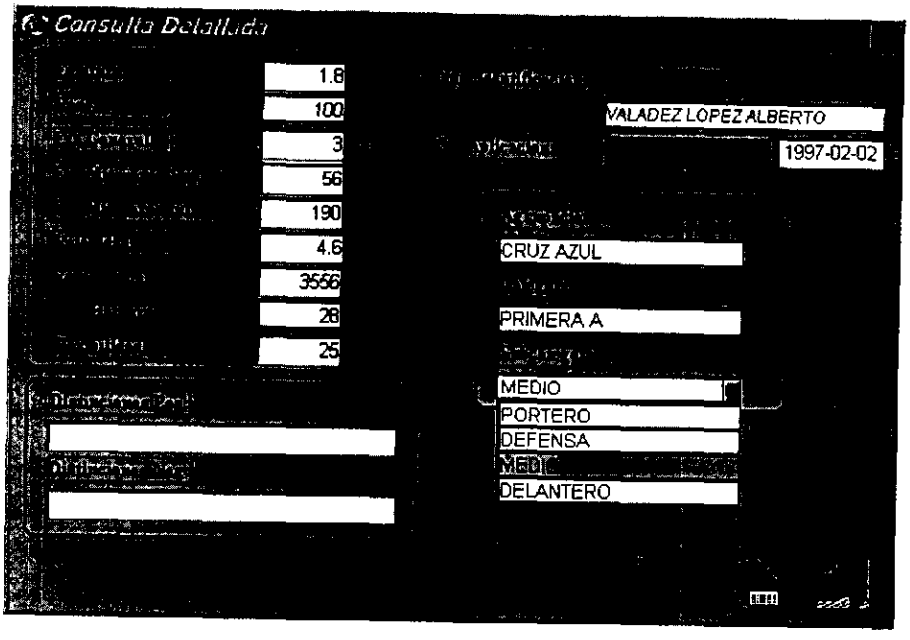


Figura 6.9. Pantalla de consulta detallada realizando modificación.

Si al hacer la modificación no se pudiera por algo se mostraría el mensaje de la figura 6.6. Donde se recomienda salir del sistema apagar la máquina y volverlo a intentar.

Fechas de Aplicación

Para la consulta de fechas de aplicación como solamente se tienen dos campos que son: la clave de la aplicación y la fecha, se muestran en la misma pantalla, por lo tanto no se tiene una consulta detallada. Ver figura 6.10.

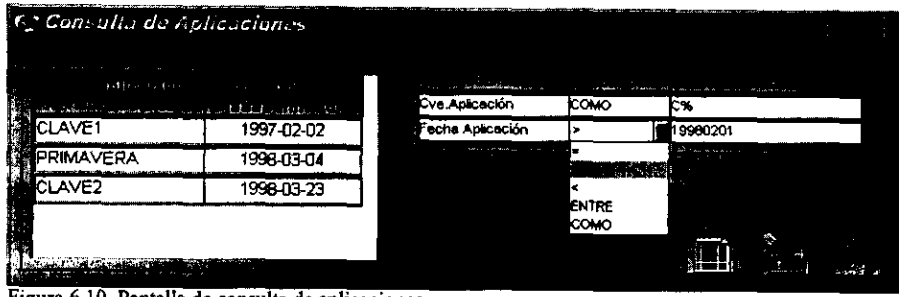


Figura 6.10. Pantalla de consulta de aplicaciones.

En esta ventana se pueden hacer búsquedas con condiciones basadas en las columnas que se muestran del mismo modo de operación que en las otras pantallas. La tabla correspondiente a las condiciones de búsqueda se pone a continuación.

COLUMNAS	OPERADORES	VALORES
Clave Aplicación	ENTRE , COMO	Alfanumérico
Fecha Aplicación	=, >, <, ENTRE	Fecha

Tabla 6.2. Columnas, operadores y valores en la consulta de aplicaciones.

En la pantalla de consulta de fechas de aplicación también se pueden borrar los registros pero aquí si se borra un registro también se borran las pruebas asociadas a el. Por lo tanto una vez dado de alta una clave de aplicación y su fecha y sea utilizada no se puede borrar a menos que se esté seguro de ello. El mecanismo de borrado es igual al de las demás pantallas y los mensajes son similares a continuación se muestra el de confirmación figura 6.11, porque si no se seleccionó registro se muestra el mismo que en la otras ventanas ver figura 6.4.

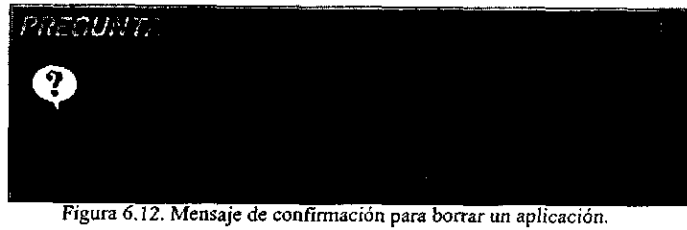


Figura 6.12. Mensaje de confirmación para borrar un aplicación.

Ventanas

Otra consulta que se da en el sistema no es a nivel de datos, sino a nivel de operación, por esto no está en el menú de consultas sino en otro aparte, esta consulta se puede accesar a través del menú como se muestra en la figura siguiente. En esta consulta aparecen las ventanas que están siendo utilizadas y se puede accesar a ellas seleccionándolas en el menú. La ventana que esté activa aparece palomeada.

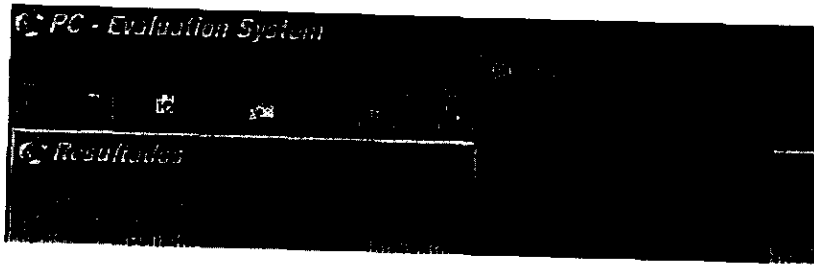


Figura 6.13. Acceso a la consulta de ventanas activas

Esto se puede utilizar con la finalidad de saber que ventanas están siendo utilizadas aunque no se vean porque estén cubiertas por otras, y si se quiere utilizar una de ellas se selecciona y se cambia a la deseada.

Capitulo VII.
Utilerías

La última opción del menú en el sistema es la de utilerías, en esta opción se nos presentan una serie de actividades a realizar no muy frecuentemente o en casos especiales bajo completa decisión del operador del "Physical Capacity – Evaluation System". La forma de accesar es a través del menú y algunas opciones a través de sus íconos de acceso rápido. El acceso a través del menú se muestra a continuación para observar las opciones que hay e ir describiendo cada una de ellas.

Figura 7.1. Acceso al menú de utilerías.

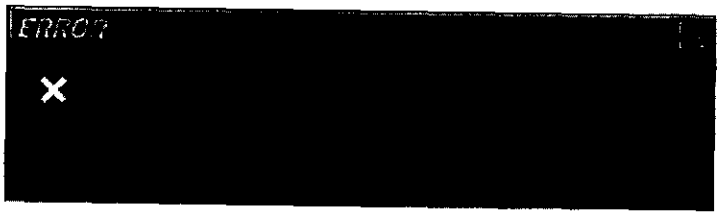
Como se puede observar se tienen cinco opciones divididas en dos grupos:

- mantenimiento de la base de datos
- claves de acceso

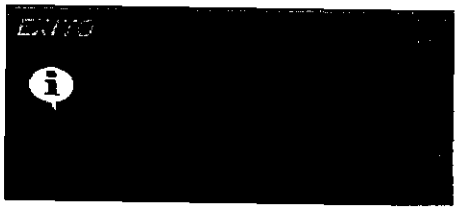
Base de Datos

Con respecto al mantenimiento de la base de datos se tienen dos opciones: una es respaldar la base de datos y la otra es cargar la base de datos. Al respaldar la base de datos lo que se hace es copiar el archivo de la base de datos (c:\Pces**Pces.db**) que se está utilizando a otro archivo que sería el respaldo (c:\Pces**Pces.rsp**) de la base de datos. De esta manera se guardan todos los cambios hechos hasta la última modificación en la base de datos en el archivo de respaldo.

Si existiera algún problema con la base de datos actual, tendríamos de esta manera un respaldo de la misma y no perderíamos el trabajo de unos día, meses o años. Cuando se ejecuta el proceso de respaldo nos indica cuando termina como termino mediante unos mensajes como los que se muestran a continuación.



(a)



(b)

Figura 7.2. Mensajes de respaldo de la base de datos (a) no se respaldó, (b) si se respaldó.

No es suficiente con tener guardado en un archivo de respaldo todos los cambios hechos hasta la última modificación si no se tiene un proceso para poder ponerlos activos y utilizarlos. Por ejemplo imaginemos que se daña la base de datos y se tiene un respaldo, sin hacer nada no nos sirve pero sin embargo si sustituimos la base de datos por el archivo de respaldo podemos seguir trabajando correctamente con los últimos datos respaldados. Esto es un proceso que realmente se podría hacer a mano sin embargo se puso la opción para hacerlo automáticamente.

Esta opción es la de cargar la base de datos, en esta opción se sustituye la base de datos por el archivo de respaldo y podemos seguir trabajando con lo último que se haya respaldado. Para garantizar que la mayoría de los cambios se realicen se realiza un respaldo de la base de datos cada vez que se entra al sistema, por lo tanto siempre se tienen los datos hasta que se inició la sesión. Si durante la sesión hacen otro respaldo bueno se tienen más datos

respaldados. El archivo que se genera de respaldo siempre sobrescribe el anterior.

En la carga de la base de datos también se nos indica por medio de mensajes como concluyó. Ver figura 7.3.

(a)

(b)

Figura 7.3. Mensajes de carga de la base de datos (a) no se cargó, (b) si se cargó.

Cuando se genera el respaldo se genera un archivo de respaldo independiente, es fácil ver que si se quiere actualizar los datos de máquina en otra máquina que tenga licencia de "Physical Capacity – Evaluation System" , bastaría con copiar el archivo de respaldo en la máquina y correr la opción de cargar la base de datos, en ese momento se tendrían datos iguales en las dos máquinas por lo tanto se podrían trabajar de igual manera. Ver las restricciones y la descripción de cómo actualizar los datos de una máquina a otra en el apéndice B.

Estas dos opciones de respaldo y carga de base de datos tienen acceso mediante íconos de acceso rápido que se muestran a continuación.

Figura 7.4. Iconos de acceso rápido de respalda y carga base de datos

Claves de Acceso

Con respecto a las claves de acceso, tenemos como se describió en el capítulo I Entrada al Sistema, la entrada al sistema se realiza mediante un login y un password (clave de acceso) , que serán proporcionados cuando se instale la versión del "Physical Capacity – Evaluation System". Esta clave de acceso servirán para entrar la primera vez al sistema sin embargo estarán muy al alcance de varias personas ya que se entregarán escritos.

Lo que se recomienda es que una vez entrando al sistema se de alta una nueva clave de acceso y se borre la anterior, al menos que se le cambie el password. Esto con el sentido que se restrinja el acceso al sistema. Una vez que se tiene completo control sobre quien puede entrar al sistema se puede permitir a otras personas tener acceso a el sin darle a conocer nuestra clave de acceso. Esto se hace dando de alta las claves de acceso que sean necesarias. Se recomienda que el password contenga letras caracteres especiales y números para que sea más difícil de decifrar por otra persona además de no decirlo a nadie ni escribirlo en ningún lado. La pantalla para dar de alta las claves de acceso se muestra en la figura 7.5.

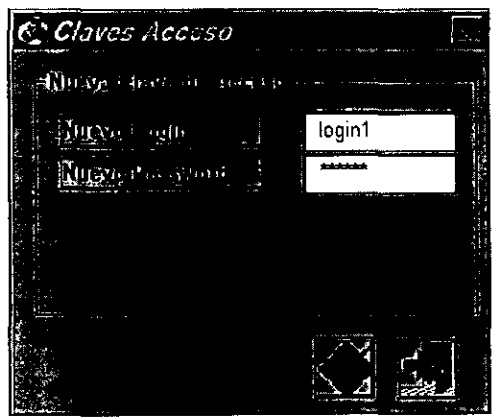


Figura 7.5. Alta de claves de acceso.

Podemos observar que en esta pantalla se pide el login y el password una vez introducidos y que se esté seguro de ellos se presiona el botón de aceptar y a partir de ese momento se puede utilizar para entrar al sistema. Si la clave de acceso es aceptada se muestra el mensaje de la figura 7.6, sino se

mandará un mensaje de error similar a los descritos anteriormente, se recomienda salir del sistema e intentarlo de nuevo.

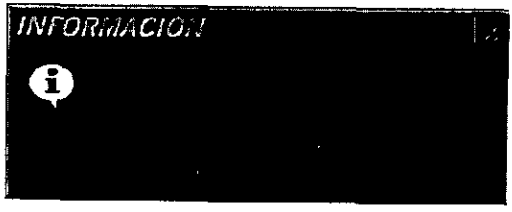


Figura 7.6. Éxito al dar de alta una clave de acceso.

Si en algún momento ya no se quiere que alguien entre al sistema se puede borrar su clave de acceso y desde ese momento ya no tendrá acceso al sistema. Hay que tener cuidado sobre la gente que se le da acceso al sistema ya que se está controlando el acceso al sistema mediante la clave de acceso más no el uso de las opciones, esto implica que cualquiera que entre al sistema puede dar de baja y alta claves de acceso mediante la pantalla que se muestra a continuación.

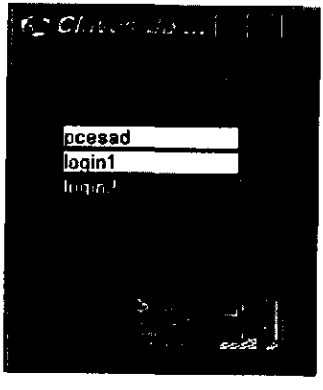


Figura 7.7. Pantalla de bajas de claves de acceso.

Aquí se tiene que seleccionar un registro con un click y después presionar el botón de borrar, sino se selecciona se mostrará un mensaje como el de la figura 6.4, si se seleccionó se mostrará un mensaje de confirmación como el de la figura 7.8 y si se acepta se borrará el registro.

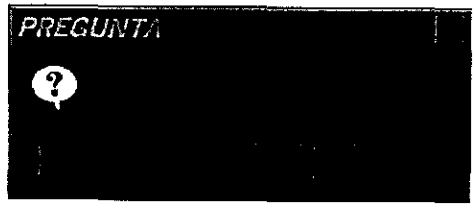


Figura 7.8. Mensaje de confirmación.

Algunas veces va a ser necesario que se quiera cambiar el password de alguna persona debido a que haya dejado de ser confidencial, esto es posible mediante la opción de cambia de clave de acceso. Ver figura 7.9.

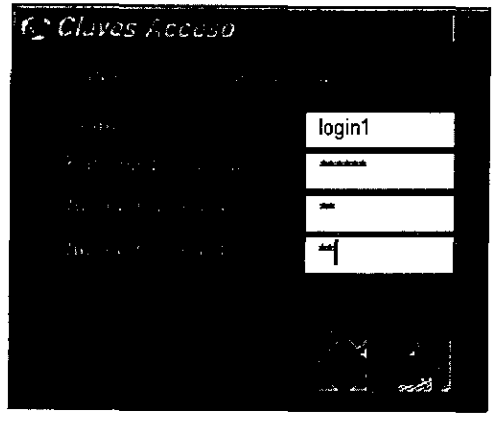
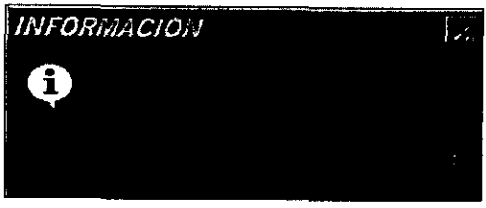


Figura 7.9. Pantalla de cambio de clave de acceso.

Al cambiar el password se presiona el botón de aceptar, si no son correctos los datos introducidos se manda el mensaje correspondiente y si son correctos se hace el cambio y se manda el mensaje correspondiente. Los mensajes que se mandan se muestran en la figura 7.10.

(a)

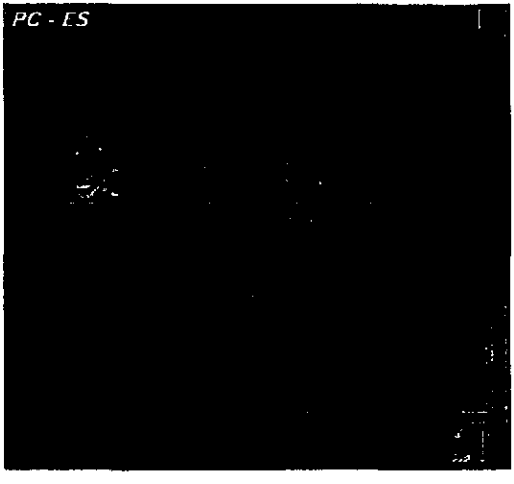


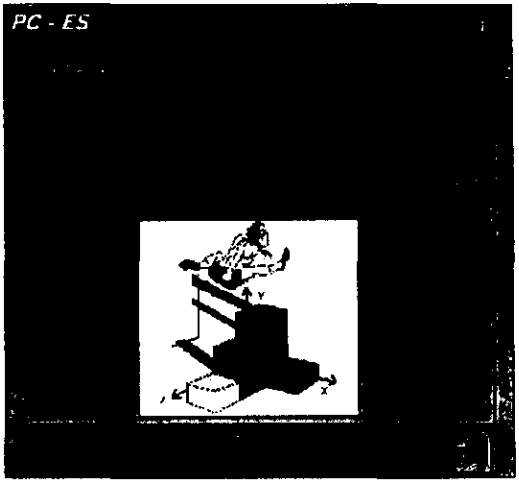
(b)

Figura 7.10. Mensaje de cambio de password. (a) fracaso datos incorrectos, (b) éxito en el cambio.

Apéndice A
Acerca del "Physical Capacity – Evaluation System"

Los datos acerca del sistema se muestran en la opción del menú acerca de. Se muestran en una ventana con dos pestañas en las cuales se indicas los derechos reservados y el registro de derechos de autor, así como a quién se le autoriza el uso del sistema. El sistema está registrado en Derechos de Autor y todos los derechos son reservados. Las ventanas son las siguientes:





Apéndice B
Intercambio de Datos

Cuando se tienen varias máquinas con el "Physical Capacity – Evaluation System" es posible el exportar los resultados a archivos tipo texto en una máquina e importarlos para su análisis en otra.

Al escoger la opción de exportar se genera un archivo texto de los datos mostrados si son resultados tabulares. Para el caso de los resultados gráficos su exportación equivaldría a exportar los resultados tabulares por intervalos ya que la información a partir de la que se hace la gráfica se toma de ahí. Los datos exportados serían separados por tabuladores y colocados en el orden mostrado, esto es con la finalidad de que se pudieran exportar en Excel y hacerles algún proceso extra.

Un archivo exportado tiene la siguiente apariencia.

result.txt - Bloc de notas

num_prueba	prueba	num_intervalo	intervalo	num_valoracion	valoracion	num_o
1	1.ESTATURA	1	1.9096 < X	1	1.EXCELENTE	1	14	
1	1.ESTATURA	2	1.8792 > X <= 1.9096	2	2.MUY BIEN	2	28	
1	1.ESTATURA	3	1.8184 > X <= 1.8792	3	3.BIEN	0	0	
1	1.ESTATURA	4	1.7879 > X <= 1.8184	4	4.REGULAR	3	42	
1	1.ESTATURA	5	1.7575 > X <= 1.7879	5	5.MINIMO	1	14	
1	1.ESTATURA	6	1.7575 > X	6	6.DEFICIENTE	0	0	
2	2.PESO 1	111.0764 < X	1	1.EXCELENTE	1	14		
2	2.PESO 2	103.787 > X <= 111.0764	2	2.MUY BIEN	1	14		
2	2.PESO 3	89.2082 > X <= 103.787	3	3.BIEN	2	28		
2	2.PESO 4	81.9188 > X <= 89.2082	4	4.REGULAR	1	14		
2	2.PESO 5	74.6294 > X <= 81.9188	5	5.MINIMO	1	14		
2	2.PESO 6	74.6294 > X	6	6.DEFICIENTE	1	14		
3	3.FUERZA GENERAL	1	4.6589 < X	1	1.EXCELENTE	2	28	
3	3.FUERZA GENERAL	2	3.946 > X <= 4.6589	2	2.MUY BIEN	0	0	
3	3.FUERZA GENERAL	3	2.5362 > X <= 3.946	3	3.BIEN	2	28	
3	3.FUERZA GENERAL	4	1.8313 > X <= 2.5362	4	4.REGULAR	3	0	
3	3.FUERZA GENERAL	5	1.1264 > X <= 1.8313	5	5.MINIMO	0	0	
3	3.FUERZA GENERAL	6	1.1264 > X	6	6.DEFICIENTE	0	0	
4	4.FZA. M. SUPERIORES	1	71.6798 < X	1	1.EXCELENTE	2	28	
4	4.FZA. M. SUPERIORES	2	65.7598 > X <= 71.6798	2	2.MUY BIEN	0	0	
4	4.FZA. M. SUPERIORES	3	53.9196 > X <= 65.7598	3	3.BIEN	4	57	
4	4.FZA. M. SUPERIORES	4	47.9996 > X <= 53.9196	4	4.REGULAR	0	0	
4	4.FZA. M. SUPERIORES	5	42.0795 > X <= 47.9996	5	5.MINIMO	0	0	
4	4.FZA. M. SUPERIORES	6	42.0795 > X	6	6.DEFICIENTE	1	14	
5	5.FZA. M. INFERIORES	1	216.5381 < X	1	1.EXCELENTE	1	14	
5	5.FZA. M. INFERIORES	2	201.5115 > X <= 216.5381	2	2.MUY BIEN	0	0	
5	5.FZA. M. INFERIORES	3	171.4582 > X <= 201.5115	3	3.BIEN	3	42	
5	5.FZA. M. INFERIORES	4	156.4316 > X <= 171.4582	4	4.REGULAR	0	0	
5	5.FZA. M. INFERIORES	5	141.4049 > X <= 156.4316	5	5.MINIMO	0	0	
5	5.FZA. M. INFERIORES	6	141.4049 > X	6	6.DEFICIENTE	1	14	
6	6.VELOCIDAD 1	5.0201 > X	1	1.EXCELENTE	2	28		
6	6.VELOCIDAD 2	5.0201 >= X < 5.3729	2	2.MUY BIEN	0	0		

No sólo es posible compartir los resultados obtenidos, debido a que el sistema está basado en una base de datos local sólo la máquina en que reside tiene acceso a los datos, sería posible que si se tiene más de una máquina se quiera tener actualizada la información contenida en ambas. Esto es posible ya que como se mencionó en el capítulo VII, al respaldarse la base de datos se genera un archivo el cual puede ser copiado a cualquier máquina que tenga el sistema y posteriormente cargar en la máquina que se copió el archivo la base de datos y en ese momento se actualizará la información ya que la base de datos se sustituye por el nuevo archivo que sería una replica de la base de datos original.

El archivo que se genera al respaldar la base de datos es :

Pces.rsp

y se encuentra en :

C:\Pces

por lo tanto si se quisiera pasar la información de la base de datos de una máquina a otra sería necesario copiar ese archivo a la misma dirección en la máquina destino, si ya existiera el archivo que va a ser lo común ya que siempre que se inicie una sesión se respalda la base de datos local, se debe de sobrescribir o borrar primero el ya existente en la máquina destino y después copiar el que contenga los datos actualizados. Para realizar esto se puede utilizar un medio magnético para copiar de un lado a otro o correo electrónico.

Al generar este archivo se tiene un medio de almacenamiento de información a largo plazo, ya que con el procedimiento adecuado se puede guardar este archivo en medio magnético y hacer un almacenamiento definitivo de la información por si en algún momento se requiere poderla volver a cargar. Esto nos da la posibilidad de que por ejemplo cada mes se guarde un disco con la base de datos respaldada, si dentro de dos años se quiere obtener los resultados obtenidos en algún mes se carga el archivo correspondiente como base de datos y se puede volver a obtenre los resultados obtenidos.

Lo mismo se puede aplicar para los archivos exportados de resultados, sólo que los archivos van a ser mucho más pequeños (aproximadamente 4K) y se podrían guardar varios en un disco de tal forma que cuando se requiriera

volver a analizar algunos resultados específicos sólo se necesitaría importar en cualquier máquina con el "Physical Capacity – Evaluation System" y se tendrían instantáneos los resultados obtenidos en algún momento anterior. De esta forma se tienen almacenados en medio magnético los datos de los resultados.

Hay que recordar que una de las ventajas de almacenar en medio magnético es que se tiene la capacidad de almacenar mayor cantidad de información en menos espacio. Por lo tanto al almacenar las bases de datos con varios cortes en el tiempo o los archivos de resultados obtenidos no es necesario guardar reportes escritos porque en cualquier momento se pueden generar.

La opción de guardar reportes escritos es dependiente de los intereses de cada agrupación, así como la implementación del proceso de respaldo a medio magnético. Aquí se sugiere que el respaldo de base de datos sea cada mes y que los resultados se respalden todos los que se obtengan.

Apéndice C
Manejo de Errores del sistema

La forma en que el sistema presenta los errores es mediante ventanas de error que dan una breve descripción del error. Los errores que se presentan son de dos tipos :

- De validación
- De acceso a la base de datos

Los errores de validación se muestran si en alguna pantalla de captura se comete un error al introducir algún dato, también se pueden presentar en las pantallas de consultas donde se da la opción de filtrar información si los campos no son correspondientes al tipo de operador y/o al tipo de dato.

Los errores de acceso a la base de datos son aquellos en los cuales se tiene que interactuar con la base de datos y no se logra, puede ser cuando se trata de introducir datos en alguna pantalla de captura (una vez que se han pasado las validaciones) , también se pueden producir cuando se quiere obtener información de la base de datos (consultas) o cuando se quieren modificar algunos datos.

Los errores de validación son fáciles de corregir y de hecho se pueden corregir en el instante en la ventana correspondiente. Sin embargo los de acceso a la base de datos son más complicados. Cada vez que se utiliza la base de datos se crea un archivo temporal (*:TMP) correspondiente a la base de datos, entonces se pueden ir acumulando varios archivos temporales que podría estar afectando al buen funcionamiento de la misma.

Por esta razón si llegan a existir frecuentemente errores de acceso a la base de datos es recomendable borrar los archivos temporales, estos son los que tienen extensión (.TMP). Es necesario buscar en la máquina estos archivos y eliminarlos.

También en necesario dar frecuentemente un exploración a la máquina (scandisk) para eliminar la posibilidad de tener fragmentos de archivos dañados que impidan el acceso a la base de datos. Esto se realiza ejecutando en el menú de inicio de Windows 95 en la opción de ejecutar "scandisk.exe",

y aparecerá una ventana donde se pregunta que unidad se desea scanear y de que tipo, se selecciona la unidad "C:" y el tipo es completa.

Una vez que se ha hecho esto si no se logra el acceso a la base de datos será necesario realizar el proceso de cargar el último respaldo. Esto se explica en el capítulo VII y en el Apéndice A.

Esto realmente es muy raro que suceda, sin embargo, es necesario tener los conocimientos para poderlo resolver.

Apéndice D
Instalación

Una vez que se adquiere el sistema, se entregará un CD o unos discos de instalación, para cualquier caso se debe de ejecutar el archivo ***instal.exe***, este archivo se encuentra en el CD o en el disco de instalación 1.

Para el caso de que se tengan discos de instalación el programa irá solicitando los discos necesarios hasta completar la instalación.

Una vez terminada la instalación se puede ejecutar el programa, para empezar a introducir datos. Se recomienda que la primera vez que se entre al sistema se debe de dar de alta claves de acceso para los usuarios y borrar la que se estableció por default. La clave se encontrará en el último de los discos de instalación y/o en el CD de instalación, el archivo se llama Pass.txt, y se puede abrir con cualquier procesador de texto.