ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

FACULTAD DE INFORMÁTICA Y ELECTRÓNICA

ESCUELA DE INGENIERÍA EN SISTEMAS

"ESTUDIO COMPARATIVO DE LOS PARADIGMAS DE PROGRAMACION APLICADO A LA CONSTRUCCION DE MODELOS MATEMATICOS PARA LA PLANIFICACION DE PRODUCCION.CASO PRACTICO: PLANTA LACTEOS ESPOCH"

TESIS DE GRADO

Previa a la obtención del título de

INGENIERO EN SISTEMAS INFORMÁTICOS

Presentado por:

SILVIA PATRICIA PINDUISACA ESPARZA

EDISON OMAR CAJAS MONTERO

RIOBAMBA – ECUADOR

2009

Agradecimiento

Agradecemos primero a Dios por darnos la vida, por ser nuestra gruía e iluminarnos durante todo nuestro camino. A nuestros Padres quienes han sido el motor que nos alienta diariamente para así poder concluir un sueño más, les agradecemos por confiar siempre en nosotros, respetar nuestras ideas, apoyarnos y levantarnos de nuestras caídas y celebrar siempre nuestros logros.

A nuestros amigos que de una u otra manera nos apoyaron en esta ardua tarea de ser estudiante, a todos aquellos que nos supieron brindar su ayuda y respeto.

A nuestro Director y miembros de Tesis: Dr. Julio Santillán, Ing. Iván Menes, por sus consejos y enseñanzas que en cada momento nos permitió crecer intelectualmente y profesionalmente, además al Ing. Marco Manzano y Dr. Alonso Álvarez quienes dedicaron su valioso tiempo en el desarrollo y perfeccionamiento de este trabajo que hoy presentamos muy orgullosamente.

Dedicatoria

Dedico el esfuerzo y fruto de este trabajo a Dios, el principal sentido de mi vida, por quien todas las cosas pueden ser hechas y el único dueño de la sabiduría absoluta, por la salud brindada y por la valentía prestada en los momentos cuando lo necesité, porque su amor es grande y su bondad infinita.

A mis padres Arturo y María, y a mis hermanas, por su apoyo moral y económico, por estar conmigo en las buenas y en las malas, por sus consejos en aquellos días donde la fe se desvanecía y por sus abrazos cuando necesitaba fuerza, a ustedes quienes han sido y serán siempre la luz de mi vida.

Al amor de mi vida Héctor, por apoyarme y enseñarme a creer en mí, por su paciencia, amor y ejemplo, por ser la bendición más grande para mi vida.

A mis familiares y amigos, por su comprensión y cariño incondicional.

Silvia

Dedico mi mayor esfuerzo a Dios quien ha estado a mi lado en todas las etapas de mi vida y me ha enseñado a resistir el dolor, a mi madre Sra. Nelly Montero la persona que ha sabido como sortear los problemas de la vida quien sin su esfuerzo y apoyo no sería la persona que soy y seré, a María Elena Jiménez la mujer que ha sido el apoyo incondicional de mi corazón, por último a mi familia y amigos personas que directa o indirectamente me apoyaron para llegar a culminar mi carrera.

Edison Cajas M.

Por último os digo: "Perece la riqueza perecen los parientes de igual forma debe el hombre perecer, pero aquí y en todo lugar sobrevive la fama de quien es justo merecedor a ella". [Erick el Rojo]

NOMBRE	**FIRMA**	**FECHA**
Dr. Romeo Rodríguez DECANO FACULTAD INFORMÁTICA Y ELECTRÓNICA	_____	_____
Ing. Iván Menes DIRECTOR DE LA ESCUELA DE INGENIERÍA EN SISTEMAS	_____	_____
Dr. Julio Santillan DIRECTOR DE TESIS	_____	_____
Ing. Iván Menes MIEMBRO DEL TRIBUNAL	_____	_____
Ing. Eduardo Tenelanda DIRECTOR CENTRO DE DOCUMENTACIÓN	_____	_____

NOTA DE LA TESIS _____

“Nosotros Silvia Patricia Pinduisaca Esparza y Edison Omar Cajas Montero, somos los responsables de las ideas, doctrinas y resultados expuestos en esta Tesis de Grado, y el patrimonio intelectual de la misma pertenece a la ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO.”

_____	_____
Silvia Patricia Pinduisaca Esparza	Edison Omar Cajas Montero

ÍNDICE DE ABREVIATURAS

POO. Programación Orientada a Objetos.

LP. Lenguaje de Programación

OO. Orientado a Objetos

RT. Requerimiento Tecnológico

RF. Requerimiento Funcional

CI. Cédula de Identidad

DB. DataBase (Base de Datos)

DBMS. Database Management System (Sistema administrador de bases de datos).

SQL. Structured Query Language (Lenguaje Estructurado de Consultas).

LP. Lenguaje de Programación

PMP. Plan Maestro de Producción

MRP. Material Requierement Planning (Planificación de las necesidades de Material)

RIU. Requerimiento de interfaz de usuario

RIS. Requerimiento de Interfaz software

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La presente tesis fue desarrollada con el fin de determinar a través del estudio comparativo de los paradigmas de programación, aquel que resuelva un modelo matemático para planificación de producción optimizando los recursos de nuestro computador.

Los paradigmas seleccionados para el estudio comparativo fueron el estructurado, orientado a objetos y el funcional, de los cuales se estudiaron características, ventajas, desventajas y posteriormente se implemento uno de los módulos que forman parte del sistema de Control de Inventarios y Planificación de Producción de la Planta de Lácteos de la ESPOCH, para de esta manera conseguir medir la cantidad de recursos que cada paradigma utiliza al momento de resolver el modelo matemático para maximizar la utilidad de la planta.

El modelo matemático antes mencionado fue construido específicamente para la planta, realizando el estudio del entorno de producción y comercialización de la misma. Para determinar la técnica matemática de resolución se realizó un estudio bibliográfico de los modelos matemáticos y técnicas de resolución para planificación de producción, de los cuales se seleccionó uno de los que consideramos según nuestro criterio, el más acoplable a las condiciones de la planta.

A continuación, se implemento en el paradigma resultante del estudio comparativo, el Sistema de Control de Inventario y Planificación de Producción para la planta de lácteos de la ESPOCH, el mismo que intenta llevar el Control de entrada y salida de insumos, control de los requerimientos de materia prima e insumos para la producción de determinada cantidad de productos, control de ventas, planificación de las cantidades que se debe elaborar de cada uno de los productos para lograr obtener un utilidad mayor, y los reportes necesarios para llevar el control de las actividades de la planta; proporcionando de esta manera una herramienta útil y eficiente al momento de administrar la planta y logrando ser un apoyo al momento de futuras auditorias.

Justificación del Proyecto de Tesis

El objetivo del estudio, es determinar el paradigma de programación que incorpore ventajas desde el punto de vista de eficiencia de líneas de código y optimización de recursos hardware.

A través de la desarrollo del modulo que implementa el modelo matemático construido para la planta de lácteos de la ESPOCH, lograremos determinar el paradigma que optimice los recursos computacionales y con la implementación del resto de módulos se logrará brindar a la planta, las siguientes ventajas:

Dar un enfoque más objetivo, sensible y disciplinado a determinar los requerimientos de materiales de la planta.

Para ello el sistema trabaja con dos parámetros básicos: tiempos y capacidades.

Calcular las cantidades de producto terminado a fabricar,

Los insumos necesarios y la materia prima a comprar para poder satisfacer la demanda del mercado.

Obteniendo los siguientes resultados:

- El plan de producción que especifica las cantidades a fabricar.
- La lista de insumos de los que se debe realizar el aprovisionamiento de las compras a los proveedores
- Reporte de compras de insumos y materia prima
- Reporte de Ventas
- Lista de órdenes de producción
- Lista de órdenes de producción terminada

Proporcionando beneficios como:

- Satisfacción del cliente
- Disminución del stock
- Reducción de las horas extras de trabajo
- Incremento de la productividad
- Menores costos, con lo cual, aumento en las utilidades

- Coordinación en la programación de producción e inventarios
- Posibilidad de conocer rápidamente las consecuencias financieras de nuestra planificación.

Definición de Objetivos

Objetivo General

Realizar un Estudio Comparativo de los Paradigmas Tradicionales de Programación frente al Paradigma Funcional para desarrollar una aplicación que construya modelos matemáticos de manera más eficiente.

Objetivos Específicos

- Estudiar los paradigmas de programación Estructurada, Orientada a Objetos y Funcional para determinar sus principales características.
- Determinar los parámetros a ser evaluados entre el Paradigma de Programación Estructurada, Orientada a Objetos y Funcional para la construcción de modelos matemáticos.
- Seleccionar el paradigma de programación óptimo para construir el modelo matemático para planificación de producción de la planta de lácteos de la ESPOCH de forma más eficiente.
- Implementar una aplicación para la planificación de la producción, aplicado a la planta de lácteos de la ESPOCH.

Planteamiento de la Hipótesis

"El paradigma resultante del estudio comparativo aplicado en la construcción de modelos matemáticos permitirá la optimización de recursos computacionales, aplicados en la planificación de la producción para la planta de lácteos de la ESPOCH."

CAPÍTULO I

PARADIGMAS DE PROGRAMACIÓN ESTRUCTURADA

1.1. Paradigma de Programación Estructurada.

1.1.1. Introducción

Durante los años 60, el desarrollo de software se encontró con severas dificultades. Los programas de entrega del software se retrasaban, sus costos excedían en gran medida los presupuestos, y los productos terminados no eran confiables. La actividad de investigación dio como resultado la evolución de la programación estructurada, un método disciplinado de escribir programas que sean claros, que se demuestre que son correctos y fáciles de modificar. Uno de los resultados más tangibles de esta investigación, fue el desarrollo en 1971 hecho por el profesor Nicklaus Wirth[1] del lenguaje de programación Pascal. Pascal fue diseñado para la enseñanza de la programación estructurada en entornos académicos y se convirtió con rapidez en el lenguaje introductorio de programación de la mayor parte de las universidades.

[1] Winterthur Suiza, 15 de febrero de 1934. Científico de la computación.

La programación estructurada nació como solución a los problemas que presentaba la programación no estructurada, la cual se empleó durante mucho tiempo antes de la invención de la misma.

Los programas computarizados pueden ser escritos con un alto grado de estructuración, lo cual les permite ser más fácilmente comprensibles en actividades tales como pruebas, mantenimiento y modificación de los mismos. Mediante la programación estructurada todas las bifurcaciones de control de un programa se encuentran estandarizadas, de forma tal que es posible leer la codificación del mismo desde su inicio hasta su terminación en forma continua, sin tener que saltar de un lugar a otro del programa siguiendo el rastro de la lógica establecida por el programador, como es la situación habitual con codificaciones desarrolladas bajo otras técnicas.

1.1.2. Definición Lógica

La programación estructurada es una teoría de programación que consiste en construir programas de fácil comprensión, es especialmente útil, cuando se necesitan realizar correcciones o modificaciones después de haber concluido un programa o aplicación. Al haberse utilizado la programación estructurada, es mucho más sencillo entender la codificación del programa, que se habrá hecho en diferentes secciones.

Se basa en una metodología de desarrollo de programas llamada refinamiento sucesivo, el mismo consiste en plantear una operación como un todo e ir dividiendo en segmentos más sencillos o de menor complejidad. Una vez terminado todos los segmentos del programa, se procede a unificar las aplicaciones realizadas por el grupo de programadores. Si se ha utilizado adecuadamente este tipo de programación, esta integración debe ser sencilla y no presentar problemas al integrar la misma y de presentar algún problema, será rápidamente detectable para su corrección.

La representación grafica se realiza a través de diagramas de flujo, el cual representa el programa con sus entradas, procesos y salidas.

En definitiva la programación estructurada propone segregar los procesos en estructuras lo más simple posibles, las cuales se conocen como secuencia, selección e interacción. Ellas

están disponibles en todos los lenguajes modernos de programación imperativa en forma de sentencias. Combinando esquemas sencillos se pueden llegar a construir sistemas amplios y complejos pero de fácil entendimiento.

1.1.3. Principios Utilizados por el diseño estructurado

Abstracción.- técnica de generalización que ignora u oculta detalles para capturar algo en común entre las diferentes instancias, con el propósito de controlar la complejidad intelectual de los sistemas de software.

Refinamiento sucesivo.- El refinamiento sucesivo es una primera estrategia de diseño descendente propuesta por Niklaus Wirth. La arquitectura de un programa se desarrolla en niveles sucesivos de refinamiento de los detalles procedimentales. Se desarrolla una jerarquía descomponiendo una declaración macroscópica de una función de una forma sucesiva, hasta que se llega a las sentencias del lenguaje de programación.

Modularidad.- el software se divide en componentes con nombres y ubicaciones determinados, que se denominan *módulos*, y que se integran para satisfacer los requisitos del problema.

Arquitectura del software.- La arquitectura del software se refiere a dos características importantes del software:

- La estructura jerárquica de los componentes procedimentales (módulos)
- La estructura de datos

Jerarquía de control.- La jerarquía de control, también denominada estructura de programa, representa la organización (frecuentemente jerárquica) de los componentes del programa (módulos) e implica una jerarquía de control. No representa aspectos procedimentales del software, tales como secuencias de procesos, o la repetición de operaciones.

Estructura de datos.- es una representación de la relación lógica existente entre los elementos individuales de datos. Debido a que la estructura de la información afectará invariablemente al diseño procedimental final, la estructura de datos es tan importante como la estructura del programa en la representación de la arquitectura del software.

Procedimientos del software.- se centra sobre los detalles de procesamiento de cada módulo individual. El procedimiento debe proporcionar una especificación precisa del procesamiento, incluyendo la secuencia de sucesos, los puntos concretos de decisiones, la repetición de operaciones, e incluso la organización o estructura de los datos.

Ocultamiento de la información.- El principio de ocultamiento de la información sugiere que los módulos se han de caracterizar por decisiones de diseño que los oculten unos a otros. Los mismos deben especificarse y diseñarse de forma que la información (procedimientos y datos) contenida dentro de un módulo sea accesible a otros únicamente a través de las *interfaces* formales establecidas para cada módulo.

1.1.4. Resolución de problemas

El proceso de resolución de problemas con computadoras conduce a la escritura de un programa y su ejecución en la misma. Aunque el proceso de diseñar programas es esencialmente un proceso creativo, se puede considerar una serie de fases o pasos comunes que generalmente deben seguir los programadores.

Estas fases son las siguientes:

- **Definición del Problema**: Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.
- **Análisis del Problema**: Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:
    - Los datos de entrada.
    - Cuál es la información que se desea producir (salida).
    - Los métodos y fórmulas que se necesitan para procesar los datos.

Una recomendación muy práctica es el que nos pongamos en el lugar de la computadora y analicemos que es lo que necesitamos que nos ordenen y en que secuencia para producir los resultados esperados.

Para un determinado problema se pueden diseñar distintos algoritmos y consecuentemente, distintos programas, la elección del algoritmo más adecuado se puede basar en una serie de características que adquieren gran importancia a la hora de evaluar el coste de diseño y mantenimiento, las características generales que debe reunir un programa y por lo tanto su algoritmo son las siguientes:

- Legibilidad, es importante que todo programa sea fácil de leer y entender.
- Portabilidad, el diseño del algoritmo debe permitir la codificación en cualquier lenguaje de programación, así como la instalación del programa en distintos sistemas.
- Modificable, debe ser fácil realizar modificaciones y actualizaciones sobre el programa, es decir, debe ser fácil realizar el mantenimiento del mismo.
- Eficiencia, se debe aprovechar al máximo los recursos del ordenador, especialmente la memoria utilizada, de igual forma, se debe minimizar el tiempo de ejecución.
- Modularidad, el programa debe estar dividido de una forma inteligente en módulos, cada uno de los cuales realice una parte del proceso de resolución del problema.
- Estructuración, el programa debe cumplir las reglas de la Programación Estructurada, para facilitar la depuración y mantenimiento del mismo.

Codificación.- La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora, a estas instrucciones se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

Prueba y Depuración.- Los errores humanos dentro de la programación de computadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración. La depuración o prueba resulta una tarea tan creativa como el mismo desarrollo de la solución, por ello se debe considerar con el mismo interés y entusiasmo.

Documentación.- Es la guía o comunicación escrita es sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas. A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones.

La documentación se divide en tres partes:

- **Documentación Interna**: son los comentarios o mensajes que se añaden al código fuente para hacer más claro el entendimiento de un proceso.
- **Documentación Externa**: se define en un documento escrito los siguientes puntos:
  - Descripción del Problema.
  - Nombre del Autor.
  - Algoritmo (diagrama de flujo o pseudocódigo).
  - Diccionario de Datos.
  - Código Fuente (programa).
- **Manual del Usuario**: describe paso a paso la manera cómo funciona el programa, con el fin de que el usuario obtenga el resultado deseado.

Mantenimiento.- Se lleva a cabo después de terminado el programa, cuando se detecta que es necesario hacer algún cambio, ajuste o complementar al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa este correctamente documentado.

1.1.5. Lenguajes de programación estructurada

Pascal

Pascal es un lenguaje de programación desarrollado por el profesor suizo Niklaus Wirth a finales de los años 60. Su objetivo era crear un lenguaje que facilitara el aprendizaje de la programación a sus alumnos. Sin embargo con el tiempo su utilización excedió el ámbito académico para convertirse en una herramienta para la creación de aplicaciones de todo tipo.
Pascal se caracteriza por ser un lenguaje de programación estructurado fuertemente tipificado[2].
Esto implica que:

- El código está dividido en porciones fácilmente legibles llamadas *funciones o procedimientos*. De esta forma *Pascal* facilita la utilización de la *programación estructurada* en oposición al antiguo estilo de *programación monolítica*.

[2] Cuando un lenguaje de programación es fuertemente tipificado, significa que cada variable que se usa es verificada en tipo y rango durante todo el programa, esto evita errores de asignación.

- El *tipo de dato* de todas las variables debe ser declarado previamente para que su uso quede habilitado.

El nombre de Pascal fue escogido en honor al matemático Blaise Pascal.

PL/1

PL/1, acrónimo de Programming Language 1 (Lenguaje de Programación 1), fue propuesto por IBM hacia 1970 para responder simultáneamente a las necesidades de las aplicaciones científicas y comerciales, disponible en las novedosas plataformas de utilidad general IBM 360 y más adelante IBM 370.

Este lenguaje tenía muchas de las características que más adelante adoptaría el lenguaje C[3] y algunas de C++[4]. Por desgracia, IBM registra el nombre del lenguaje como forma de mantener control sobre su desarrollo, lo que disuadió a otras empresas de dar ese nombre a sus implementaciones. No siendo posible encontrar un único lenguaje para diversas plataformas, los potenciales usuarios del lenguaje prefirieron no adoptarlo a pesar de sus múltiples innovaciones, que incluían multiprocesamiento, recursión, estructuras de control modernas, facilidades para la puesta a punto, asignación dinámica de espacio para estructuras de datos, procedimientos genéricos, etc.

Sin embargo, dentro de los usuarios de IBM, el lenguaje se utilizó con bastante intensidad, y el proyecto Multics utilizó PL/1 como lenguaje de desarrollo para su sistema de operación.

PL/1 fue probablemente el primer lenguaje comercial cuyo compilador estaba escrito en el lenguaje que compilaba.

1.1.6. Estructuras de control de flujo

Un programa propio contempla dos segmentos básicos:

- Tiene exactamente un punto de entrada y uno de salida

[3] Lenguaje de programación creado en 1972 en los Laboratorios Bell como evolución del anterior lenguaje B, orientado a la implementación de Sistemas Operativos, concretamente Unix.

[4] Lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos.

- Dentro de ese punto de entrada y salida hay trayectorias que conducen a cada parte del programa; esto significa que no existen bucles infinitos o una codificación inalcanzable.

El teorema de la estructura se refiere a que cualquier programa propio que se puede escribir usando solamente las tres estructuras de control: secuencia, selección e iteración.

Secuencia

Indica que las instrucciones de un programa se ejecutan una después de la otra, en el mismo orden en el cual aparecen en el programa. Se representa gráficamente como una caja después de otra, ambas con una sola entrada y una única salida.

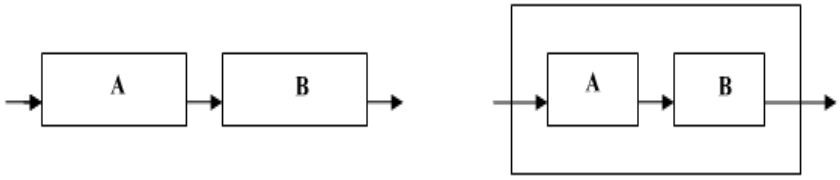

Figura I.1 Secuencia de un Programa

A y B pueden ser simples instrucciones hasta módulos completos. A y B deben ser ambos programas propios en el sentido ya definido de entrada y salida. La combinación de A y B es también un programa propio y que tiene también una entrada y una salida.

Selección

También conocida como la estructura SI-CIERTO-FALSO, plantea la selección entre dos alternativas con base en el resultado de la evaluación de una condición o predicado, equivale a la instrucción IF de todos los lenguajes de programación y se representa gráficamente de la siguiente manera:

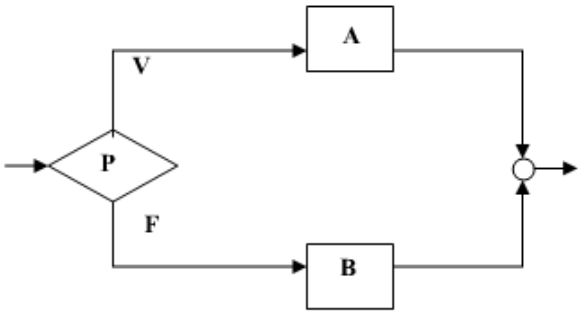

Figura I.2 Condición de Selección

En el diagrama de flujo anterior, P es una condición que se evalúa; A es la acción que se ejecuta cuando la evaluación de este predicado resulta verdadera y B es la acción ejecutada cuando indica falso. La estructura también tiene una sola entrada y una sola salida y las funciones A y B también pueden ser cualquier estructura básica o conjunto de estructuras.

- **La estructura de selección si...entonces (if)**

Dado que las expresiones lógicas toman el valor verdadero y falso, se necesita una sentencia de control para la toma de decisiones, cuando se desea ejecutar una acción si una expresión es verdadera o falsa.

Para ello utilizaremos la sentencia de selección if (si), el enunciado en pseudocódigo es el siguiente:

Tabla I.1 Sentencia IF Simple

```
si (exp. lógica simple o compuesta)
    acciones a ejecutar
fin_si
```

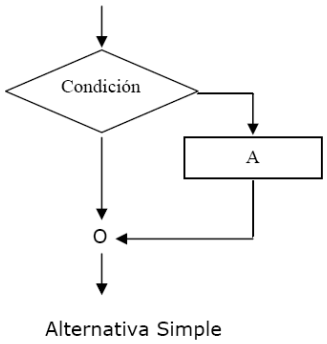

Figura I.3 Sentencia IF Simple

Cuando la expresión lógica contenida por los paréntesis es verdadera, se ejecutan las instrucciones dentro de la estructura de selección, cuando es falsa, el programa ignora la estructura y se sigue ejecutando la instrucción siguiente a la estructura de control.

- **Estructura de selección si/sino (if/else)**

La estructura de selección Si ejecuta una acción indicada solo cuando la condición es verdadera, de lo contrario la acción es pasada por alto. La estructura de selección si/sino (en algunos textos de programación puede aparecer como si/de_lo_contrario) permite que el programador especifique la ejecución de una acción distinta cuando la condición es falsa. Por ejemplo, el enunciado en seudocódigo:

Tabla I.2 Sentencia IF Doble

```
Si (nota >= 60) entonces
    escribir ("Aprobado")
sino (de lo contrario)
    escribir ("No Aprobado")
fin_si
```

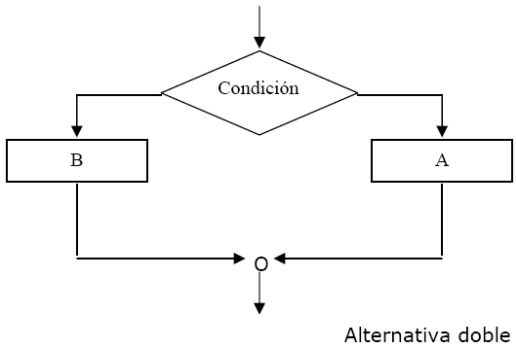

Figura I.4 Sentencia IF Doble

Imprime "Aprobado", si la calificación del alumno es mayor o igual a 60, e imprime "No aprobado" si la calificación es menor que 60. En cualquiera de los casos, después de haber impreso alguno de los mensajes, el programa ejecutará el enunciado siguiente al si.

- **Sentencias selectivas anidadas**

Dentro de las sentencias que figuran dentro de una sentencia if, pueden colocarse también otras sentencias selectivas. De esta manera:

Supongamos que deseamos imprimir en pantalla la nota de un alumno, clasificándolo en "aprobado", "no aprobado", y "deficiente". El algoritmo quedaría de esta manera.

Tabla I.3 Sentencia de Selección Múltiple

```
si (nota >= 60) entonces
    escribir ("aprobado")
sino
 si (nota < 60) and (nota >= 30) entonces
      escribir ("no aprobado")
 sino
   si (nota < 30) entonces
        escribir ("deficiente")
   fin_si
 fin_si
fin_si.
```

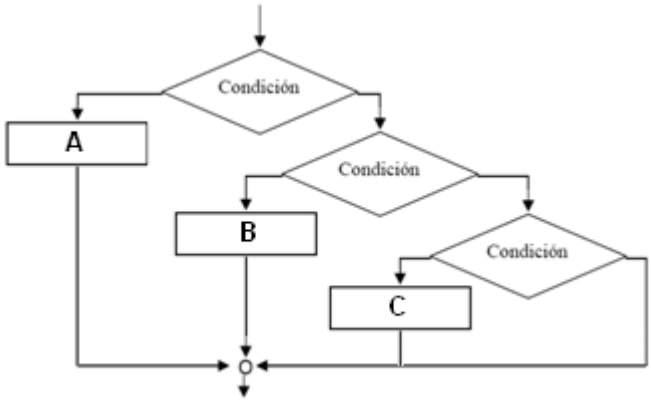

Figura I.5 Sentencia Selección Múltiple

Es muy importante que se utilice un buen sangrado en cada sentencia selectiva, para que sea más legible el código, y además que se comente el mismo, para de esta manera lograr tener organizado y también por si alguna otra persona, desea actualizarlo.

- **Sentencias de selección según sea (case)**

Esta sentencia se utiliza para elegir entre diferentes alternativas. Esta se compone de varias sentencias simples, cuando se ejecuta, una y solo una de las sentencias simples se selecciona y ejecuta.

La sintaxis es la siguiente:

Tabla I.4 Sentencia de Selección CASE

```
Según sea (selector) hacer
    caso1, caso2,..:  sentencia1
    .... : .....
    caso1n, caso2n,.:  sentencian
sino
    sentencia opcional
fin_según
```

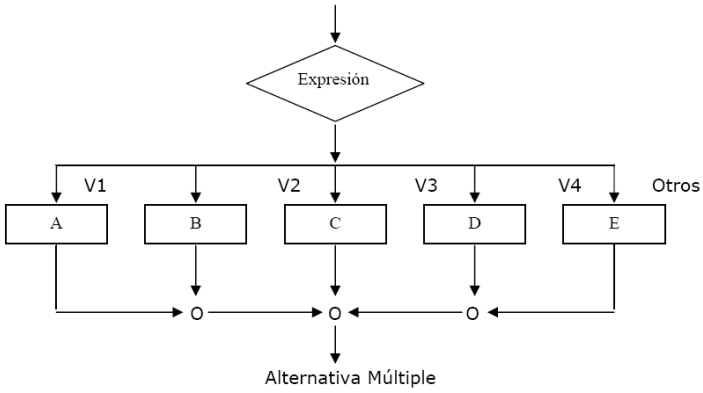

Figura I.6 Sentencia de Selección CASE

El valor de selector debe ser un número entero, y los valores constantes deben tener el mismo tipo que el selector.

Se pueden utilizar sentencias if anidadas, pero la sentencia según sea (case) es mas legible.

Iteración

También llamada la estructura HACER-MIENTRAS-QUE, corresponde a la ejecución repetida de una instrucción mientras que se cumple una determinada condición. El diagrama de flujo

para esta estructura es el siguiente: Repetir varias veces una acción hasta cuando deje de cumplirse la condición.

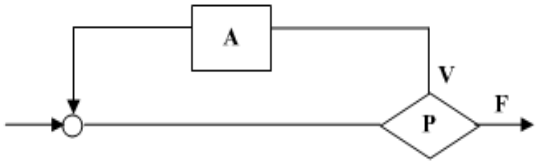

Figura I.7 Sentencias de Iteración o Bucle

Las computadoras están especialmente diseñadas para ejecutar tareas repetidamente. Las estructuras de control repetitivas son aquellas en las que una sentencia o grupos de sentencias se repiten muchas veces.

Una estructura de control que permite la repetición de una serie determinada de sentencias se denomina bucle (lazo o ciclo). El cuerpo del bucle contiene las sentencias que se repiten.

La acción o acciones que se repiten en un bucle se denominan el cuerpo del bucle, y cada repetición del cuerpo del bucle se denomina iteración.

- **Sentencia Mientras Hacer (while)**

La estructura repetitiva mientras es aquella en la que el número de iteraciones no se conoce por anticipado y el cuerpo del bucle se repite *mientras* se cumple una determinada condición. Por esta razón a estos bucles se les denomina bucles condicionales.

La sintaxis es la siguiente:

Tabla I.5 Sentencia de Iteración while

```
Mientras (condicion) hacer
 sentencias
fin_mientras
```

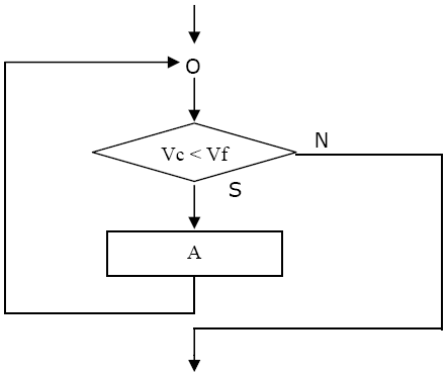

Figura I.8 Sentencia de Iteración while

Cuando la sentencia mientras se ejecuta, el primer paso es la evaluación de la expresión lógica. Si se evalúa en falso, ninguna acción se realiza y el programa prosigue en la siguiente sentencia después del bucle. Si se evalúa en verdadera, entonces se ejecuta las sentencias contenidas dentro del cuerpo del bucle y se evalúa de nuevo la expresión. Este proceso se repite *mientras* que la expresión lógica sea verdadera. Después de cada iteración, la expresión se evalúa y se verifica de nuevo, y si es verdadera, el bucle se repite de nuevo; si cambia de verdadera a falsa, la sentencia *mientras* finaliza y el programa prosigue en la siguiente sentencia del programa.

- **Sentencia repetir mientras (do while)**

Una variante de la sentencia *mientras*, es la sentencia repetir. Una de las características de la sentencia mientras es que la condición lógica se evalúa al principio de cada iteración. En particular, si la condición es falsa cuando la sentencia comienza, entonces el bucle no se ejecuta nunca. La sentencia repetir…mientras, especifica un bucle condicional que se repite mientras que la condición sea verdadera. Después de cada iteración el cuerpo del bucle evalúa la condición. Si la condición es verdadera, el bucle repite, ejecutándose la siguiente sentencia. Si la condición es falsa el bucle sale.

El seudocódigo es el siguiente:

Tabla I.6 Sentencia de Iteración do...while

```
repetir
 sentencias
mientras (expresión lógica)
```

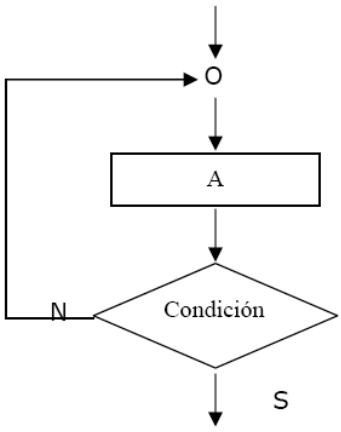

Figura I.9 Sentencia de Iteración do...while

- **Sentencia Para (For)**

En numerosas ocasiones se desea un bucle que se ejecute un número deseado de veces, y cuyo número de iteraciones se conozca por anticipado. Para este tipo de aplicaciones se utiliza la sentencia *para (for)*. La sentencia *para* requiere que conozcamos por anticipado el número de veces que se ejecutan las sentencias del interior del bucle.

El seudocódigo es el siguiente:

Tabla I.7 Sentencia For

```
Para cont ← (valor inicial)  hasta (valor final) hacer
sentencias
fin_para
```

Al ejecutarse la sentencia para (o desde) la primera vez, el valor inicial se asigna a cont, que se denomina *variable de control*, y a continuación se ejecuta la sentencia del interior del bucle. Al llegar al final del bucle se verifica si el valor final es mayor que el valor inicial; en caso negativo se incrementa el valor de la variable de control en uno y se vuelven a ejecutar todas las sentencias del interior del bucle, hasta que la variable de control sea mayor que el valor final, en cuyo momento se termina.

1.1.7. Ventajas de la programación estructurada

- Los programas son más fáciles de entender. Un programa estructurado puede ser leído en secuencia, de arriba hacia abajo, sin necesidad de estar saltando de un sitio a otro en la lógica, lo cual es típico de otros estilos de programación.
- La estructura del programa es más clara puesto que las instrucciones están más ligadas o relacionadas entre sí, por lo que es más fácil comprender lo que hace cada función.
- Reducción del esfuerzo en las pruebas. El programa se puede tener listo para producción normal en un tiempo menor del tradicional, por otro lado, el seguimiento de las fallas debugging[5] se facilita debido a la lógica más visible, de tal forma que los errores se pueden detectar y corregir de una manera más sencilla.
- Reducción de los costos de mantenimiento.
- Los programas quedan mejor documentados internamente.
- La estructura del programa es clara puesto que las instrucciones están más ligadas o relacionadas entre sí.
- Programas más sencillos y más rápidos (ya que es más fácil su optimización).
- Los bloques de código son auto explicativos, lo que facilita a la documentación.
- Un programa escrito de acuerdo a estos principios no solamente tendrá una estructura sino también una excelente presentación.

1.1.8. Lenguaje de programación estructurada Pascal

Introducción

El lenguaje Pascal se creó en la década de los 70 con el objetivo de disponer de un lenguaje de programación de alto nivel y propósito general (se utiliza para gran diversidad de aplicaciones) orientado hacia los nuevos conceptos de programación, desarrollado por el profesor suizo Niklaus Wirth como un lenguaje para la enseñar la programación de modo

[5] Depuración. Corrección de errores en la programación.

disciplinado, fácil de aprender y con la complejidad suficiente para permitir una correcta preparación de los programadores futuros.

Una versión preliminar del lenguaje apareció en 1968 y el primer compilador[6] totalmente completo apareció a finales de 1970. Desde entonces muchos compiladores han sido construidos y están disponibles para diferentes máquinas. De entre todas ellas **Turbo Pascal** es sin duda la versión más importante, ofreciéndonos un entorno operativo de programación en la que se integran el mismo editor de programa, el compilador y un intérprete, perfectamente desarrollado para desarrollar programas en Pascal.

El lenguaje estándar presenta una serie de características que lo hacen el lenguaje perfecto para aquellas personas iniciadas en la programación:

- Excelente para el aprendizaje de la programación.
- Lenguaje de propósito general, es decir, se puede aplicar a gran diversidad de aplicaciones.
- Utilización de procedimiento (programación modular).
- Lenguaje estructurado, se utilizan secuencias de control de bifurcación y bucles(if, for, while, repeat) sin necesidad de la famosa instrucción GOTO tan utilizada en muchos lenguajes como BASIC.
- Soporta la recursividad[7], es decir, propiedad que tienen los procedimientos para llamarse a sí mismo.
- Tipo de datos simples y estructurados, así como definidos por el usuario.
- Posibilidad de trabajar con punteros (variables dinámicas), de este modo permite definir nuestras propias estructuras de datos (lista, pilas, colas, etc.).

Todas estas características del lenguaje Pascal se ven favorecidas con la utilización de Turbo Pascal 7.0 (última versión), que aunque trabaja en un entorno de MS-DOS es la perfecta herramienta para la programación en este lenguaje, bajo un entorno gráfico potente, fácil e idóneo para el aprendizaje de profesionales o aficionados a la programación

[6] programa que hace posible la traducción de un lenguaje de programación al medio que solo entiende el ordenador

[7] Técnica muy empleada que consiste en que una función se llame a sí misma.

Estructura de un programa pascal

- **Encabezado**

La cabecera es una sección obligatoria, debe figurar en todos los programas. Debe comenzar con la palabra reservada program seguida del nombre del programa y un ";". Con esto ya se cumplirían los requisitos mínimos que debe tener una cabecera, pero se puede y es muy recomendable incluir también un comentario. Este comentario lo utilizamos para documentar el programa, que es algo que la gente suele dejar en segundo plano, pero es de lo más importante en programación. En el comentario se debe incluir el mayor número de componentes de los que se citan a continuación:

  - Autor del programa
  - Versión actual
  - Fecha de inicio del programa
  - Fecha de la última modificación
  - Qué se pretende que haga el programa
  - Nombre del fichero fuente en el que se guarda
  - Otras cosas que ayuden a documentar el programa

- **Bloque de Declaraciones**

En esta sección el programador puede insertar variables de tipo global, además se ingresaran tipos, constantes, procedimientos y funciones, es decir, todo lo necesario para que un programador lo utilice en su aplicación.

- **Cuerpo de programa Principal**

También se le llama bloque del programa, y es junto con la cabecera, la única sección obligatoria en un programa Pascal. Debe comenzar y finalizar con las palabras reservadas begin y end respectivamente. Muy importante: Después de la palabra end, siempre tiene que ir un punto que indica el final del programa.

Entre begin y end se escriben una o más sentencias, ya sean simples o compuestas. Las sentencias pueden ser varias: asignaciones, llamadas a procedimientos y funciones, sentencias selectivas (sentencias if), sentencias iterativas (sentencias for, while).

Tabla I.8 Estructura de un Programa Pascal

Program identificador ; {cabecera opcional en Turbo Pascal}	Encabezado
Uses identificadores Label lista de etiquetas ; {sección de etiquetas} Const definiciones de constantes Type declaración de tipos de datos definidos por el usuario Var declaración de variables Procedure definiciones de procedimientos Function definiciones de funciones	Bloque de declaraciones
begin sentencias end.	Cuerpo del Programa Principal

Las cinco secciones de declaración Label, Const, Type y Procedure y/o Function, así como la cláusula Uses y Program, no tiene que estar presentes en todos los programas. Turbo Pascal es muy flexible al momento de escribir las secciones de declaración, ya que se pueden hacer en cualquier orden (en Pascal estándar ISO si se require este orden). Sin embargo es conveniente seguir el orden establecido, le evitará futuros problemas.

Ejemplo

Tabla I.9 Programa en Pascal

```
Program MiPrimerPrograma; {cabecera}
Uses
  Crt; {declaraciones}
Const
  iva =0.10;
```

```
Type
  cadena =string[35];
  meses  =1..12;
Var
  sueldo :real;
  numero :integer;
  nombre :cadena;
  Nmes   :meses;
begin
  ClrScr; {Limpia la pantalla}
  Write ('Ingrese su Nombre: ');
  {Visualiza información en pantalla}
  ReadLn(nombre);{Leer un dato del teclado}
  WriteLn ('Hola Mundo, ', nombre);
  {Visualiza información en pantalla}
  Readkey; {Espera la pulsación de una tecla}
  ClrScr
end.
```

En la mayoría de los programas de computador, es necesario manejar datos de entrada o de salida, los cuales necesitan almacenarse en la memoria principal del computador en el tiempo de ejecución. Para poder manipular dichos datos, necesitamos tener acceso a las localidades de memoria donde se encuentran almacenados; esto se logra por medio de los nombres de los datos o IDENTIFICADORES.

Las reglas para formar las variables e identificadores en Pascal son las siguientes:

- Pueden estar compuestos de caracteres alfabéticos, numéricos y el carácter de subrayado (_).
- Deben comenzar con un carácter alfabético o el carácter de subrayado.
- Puede ser de cualquier longitud (sólo los 63 primeros caracteres son significativos).
- No se hace distinción entre mayúsculas y minúsculas.

- No se permite el uso de los IDENTIFICADORES RESERVADOS en los nombres de variables, constantes, programas o sub-programas.

- **Funciones**

Una función es un modulo de un programa separado del cuerpo principal, que realiza una tarea específica **y que puede regresar** un valor a la parte principal del programa u otra función o procedimiento que la invoque. Las funciones nacen con el propósito de ser subprogramas que siempre tienen que devolver algún valor.

Tabla I.10 Ejemplo de Función en Pascal

```
function nombre (lista_parametros): tipo;
  const lista_ctes;
  type lista_tipos;
  var lista_vars;
  begin
    (* cuerpo de la función *)
    nombre := valor_devuelto;
  end;
```

La lista de parámetros (lista_parametros) está encerrada entre paréntesis porque es opcional como en los procedimientos.

La palabra **tipo** es el tipo de dato que devolverá la función. Así podemos dividir las funciones en lógicas (boolean), enteras (integer), reales (real) y de carácter (char)

Y al final del cuerpo de la función es obligatorio asignarle un valor del tipo devuelto al nombre de la función, porque como ya hemos dicho una función siempre devuelve un valor.

- **Procedimientos**

Un procedimiento es un subprograma que realiza una tarea específica. Para invocarlo, es decir, para hacer que se ejecute, basta con escribir su nombre en el cuerpo de otro procedimiento o en el programa principal. Pero, hay que tener muy en cuenta que su declaración debe hacerse antes de que sea llamado por otro módulo.

Las dos principales diferencias entre procedimientos y **funciones** son:

- Las funciones siempre devuelven un valor al programa que las invocó.
- Para llamar a un procedimiento se escribe su nombre en el cuerpo del programa, y si los necesita, se incluyen los parámetros entre paréntesis. Para invocar una función es necesario hacerlo en una expresión.

1.2. Paradigma de Programación Orientada a Objetos

1.2.1. Introducción

Actualmente una de las áreas más acogidas en la industria y en el ámbito académico es la orientación a objetos. La misma que promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software como la falta de portabilidad del código y reusabilidad, código que es difícil de modificar, ciclos de desarrollo largos y técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas: debe estar basado en objetos, basado en clases y capaz de tener herencia[8] de clases. Muchos lenguajes cumplen uno o dos de estos puntos y muy pocos cumplen los tres. La barrera más difícil de sortear es usualmente la herencia.

El concepto de Programación Orientada a Objetos (POO) no es nuevo, lenguajes clásicos como SmallTalk se basan en ella. Dado que la POO se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo.

El elemento fundamental de la POO es, como su nombre lo indica, el objeto. Podemos definir un **objeto** como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización.

[8] Propiedad que permite que los objetos sean creados a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo.

1.2.2. Características

La POO parece ser el paradigma de la programación actual, entrando a reemplazar las técnicas de programación estructurada que se desarrollaron a principios de los 70.

Basa su ideología en la funcionalidad empaquetada[9]. Por ejemplo, para armar una computadora se adquieren las diferentes piezas con ciertas propiedades (entradas, salidas, capacidad de almacenamiento, voltaje, etc.) y una cierta funcionalidad (Alimentar Potencia, almacenar, leer, guardar o desplegar información, reproducir o grabar sonidos, etc.). La POO surge de la misma idea: el programa está compuesto de objetos con ciertas propiedades y funciones. No importa si el objeto se construya o se adquiera, con tal de que el objeto satisfaga sus especificaciones, no le importará demasiado la forma en que este funcione. En este tipo de programación, lo que importa es lo que los objetos exponen.

De la misma forma que al armar una computadora no importa el interior de las piezas siempre que éstas hagan lo que de ellas se espera, la mayoría de los programadores no tienen que preocuparse de cómo funcionan internamente los *applets*, siempre que éstos hagan lo que se espera de ellos. La clave para ser más productivo en este tipo de programación es conseguir que los objetos sean lo más completos posible y lograr que los objetos y partes del programa les indiquen lo más posible lo que deben hacer.

Hasta el momento, las formas de programación utilizadas por los programadores son:

- La programación construyendo el programa instrucción a instrucción, utilizando las tres estructuras básicas de control (*Secuencial, Condicional e Iterativa*): PROGRAMACION IMPERATIVA[10].

[9] Poner que es funcionalidad empaquetada

[10] Conjunto de instrucciones que le indican al computador cómo realizar una tarea. La implementación de hardware de la mayoría de computadores es imperativa

- La programación construyendo un programa mediante un conjunto de funciones de orden superior, que se han definido previamente (*Subprograma*) y aplicando posteriormente la composición funcional y la recursión: PROGRAMACION FUNCIONAL.
- La programación construyendo un programa como un conjunto de asertos y reglas lógicas, que definen relaciones: PROGRAMACION LOGICA.
- La programación que ve un programa como un conjunto de objetos que se relacionan unos a otros enviándose mensajes: PROGRAMACION ORIENTADA A OBJETOS (POO).

Objeto

Un Objeto es un elemento real o abstracto que tiene un estado, un comportamiento y una identidad. Un objeto es, pues, una mesa, un alumno, etc., pues son elementos reales y están bien definidos. También lo puede ser un concepto abstracto como un elemento llamado "Ordenador" que es capaz de recibir un conjunto de números y ordenarlo ascendente o descendentemente.

Estructura de un Objeto

Un objeto puede considerarse como una especie de cápsula dividida en tres partes, cada uno de estos componentes desempeña un papel totalmente independiente.

Relaciones

Las relaciones permiten que el objeto se inserte en la organización y están formadas esencialmente por punteros a otros objetos. Las relaciones entre objetos son, precisamente, los enlaces que permiten a un objeto relacionarse con aquellos que forman parte de la misma organización.

Existen 2 tipos de relaciones:

Relaciones jerárquicas. Son esenciales para la existencia misma de la aplicación porque la construyen. Son bidireccionales, es decir, un objeto es padre de otro cuando el primer objeto se encuentra situado inmediatamente encima del segundo en la organización en la que ambos forman parte; asimismo, si un objeto es padre de otro, el segundo es hijo del primero. Una organización jerárquica simple puede definirse como aquella en la que un objeto puede tener

un solo padre, mientras que en una organización jerárquica compleja un hijo puede tener varios padres.

Relaciones semánticas. Se refieren a las relaciones que no tienen nada que ver con la organización de la que forman parte los objetos que las establecen. Sus propiedades y consecuencia solo dependen de los objetos en sí mismos (de su significado) y no de su posición en la organización.

Propiedades

Las propiedades distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización. Todo objeto puede tener cierto número de propiedades, cada una de las cuales tendrá, a su vez, uno o varios valores. En OOP, las propiedades corresponden a las clásicas "variables" de la programación estructurada. Son, por lo tanto, datos encapsulados dentro del objeto, junto con los métodos (programas) y las relaciones (punteros a otros objetos). Las propiedades de un objeto pueden tener un valor único o pueden contener un conjunto de valores más o menos estructurados (matrices, vectores, listas, etc.). Además, los valores pueden ser de cualquier tipo (numérico, alfabético, etc.) si el sistema de programación lo permite.

Pero existe una diferencia con las "variables", y es que las propiedades se pueden heredar de unos objetos a otros. En consecuencia, un objeto puede tener una propiedad de maneras diferentes:

- **Propiedades propias**. Están formadas dentro de la cápsula del objeto.
- **Propiedades heredadas.** Están definidas en un objeto diferente, antepasado de éste (padre, “abuelo", etc.). A veces estas propiedades se llaman propiedades miembro porque el objeto las posee por solo el hecho de ser miembro de una clase.

Métodos

Los métodos son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia. Es decir una operación que realiza acceso a los datos.

Podemos definir método como un programa procedimental o procedural escrito en cualquier lenguaje, que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes.

Son sinónimos de 'método' todos aquellos términos que se han aplicado tradicionalmente a los programas, como procedimiento, función, rutina, etc. Sin embargo, es conveniente utilizar el término 'método' para que se distingan claramente las propiedades especiales que adquiere un programa en el entorno OOP, que afectan fundamentalmente a la forma de invocarlo (únicamente a través de un mensaje) y a su campo de acción, limitado a un objeto y a sus descendientes, aunque posiblemente no a todos.

Si los métodos son programas, se deduce que podrían tener argumentos, o parámetros. Puesto que los métodos pueden heredarse de unos objetos a otros, un objeto puede disponer de un método de dos maneras diferentes:

- **Métodos propios**. Están incluidos dentro de la cápsula del objeto.
- **Métodos heredados.** Están definidos en un objeto diferente, antepasado de éste (padre, “abuelo", etc.). A veces estos métodos se llaman métodos miembro porque el objeto los posee por solo el hecho de ser miembro de una clase.

Encapsulamiento y ocultación

Se dice que un objeto está encapsulado cuando está protegido del acceso indiscriminado de cualquier persona. Así cuando se tiene en las manos una calculadora se ve que se puede encender o apagar, digitar números u operaciones, pero no se puede ver cómo realiza algún cálculo o cómo despliega los caracteres en la pantalla.

La encapsulación es el proceso que aplica el diseñador de un objeto para ocultar aquellos detalles del objeto que no son específicamente necesarios para su uso.

Los objetos son inaccesibles, e impiden que otros objetos, los usuarios, o incluso los programadores conozcan cómo está distribuida la información o qué información hay disponible. Esta propiedad de los objetos se denomina ocultación de la información.

Esto no quiere decir, sin embargo, que sea imposible conocer lo necesario respecto a un objeto y a lo que contiene. Si así fuera no se podría hacer gran cosa con él. Lo que sucede es que las peticiones de información a un objeto deben realizarse a través de mensajes dirigidos a él, con

la orden de realizar la operación pertinente. La respuesta a estas órdenes será la información requerida, siempre que el objeto considere que quien envía el mensaje está autorizado para obtenerla.

El hecho de que cada objeto sea una cápsula facilita enormemente que un objeto determinado pueda ser transportado a otro punto de la organización, o incluso a otra organización totalmente diferente que precise de él. Si el objeto ha sido bien construido, sus métodos seguirán funcionando en el nuevo entorno sin problemas. Esta cualidad hace que la OOP sea muy apta para la reutilización de programas.

Clase

Una *Clase* se describe normalmente como la plantilla o el proyecto a partir del cual se hace realmente el objeto. La forma normal de imaginarse las clases es pensando en ellas como la plantilla para hacer un billete, mientras que el objeto propiamente dicho es el billete obtenido con dicha plantilla. Cuando se crea un objeto a partir de una clase, se dice que el programador ha creado una *Instancia* de dicha clase.

Herencia

La capacidad de crear clases que descienden de otras clases (conocidas como *Superclases*) se conoce como *Herencia*. La finalidad de la herencia es facilitar la fabricación de código para tareas especializadas. Las variables de instancia y los métodos de las clases descendientes (llamadas *Subclases*) comienzan siendo las mismas.

A veces se permite ignorar alguno de los métodos, lo cual se denomina *Polimorfismo*. La idea que lo sustenta es que, aunque el mensaje puede ser el mismo, el objeto determina la forma en que responde. El polimorfismo puede aplicarse a cualquier método que se herede de una clase básica.

La herencia puede ser *Simple* o *Múltiple*. En el primer caso, cada subclase tiene una única Superclase de la que es derivada (aunque esta superclase puede ser una subclase de otra superior). Mientras que en la herencia múltiple, una clase hereda a la vez varias superclases.

Polimorfismo

Una de las características fundamentales de la POO es el polimorfismo, que no es otra cosa que la posibilidad de construir varios métodos con el mismo nombre, pero con relación a la

clase a la que pertenece cada uno, con comportamientos diferentes. Esto conlleva la habilidad de enviar un mismo mensaje a objetos de clases diferentes. Estos objetos recibirían el mismo mensaje global pero responderían a él de formas diferentes; por ejemplo, un mensaje "+" a un objeto ENTERO significaría suma, mientras que para un objeto STRING significaría concatenación ("pegar" strings uno seguido al otro)

Organización de los objetos

En principio, los objetos forman siempre una organización jerárquica, en el sentido de que ciertos objetos son superiores a otros de cierto modo.

Existen varios tipos de jerarquías: serán simples cuando su estructura pueda ser representada por medio de un "árbol". En otros casos puede ser más compleja.

En cualquier caso, sea la estructura simple o compleja, podrán distinguirse en ella tres niveles de objetos.

La raíz de la jerarquía. Se trata de un objeto único y especial. Este se caracteriza por estar en el nivel más alto de la estructura y suele recibir un nombre muy genérico, que indica su categoría especial, como por ejemplo objeto madre, Raíz o Entidad.

Los objetos intermedios. Son aquellos que descienden directamente de la raíz y que a su vez tienen descendientes. Representan conjuntos o clases de objetos, que pueden ser muy generales o muy especializados, según la aplicación. Normalmente reciben nombres genéricos que denotan al conjunto de objetos que representan, por ejemplo, VENTANA, CUENTA, FICHERO. En un conjunto reciben el nombre de clases o tipos si descienden de otra clase o subclase.

Los objetos terminales. Son todos aquellos que descienden de una clase o subclase y no tienen descendientes. Suelen llamarse casos particulares, instancias o ítems porque representan los elementos del conjunto representado por la clase o subclase a la que pertenecen.

Demonios

Es un tipo especial de métodos, relativamente poco frecuente en los sistemas de OOP, que se activa automáticamente cuando sucede algo especial. Es decir, es un programa, como los métodos ordinarios, pero se diferencia de estos porque su ejecución no se activa con un

mensaje, sino que se desencadena automáticamente cuando ocurre un suceso determinado: la asignación de un valor a una propiedad de un objeto, la lectura de un valor determinado, etc.
Los demonios, cuando existen, se diferencian de otros métodos porque no son heredables y porque a veces están ligados a una de las propiedades de un objeto, más que al objeto entero.

1.2.3. Leguajes de programación Orientado a Objetos

C++

Está basado en el lenguaje de programación C, el cual está a su vez basado en dos lenguajes muy primitivos (BCPL y B).
Sus fundadores fueron Martin Richards (1976) y Ken Thompson (1970) respectivamente. Dos ańos mas tarde de la creación de B Dennis Ritchie implementó el diseño de BCPL y B y creó C, que se dio a conocer por ser el lenguaje de programación de desarrollo de UNIX.
A principios de los años 80, Bjarne Stroustrup (de los laboratorios Bell) empezó a desarrollar C++, que recibiría formalmente su nombre a finales de 1983. En octubre de 1985, apareció la primera divulgación comercial del lenguaje y la primera edición del libro "The C++ Programming Language", escrito por el propio creador de C++.
La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.
Posteriormente se añadieron facilidades programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje multiparadigma.
Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT (enlace externo).
Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales.
C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos

automatismos trae, con lo que obliga a usar librerías de terceros, como por ejemplo Boost (enlace externo)

El nombre C++ fue propuesto por Rick Mascitti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, la expresión "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

C++ tiene varias características que otros lenguajes de programación no tienen. Las más destacadas son:

Programación orientada a objetos: La posibilidad de orientar la programación a objetos permite al programador diseñar aplicaciones desde un punto de vista más cercano a la vida real. Además, permite la reutilización del código de una manera más lógica y productiva.

Portabilidad: Un código escrito en C++ puede ser compilado en casi todo tipo de ordenadores y sistemas operativos sin hacer apenas cambios.

Brevedad: El código escrito en C++ es muy corto en comparación con otros lenguajes, sobretodo porque en este lenguaje es preferible el uso de caracteres especiales que las "palabras clave".

Programación modular: Un cuerpo de aplicación en C++ puede estar hecho con varios ficheros de código fuente que son compilados por separado y después unidos. Además, esta característica permite unir código en C++ con código producido en otros lenguajes de programación como Ensamblador o el propio C

Velocidad: El código resultante de una compilación en C++ es muy eficiente, gracias a su capacidad de actuar como lenguaje de alto y bajo nivel y a la reducida medida del lenguaje.

Java

Este lenguaje de programación posee una curva de aprendizaje muy rápida. Resulta relativamente sencillo escribir applets interesantes desde el principio. Todos aquellos familiarizados con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros. Debido a su semejanza con C y C++, y dado que la mayoría de la gente los conoce aunque sea de forma elemental, resulta muy fácil aprender Java. Los

programadores experimentados en C++ pueden migrar muy rápidamente a Java y ser productivos en poco tiempo.

Entre sus principales características están:

Orientado a objetos

Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos. La tendencia del futuro, a la que Java se suma, apunta hacia la programación orientada a objetos, especialmente en entornos cada vez más complejos y basados en red.

Distribuido

Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

Interpretado y compilado a la vez

Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los bytecodes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los bytecodes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time).

Robusto

Java fue diseñado para crear software altamente fiable. Para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores (la aritmética de punteros), ya que se ha prescindido por completo los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

Seguro

Dada la naturaleza distribuida de Java, donde las applets se bajan desde cualquier punto de la Red, la seguridad se impuso como una necesidad de vital importancia. A nadie le gustaría ejecutar en su ordenador programas con acceso total a su sistema, procedentes de fuentes desconocidas. Así que se implementaron barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

Indiferente a la arquitectura

Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows Nt, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución tan variopintos, el compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura diseñado para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.

Portable

La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.

Estas dos últimas características se conocen como la *Máquina Virtual Java* (JVM).

Rendimiento

En cuanto se refiere al rendimiento Java posee características como:

- ***Multihebra.-*** Hoy en día ya se ven como terriblemente limitadas las aplicaciones que sólo pueden ejecutar una acción a la vez. Java soporta sincronización de múltiples hilos de ejecución (*multithreading*) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas. Así, mientras un hilo se encarga de la

comunicación, otro puede interactuar con el usuario mientras otro presenta una animación en pantalla y otro realiza cálculos.

- ***Dinámico.-*** El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.
- ***Produce applets.-*** Java puede ser usado para crear dos tipos de programas: aplicaciones independientes y applets. Las aplicaciones independientes se comportan como cualquier otro programa escrito en cualquier lenguaje, como por ejemplo el navegador de Web HotJava, escrito íntegramente en Java. Por su parte, las applets son pequeños programas que aparecen embebidos en las páginas Web, como aparecen los gráficos o el texto, pero con la capacidad de ejecutar acciones muy complejas, como animar imágenes, establecer conexiones de red, presentar menús y cuadros de diálogo para luego emprender acciones, etc.

1.2.4. Evolución de los Lenguajes de Programación Orientados a Objetos

En los primeros días de la informática Programadores enviaban instrucciones binarias al computador

- Manipulaban directamente interrupciones en sus paneles frontales

Años 40 Lo procedural fue el paradigma principal de programación

- Define el programa como un algoritmo.
- Las computadoras eran utilizadas por los militares con propósitos militares.
- 1957 Aparece el lenguaje de programación FORTRAN (traducción de fórmulas)

Años 50 Aparecen el lenguaje de máquina y el lenguaje ensamblador

Después aparecieron los lenguajes de programación de alto nivel

Permitieron a los programadores distanciarse de las características arquitectónicas de una computadora.

Cada instrucción puede invocar varias instrucciones maquina.

Esto permite escribir software sin preocuparse de la maquina que ejecutara el programa

Aparecen los procedimientos que son secuencias de sentencias que se invocan por una sentencia.

Años 60 Aun el termino ciencia de la computación no era de uso común

- Una de las desventajas de la programación basada en procedimientos fue la aparición de pantallas gráficas y el interés de aplicaciones utilizando ventanas. El acceso a una interfaz gráfica de usuarios (GUI) donde un usuario puede moverse con facilidad alrededor de una sola ventana es un desafío cuando se intenta lograrlo con un código procedural.

Años 60 Nace la programación estructurada con el objetivo de dar soluciones o mejoras a la programación mediante procedimientos.

- Método disciplinado para escribir programas que son:
- Más claros
- Fáciles de probar y corregir
- Más fáciles de modificar que los no estructurados

Gracias a esta evolución aparecen conceptos como: estructuras de control, funciones y módulos.

Años 80 El desarrollo de la OOP empieza a destacar tomando en cuenta la programación estructurada, a la que engloba y dotando al programador de nuevos elementos para el análisis y desarrollo de software.

La POO modela objetos del mundo real

Toma ventaja de las relaciones entre clases, en donde los objetos de cierta clase tiene las mismas características y comportamientos.

Aparecen conceptos de Herencia, clase, objeto, polimorfismo, encapsulamiento

Gracias a la modularidad y a la herencia una aplicación diseñada bajo el paradigma de la orientación a objetos puede ser fácilmente extensible para cubrir necesidades de crecimiento de la aplicación.

1.2.5. Ventajas de la Programación Orientada a Objetos

- La programación orientada a objetos (POO) modela objetos del mundo real con contrapartes en software. Toma ventaja de las relaciones entre clases, en donde los objetos de cierta clase tiene las mismas características y comportamientos.
- Aprovecha las relaciones de herencia e incluso las relaciones de herencia múltiple, en donde las clases recién creadas se derivan mediante la herencia de las características de clases existentes, aunque contiene características únicas propias.
- La programación orientada a objetos proporciona una manera intuitiva de ver el proceso de programación, a saber, mediante el modelado de objetos reales, sus atributos y sus comportamientos.
- La POO también modela la comunicación entre objetos mediante mensajes.
- La POO encapsula los datos (atributos) y las funciones (comportamiento) dentro de los objetos.
- Los objetos tienen la propiedad de ocultar información. Aunque los objetos pueden saber cómo comunicarse entre sí a través de interfaces bien definidas, los objetos por lo general no están autorizados para saber los detalles de la implementación de otros objetos (para eliminar dependencias innecesarias). El ocultamiento de información es crucial para una buena ingeniería de software.

1.2.6. Lenguaje de programación Orientada a Objetos C#

Origen y necesidad de un nuevo lenguaje

C# es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET. Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.
Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes

ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el **lenguaje nativo de .NET**

La sintaxis y estructuración de C# es muy similar a la de C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

Un lenguaje que hubiese sido ideal utilizar para estos menesteres es Java, pero debido a problemas con la empresa creadora del mismo Sun, Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado añadirle para mejorarlo aún más y hacerlo un lenguaje orientado al desarrollo de componentes.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el *.NET Framework SDK*

Características de C#

Sencillez: C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET. Por ejemplo:

- El código escrito en C# es **auto contenido**, lo que significa que no necesita de ficheros adicionales al propio fichero fuente tales como ficheros de cabecera o ficheros IDL.
- El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o máquina para quienes se compile (no como en C++), lo que facilita la portabilidad del código.
- No se incluyen elementos poco útiles de lenguajes como C++ tales como macros, herencia múltiple o la necesidad de un operador diferente del punto (**.**) para acceder a miembros de espacios de nombres (**::**).

Modernidad: C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose son muy útiles para el desarrollo de aplicaciones y que en otros lenguajes

como Java o C++ hay que simular, como un tipo básico **decimal** que permita realizar operaciones de alta precisión con reales de 128 bits (muy útil en el mundo financiero), la inclusión de una instrucción **foreach** que permita recorrer colecciones con facilidad y es ampliable a tipos definidos por el usuario, la inclusión de un tipo básico **string** para representar cadenas o la distinción de un tipo **bool** específico para representar valores lógicos.

Orientación a objetos: Como todo lenguaje de programación de propósito general actual, C# es un lenguaje orientado a objetos, aunque eso es más bien una característica del CTS que de C#. Una diferencia de este enfoque orientado a objetos respecto al de otros lenguajes como C++ es que el de C# es más puro en tanto que no admiten ni funciones ni variables globales sino que todo el código y datos han de definirse dentro de definiciones de tipos de datos, lo que reduce problemas por conflictos de nombres y facilita la legibilidad del código.

C# soporta todas las características propias del paradigma de programación orientada a objetos: **encapsulación**, **herencia** y **polimorfismo**.

En lo referente a la encapsulación es importante señalar que aparte de los típicos modificadores **public**, **private** y **protected**, C# añade un cuarto modificador llamado **internal**, que puede combinarse con **protected** e indica que al elemento a cuya definición precede sólo puede accederse desde su mismo ensamblado.

Respecto a la herencia a diferencia de C++ y al igual que Java C# sólo admite herencia simple de clases ya que la múltiple provoca más dificultades que facilidades y en la mayoría de los casos su utilidad puede ser simulada con facilidad mediante herencia múltiple de interfaces. De todos modos, esto vuelve a ser más bien una característica propia del CTS que de C#.

Por otro lado y a diferencia de Java, en C# se ha optado por hacer que todos los métodos sean por defecto sellados y que los redefinibles hayan de marcarse con el modificador **virtual** (como en C++), lo que permite evitar errores derivados de redefiniciones accidentales. Además, un efecto secundario de esto es que las llamadas a los métodos serán más eficientes por defecto al no tenerse que buscar en la tabla de funciones virtuales la implementación de los mismos a la que se ha de llamar. Otro efecto secundario es que permite que las llamadas a los métodos virtuales se puedan hacer más eficientemente al contribuir a que el tamaño de dicha tabla se reduzca.

Orientación a componentes: La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. Es decir, la sintaxis de C# permite definir cómodamente **propiedades**, **eventos** o **atributos**.

Gestión automática de memoria: todo lenguaje de .NET tiene a su disposición el recolector de basura del CLR. Esto tiene el efecto en el lenguaje de que no es necesario incluir instrucciones de destrucción de objetos. Sin embargo, dado que la destrucción de los objetos a través del recolector de basura es indeterminista y sólo se realiza cuando éste se active ya sea por falta de memoria, finalización de la aplicación o solicitud explícita en el fuente, C# también proporciona un mecanismo de liberación de recursos determinista a través de la instrucción **using**.

Seguridad de tipos: C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que permite evitar que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente a ningún objeto y es especialmente necesario en un entorno gestionado por un recolector de basura. Por esta razón se toman medidas de los siguientes tipos.

- Sólo se admiten **conversiones entre tipos compatibles**. Esto es, entre un tipo y antecesores suyos, entre tipos para los que explícitamente se haya definido un operador de conversión, y entre un tipo y un tipo hijo suyo del que un objeto del primero almacenase una referencia del segundo (**downcasting**) Obviamente, lo último sólo puede comprobarlo en tiempo de ejecución el CLR y no el compilador, por lo que en realidad el CLR y el compilador colaboran para asegurar la corrección de las conversiones.
- No se pueden usar **variables no inicializadas**. El compilador da a los campos un valor por defecto consistente en ponerlos a cero y controla mediante análisis del flujo de control de fuente que no se lea ninguna variable local sin que se le haya asignado previamente algún valor.
- Se comprueba que todo **acceso a los elementos de una tabla** se realice con índices que se encuentren dentro del rango de la misma.

- Se puede controlar la **producción de desbordamientos** en operaciones aritméticas, informándose de ello con una excepción cuando ocurra. Sin embargo, para conseguirse un mayor rendimiento en la aritmética estas comprobaciones no se hacen por defecto al operar con variables sino sólo con constantes (se pueden detectar en tiempo de compilación)
- A diferencia de Java, C# incluye **delegados**, que son similares a los punteros a funciones de C++ pero siguen un enfoque orientado a objetos, pueden almacenar referencias a varios métodos simultáneamente, y se comprueba que los métodos a los que apunten tengan parámetros y valor de retorno del tipo indicado al definirlos.
- Pueden definirse métodos que admitan un número indefinido de parámetros de un cierto tipo, y a diferencia de lenguajes como C/C++, en C# siempre se comprueba que los valores que se les pasen en cada llamada sean de los tipos apropiados.

Instrucciones seguras: Para evitar errores muy comunes, en C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes. Por ejemplo, la guarda de toda condición ha de ser una expresión condicional y no aritmética, con lo que se evitan errores por confusión del operador de igualdad (**==**) con el de asignación (**=**); y todo caso de un **switch** ha de terminar en un **break** o **goto** que indique cuál es la siguiente acción a realizar, lo que evita la ejecución accidental de casos y facilita su reordenación.

Sistema de tipos unificado: A diferencia de C++, en C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada **System.Object**, por lo que dispondrán de todos los miembros definidos en ésta clase (es decir, serán "objetos")

A diferencia de Java, en C# esto también es aplicable a los tipos de datos básicos. Además, para conseguir que ello no tenga una repercusión negativa en su nivel de rendimiento, se ha incluido un mecanismo transparente de **boxing** y **unboxing** con el que se consigue que sólo sean tratados como objetos cuando la situación lo requiera, y mientras tanto puede aplicárseles optimizaciones específicas.

El hecho de que todos los tipos del lenguaje deriven de una clase común facilita enormemente el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.

Extensibilidad de tipos básicos: C# permite definir, a través de estructuras, tipos de datos para los que se apliquen las mismas optimizaciones que para los tipos de datos básicos. Es decir, que se puedan almacenar directamente en pila (luego su creación, destrucción y acceso serán más rápidos) y se asignen por valor y no por referencia. Para conseguir que lo último no tenga efectos negativos al pasar estructuras como parámetros de métodos, se da la posibilidad de pasar referencias a pila a través del modificador de parámetro **ref**.

Extensibilidad de operadores: Para facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través de las estructuras estén al mismo nivel que los básicos predefinidos en el lenguaje, al igual que C++ y a diferencia de Java, C# permite redefinir el significado de la mayoría de los operadores incluidos los de conversión, tanto para conversiones implícitas como explícitas cuando se apliquen a diferentes tipos de objetos.

Las redefiniciones de operadores se hacen de manera inteligente, de modo que a partir de una única definición de los operadores **++** y **--** el compilador puede deducir automáticamente como ejecutarlos de manera prefijas y postfija; y definiendo operadores simples (como **+**), el compilador deduce cómo aplicar su versión de asignación compuesta (+=) Además, para asegurar la consistencia, el compilador vigila que los operadores con opuesto siempre se redefinan por parejas (por ejemplo, si se redefine **==**, también hay que redefinir **!=**)

También se da la posibilidad, a través del concepto de **indizador**, de redefinir el significado del operador **[ ]** para los tipos de dato definidos por el usuario, con lo que se consigue que se pueda acceder al mismo como si fuese una tabla. Esto es muy útil para trabajar con tipos que actúen como colecciones de objetos.

Extensibilidad de modificadores: C# ofrece, a través del concepto de **atributos**, la posibilidad de añadir a los metadatos del módulo resultante de la compilación de cualquier fuente información adicional a la generada por el compilador que luego podrá ser consultada en tiempo de ejecución a través de la librería de reflexión de .NET . Esto, que más bien es una característica propia de la plataforma .NET y no de C#, puede usarse como un mecanismo para definir nuevos modificadores.

Versionable: C# incluye una **política de versionado** que permite crear nuevas versiones de tipos sin temor a que la introducción de nuevos miembros provoquen errores difíciles de

detectar en tipos hijos previamente desarrollados y ya extendidos con miembros de igual nombre a los recién introducidos.

- Si una clase introduce un nuevo método cuyas redefiniciones deban seguir la regla de llamar a la versión de su padre en algún punto de su código, difícilmente seguirían esta regla miembros de su misma signatura definidos en clases hijas previamente a la definición del mismo en la clase padre; o si introduce un nuevo campo con el mismo nombre que algún método de una clase hija, la clase hija dejará de funcionar. Para evitar que esto ocurra, en C# se toman dos medidas:
- Se obliga a que toda redefinición deba incluir el modificador **override**, con lo que la versión de la clase hija nunca sería considerada como una redefinición de la versión de miembro en la clase padre ya que no incluiría **override**. Para evitar que por accidente un programador incluya este modificador, sólo se permite incluirlo en miembros que tengan la misma signatura que miembros marcados como redefinibles mediante el modificador **virtual**. Así además se evita el error tan frecuente en Java de creerse haber redefinido un miembro, pues si el miembro con **override** no existe en la clase padre se producirá un error de compilación.
- Si no se considera redefinición, entonces se considera que lo que se desea es ocultar el método de la clase padre, de modo que para la clase hija sea como si nunca hubiese existido. El compilador avisará de esta decisión a través de un mensaje de aviso que puede suprimirse incluyendo el modificador **new** en la definición del miembro en la clase hija para así indicarle explícitamente la intención de ocultación.

Eficiente: En principio, en C# todo el código incluía numerosas restricciones para asegurar su seguridad y no permitir el uso de punteros. Sin embargo, y a diferencia de Java, en C# es posible saltarse dichas restricciones manipulando objetos a través de punteros. Para ello basta marcar regiones de código como inseguras (modificador **unsafe**) y podrán usarse en ellas punteros de forma similar a cómo se hace en C++, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad procesamiento muy grandes.

Compatible: Para facilitar la migración de programadores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C#

fragmentos de código escrito en estos lenguajes, sino que el CLR también ofrece, a través de los llamados **Platform Invocation Services** (**PInvoke**), la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos tales como las DLLs de la API Win32. Nótese que la capacidad de usar punteros en código inseguro permite que se pueda acceder con facilidad a este tipo de funciones, ya que éstas muchas veces esperan recibir o devuelven punteros.

También es posible acceder desde código escrito en C# a objetos COM. Para facilitar esto, el *.NET Framework SDK* incluye una herramientas llamadas **tlbimp** y **regasm** mediante las que es posible generar automáticamente clases proxy que permitan, respectivamente, usar objetos COM desde .NET como si de objetos .NET se tratase y registrar objetos .NET para su uso desde COM.

Finalmente, también se da la posibilidad de usar controles ActiveX desde código .NET y viceversa. Para lo primero se utiliza la utilidad **aximp**, mientras que para lo segundo se usa la ya mencionada **regasm**.

Ejemplo

Tabla I.11 Ejemplo de un Programa en C#

```
1:    class HolaMundo
2:    {
3:      static void Main()
4:      {
5:        System.Console.WriteLine("¡Hola Mundo!");
6:      }
7:    }
```

Todo el código escrito en C# se ha de escribir dentro de una definición de clase, y lo que en la línea **1:** se dice es que se va a definir una clase (**class**) de nombre HolaMundo cuya definición estará comprendida entre la llave de apertura de la línea **2:** y su correspondiente llave de cierre en la línea línea **7:**

Dentro de la definición de la clase (línea **3:**) se define un método de nombre Main cuyo código es el indicado entre la llave de apertura de la línea **4:** y su respectiva llave de cierre (línea **6:**) Un **método** no es más que un conjunto de instrucciones a las que se les asocia un nombre, de

modo que para posteriormente ejecutarlas baste referenciarlas por su nombre en vez de tener que reescribirlas.

La partícula que antecede al nombre del método indica cuál es el tipo de valor que se devuelve tras la ejecución del método, y en este caso es **void** que significa que no se devuelve nada. Por su parte, los paréntesis que se coloca tras el nombre del método indican cuáles son los parámetros que éste toma, y como en este caso están vacíos ello significa que el método no toma parámetros. Los parámetros de un método permiten variar el resultado de su ejecución según los valores que se les dé en cada llamada.

La palabra **static** que antecede a la declaración del tipo de valor devuelto es un **modificador** del significado de la declaración de método que indica que el método está asociado a la clase dentro de la que se define y no a los objetos que se creen a partir de ella. Main() es lo que se denomina el **punto de entrada** de la aplicación, que no es más que el método por el que comenzará su ejecución. Necesita del modificador **static** para evitar que para llamarlo haya que crear algún objeto de la clase donde se haya definido.

Finalmente, la línea **5:** contiene la instrucción con el código a ejecutar, que lo que hace es solicitar la ejecución del método **WriteLine()** de la clase **Console** definida en el espacio de nombres **System** pasándole como parámetro la cadena de texto con el contenido ¡Hola Mundo! Nótese que las cadenas de textos son secuencias de caracteres delimitadas por comillas dobles aunque dichas comillas no forman parte de la cadena. Por su parte, un espacio de nombres puede considerarse que es algo similar para las clases a lo que un directorio es para los ficheros; es decir, es una forma de agruparlas.

1.3. Paradigma de Programación Funcional

1.3.1. Introducción

El objetivo del paradigma funcional es conseguir lenguajes expresivos y matemáticamente elegantes, en los que no sea necesario bajar al nivel de la máquina para describir el proceso llevado a cabo por el programa, y evitando el concepto de estado de cómputo. La secuencia de computaciones llevadas a cabo por el programa se regiría única y exclusivamente por la

reescritura de definiciones más amplias a otras cada vez más concretas y definidas, usando lo que se denominan definiciones dirigidas.

Todo esto con el objetivo de familiarizar a los programadores con un lenguaje elegante en el cual se pueda manejar más fácilmente y así los programas sean menos extensos y complejos.

Otro de los objetivos primordiales de dicho paradigma es buscar satisfacer las necesidades del usuario con respecto a operaciones matemáticas y convertirse en un lenguaje más expresivo.

Sus orígenes provienen del Cálculo Lambda (o λ-cαlculo), una teoría matemática elaborada por Alonso Church como apoyo a sus estudios sobre Computabilidad. Un lenguaje funcional es, a grandes rasgos, un azúcar sintáctico del Cálculo Lambda.

Cálculo Lambda

Los orígenes teóricos del modelo funcional se remontan a la década del 30, más precisamente al año 1934, cuando Alonso Church introdujo un modelo matemático de computación llamado lambda calculo.

A pesar de que en esta época las computadoras aun no existían el lambda cálculo se puede considerar como el primer lenguaje funcional de la historia y sus fundamentos fueron la base de toda la teoría de la programación funcional y de los lenguajes funcionales desarrollados posteriormente. Se puede decir que los lenguajes funcionales modernos son versiones de lambda cálculo con numerosas ayudas sintácticas.

Aunque cuando aparece el cálculo lambda, aún no existían las computadoras, resulta ser una herramienta simple que se adelanta a su época, que abarca dos operaciones:

- Definir alguna(s) función(es) de un solo argumento y con un cuerpo específico, denotado por la siguiente terminología:
  - lx.B, en donde:
    - x: Define el parámetro o argumento formal.
    - B: Representa el cuerpo de la función.
    - Es decir f(x) = B.
- Reducción:
    - Consiste en aplicar alguna de las funciones creadas, sobre un argumento real (A); resultado de sustituir las ocurrencias del argumento formal (x), que

aparezcan en el cuerpo (B) de la función, con el argumento (A), es decir: (lx.B)

Ejemplo:

(lx.(x+5))3, indica que en la expresión x + 5, debemos sustituir el valor de x por 3.

Cuando ya no es posible reducir una función, se dice que ésta se encuentra en su estado normal, o sea hemos encontrado el valor de la función, que dependerá únicamente de los argumentos y siempre tendrá la consistencia de regresar el mismo valor para los mismos argumentos.

Lo anterior es la transferencia referencial y al no contar con variables globales, permiten al sistema la ejecución de procesos en forma paralela para incrementar su eficiencia.

Sobre estos simples conceptos está basada la programación funcional, aunque existen otros, usados para identificarla y aumentan su potencial en el desarrollo de aplicaciones.

Los dos mecanismos básicos presentados anteriormente se corresponden con los conceptos de abstracción funcional y aplicación de función; si le agregamos un conjunto de identificadores para representar variables se obtiene lo mínimo necesario para tener un lenguaje de programación funcional. Lambda calculo tiene el mismo poder computacional que cualquier lenguaje imperativo tradicional.

1.3.2. Características

El componente básico de los lenguajes funcionales es la noción de función y su estructura de control esencial la aplicación de una función.

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones, entendiendo éstas no como subprogramas clásicos de un lenguaje imperativo, sino como funciones puramente matemáticas, en las que se verifican ciertas propiedades como la transparencia referencial (el significado de una expresión depende únicamente del significado de sus subexpresiones), y por tanto, la carencia total de efectos laterales.

Otras características propias de estos lenguajes son la no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración; lo que

obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo por medio de funciones recursivas.

Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes procedimentales, como las secuencias de instrucciones o la asignación de variables.

En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial, algo que no se cumple siempre con un lenguaje funcional híbrido.

Entre los lenguajes funcionales puros, cabe destacar a Haskell y Miranda. Los lenguajes funcionales híbridos más conocidos son Lisp, Scheme, Ocaml y Standard ML (estos dos últimos, descendientes del lenguaje ML).

En definitiva la programación funcional, es un modelo basado en la evaluación de funciones matemáticas, entendidas como mecanismos para aplicar ciertas operaciones sobre algunos valores o argumentos, para obtener un resultado o valor de la función para tales argumentos.

Sin embargo, tanto argumentos como resultado de una función, pueden ser otra función, o incluso la misma, tal como una forma de recursividad, que constituye una poderosa herramienta de la programación funcional.

Debemos además tener en claro que la programación funcional es un modelo de programación algo diferente al que estamos habituados, no se parece mucho a la programación imperativa convencional, típica de C, Pascal, etc.

En programación funcional, como su nombre indica, todo son funciones, excepto los valores. No existen variables, ni procedimientos. Al realizar un programa será seguramente una función, que recibe unos parámetros, y devuelve un resultado.

Debido a que no existe la asignación de valores a variables, las mismas simplemente sirven para guardar un estado entre distintas instrucciones que se ejecutan secuencialmente. No existe la composición secuencial de instrucciones. Entonces las variables no hacen falta, porque una función empieza, realiza una transformación sobre los parámetros, y devuelve un

resultado. Precisamente se utiliza mucho el concepto de función recursiva, para realizar tareas que en programación convencional se realizan con bucles.

1.3.3. Lenguajes de Programación Funcional

Los matemáticos desde hace un buen tiempo están resolviendo problemas usando el concepto de función. Una función convierte ciertos datos en resultados. Si supiéramos cómo evaluar una función, usando la computadora, podríamos resolver automáticamente muchos problemas.

Así pensaron algunos matemáticos, que no le tenían miedo a la máquina, e inventaron los lenguajes de programación funcionales. Además, aprovecharon la posibilidad que tienen las funciones para manipular datos simbólicos, y no solamente numéricos, y la propiedad de las funciones que les permite componer, creando de esta manera, la oportunidad para resolver problemas complejos a partir de las soluciones a otros más sencillos.

También se incluyó la posibilidad de definir funciones recursivamente.

Un lenguaje funcional ofrece conceptos que son muy entendibles y relativamente fáciles de manejar para todos los que no se durmieron en las clases de matemáticas.

Entre los lenguajes funcionales se encuentran ISWIM, ML, LISP y todos sus derivados, como Scheme.

El valor de una expresión depende sólo de los valores de sus subexpresiones, si las tiene. La programación funcional pura es una programación sin asignaciones. En realidad, la mayoría de los lenguajes funcionales son impuros, ya que permiten asignaciones. Sin embargo, su estilo de programación es diferente al de los lenguajes de programación imperativa.

Algunas de las características que poseen los lenguajes funcionales son:

Almacenamiento implícito. El programador no debe preocuparse en manejar el almacenamiento de datos. Una consecuencia de esto es que la implementación del lenguaje debe realizar una "recolección de basura" para recuperar la memoria que se ha usado y no se volverá a utilizar.

Las funciones son valores de primera clase. Una función puede ser el valor de una expresión, pasarse como argumento o colocarse en una estructura de datos. Esto permite potentes operaciones.

Los lenguajes funcionales, son más cercanos a la manera en que funciona la mente humana, pues tienden a permitirles a los programadores describir sus algoritmos como expresiones que serán evaluadas.

Hay varias consecuencias del énfasis en la evaluación de expresiones que son comunes a la mayoría de lenguajes funcionales:

- Los procedimientos son ciudadanos de primera categoría: Los procedimientos y funciones son objetos visibles que pueden ser almacenados dentro de estructuras complejas, pasados como argumentos, construidos en tiempo de ejecución y manipulados al igual que otros tipos como los números y las cadenas.
- Manejo automático de memoria El programador no necesita llevar un registro manual de la memoria utilizada por cada objeto y liberarla; la implementación del lenguaje se encarga de eso de manera automática.
- Los lenguajes funcionales han permanecido en uso por mucho tiempo y han mostrado todo el poder de su gran nivel de expresividad. Al permitir la creación de procedimientos en tiempo de ejecución, permiten un gran nivel de modularidad que, de acuerdo con sus proponentes, difícilmente puede alcanzarse en otros paradigmas de programación.
- El lenguaje provee algunas funciones básicas, que son primitivas para construir funciones más complejas. Se definen algunas estructuras para representar datos en los parámetros y resultados de las funciones. Este paradigma normalmente se implementa mediante interpretadores, pero también se pueden compilar.

Lisp

Lisp no es sólo uno de los lenguajes más viejos que existen, sino también el primero en proporcionar recursividad, funciones como ciudadanos de primera clase, recolección de basura y una definición formal del lenguaje (escrita asimismo en Lisp). Las diversas implementaciones de Lisp desarrolladas a través de los años han sido también pioneras en cuanto al uso de ambientes integrados de programación, los cuales combinan editores, intérpretes y depuradores.

Las primeras implementaciones de Lisp tuvieron, sin embargo, algunos problemas que las hicieron perder popularidad, como por ejemplo su tremenda lentitud para efectuar cálculos numéricos, su sintaxis basada por completo en paréntesis que suele causar gran confusión entre los usuarios novatos y su carencia de tipos que hace difícil la detección de errores y el desarrollo de compiladores.

Además, problemas adicionales tales como la carencia de verdaderas funciones como ciudadanos de primera clase y el uso de reglas de ámbito dinámicas, bajo las cuales el valor de una variable libre se toma del ambiente de activación, hicieron que Lisp se mantuviera durante un buen tiempo como un lenguaje restringido a los laboratorios de investigación, lejos del alcance de un número significativo de usuarios. El mismo McCarthy afirmó que su lenguaje no era apropiado "para los programadores novatos o los no programadores", pues se requería una cierta cantidad de "conocimientos sofisticados para apreciar y usar el lenguaje efectivamente".

Scheme

Scheme es un lenguaje funcional, derivado de LISP. Scheme es un lenguaje compacto con un alto grado de abstracción, por lo cual resulta adecuado para cursos introductorios de computación, donde el énfasis está en la metodología de resolución de problemas. Scheme permite resolver problemas complejos con programas cortos y elegantes. De este modo, el lenguaje se convierte en un aliado para resolver problemas, no un problema más que resolver.

Hoy en día Scheme es un lenguaje simple, pero poderoso; pequeño, pero flexible. Su naturaleza lo hace ideal para la enseñanza, incluso como primer lenguaje de programación, por su notable facilidad para incorporar diversos paradigmas con sólo un puñado de primitivas.

Como lenguaje imperativo, conserva la pureza y elegancia que le proporcionan la recursividad y las funciones de orden superior, superando así a muchos de los lenguajes que hoy gozan de gran popularidad, tales como Pascal y C. Como lenguaje orientado a objetos, hace alarde de un sistema de paso de mensajes y de manipulación de objetos equiparable únicamente al de Smalltalk. Su mecanismo de continuaciones proporciona procedimientos de escape como ciudadanos de primer orden, condenando a la obsolescencia a los mecanismos similares con que cuentan otros lenguajes.

El uso de streams permite implementar la evaluación concisa, convirtiendo al lenguaje en una herramienta idónea para cursos avanzados de lenguajes de programación. Una de sus pocas desventajas estriba en su carencia de tipos, aunque existen versiones de Scheme que incorporan un sistema similar al de ML (por ejemplo, Scheme 48).

Desde una perspectiva más pragmática, podemos decir que su sintaxis es extremadamente simple, lo que permite que el lenguaje pueda dominarse fácilmente en sólo 6 meses. Los estudiantes suelen preferirlo, sobre todo, cuando se trata del primer lenguaje de programación que aprenden, y los instructores lo elogian porque facilita la enseñanza de ideas de abstracción y diseño de algoritmos, tan útiles para formar buenos programadores.

Con la sabia filosofía de proporcionar sólo una primitiva que haga lo que queremos en vez de varias, como en otros lenguajes, Scheme se erige en la cima de los lenguajes preferidos por las nuevas generaciones, ya no sólo como una mera curiosidad científica, como se vió a Lisp en otra época, sino como una herramienta efectiva que sirve para aprender a programar y a resolver problemas del mundo real.

1.3.4. Ventajas de la Programación Funcional

Las ventajas de tener un lenguaje tan simple son:

- Permite definiciones simples.
- Facilita el estudio de aspectos computacionales.
- Su carácter formal facilita la demostración de propiedades.

Aplicaciones

- Compilación de lenguajes funcionales.
- Especificar semántica a lenguajes imperativos.
- Formalismo para definir otras teorías.

1.3.5. Lenguaje de Programación Funcional F#

Es un lenguaje de programación, que proporciona la mejor combinación de seguridad, rendimiento y script, con todas las ventajas de ejecutarse en un runtime moderno. Se ejecuta sobre el Framework de .NET

Además es un lenguaje de programación de script, funcional, imperativo, orientado a objetos que es una base fantástica para realizar diversidad de tareas dentro de la programación, prácticas científicas, tareas en web, etc.

- Script interactivo como Phyton.
- Ambiente interactivo de visualización de datos como MATLAB.
- Fuerte inferencia de tipos y seguridad de ML.
- Compilación compatible compartida con el lenguaje popular OCaml.
- Un performance como C#.
- Fácil acceso a todas las librerías de clases base que tenemos en .NET así como herramientas de acceso a datos.
- Manejo de esquemas.
- Una integración con Visual Studio.
- La velocidad de ejecución de código nativo, ya sea portable, o distribuido.

F# es una variante de ML que comparte un lenguaje administrador con OCaml. Los programas hechos en F# corren sobre el Framework de .NET.

Es un lenguaje 100% matemático que utiliza el compilador de .Net para crear los runtimes de ejecución del mismo. Además esta versión es compatible con Mono.

Es el primer lenguaje ML donde todos los tipos y valores en un programa de ML pueden ser accedidos de una manera predecible y amistosa desde otros lenguajes (ej. C #).

F # fue el primer lenguaje de .NET liberado, en producir IL Genérico, y el compilador fue diseñado en parte con este lenguaje en mente.

Soporta características que a menudo están ausentes en implementaciones ML tales como Unicode strings, encadenamiento dinámico, multihebras con derecho preferente y maquina de soporte SMP.

El ambiente interactivo fsi.exe soporta el desarrollo de alto nivel y exploración de la dinámica de su código y ambiente.

El compilador de línea de comandos fsc.exe soporta la compilación separada, depuración y optimización de la información.

F # viene con F # para Visual Studio, una extensión de Visual Studio 2003 y Visual Studio 2005 que es compatible con características como un entorno integrado compilación/depuración, depuración gráfica, resaltado de sintaxis interactivo, análisis y comprobación de tipos (tipos de datos), IntelliSense (), CodeSense, MethodTips (Sugerencia de Métodos) y un sistema simple del proyecto.

Puede usarse con herramientas del Framework de .NET, Visual Studio de Microsoft y muchas otras herramientas de desarrollo de .NET.

Viene con una librería compatible de ML que aproxima y extiende algunas librerías de OCaml 3.06. Esto significa que usted no tiene que usar librerías de .NET si no es necesario. Es posible escribir aplicaciones grandes y sofisticadas que pueden ser compiladas como código OCaml o código F #.

CAPÍTULO II

MODELOS MATEMÁTICOS PARA LA PLANIFICACIÓN DE LA PRODUCCIÓN

2.1. Modelos Matemáticos

2.1.1. Introducción

Un modelo es una representación ideal de un sistema y la forma en que este opera. El objetivo es analizar el comportamiento del sistema o bien predecir su comportamiento futuro. Obviamente los modelos no son tan complejos como el sistema mismo, de tal manera que se hacen las suposiciones y restricciones necesarias para representar las porciones más relevantes del mismo.

Un modelo matemático se define como una descripción desde el punto de vista de las matemáticas de un hecho o fenómeno del mundo real, desde el tamaño de la población, hasta fenómenos físicos como la velocidad, aceleración o densidad. El objetivo del modelo

matemático es entender ampliamente el fenómeno y tal vez predecir su comportamiento en el futuro.

Claramente no habría ventaja alguna de utilizar modelos si estos no simplificaran la situación real. En muchos casos podemos utilizar modelos matemáticos que, mediante letras, números y operaciones, representan variables, magnitudes y sus relaciones.

El proceso para elaborar un modelo matemático es el siguiente:

- Encontrar un problema del mundo real
- Formular un modelo matemático acerca del problema, identificando variables (dependientes e independientes) y estableciendo hipótesis lo suficientemente simples para tratarse de manera matemática.
- Aplicar los conocimientos matemáticos que se posee para llegar a conclusiones matemáticas.
- Comparar los resultados obtenidos como predicciones con datos reales. Si los datos son diferentes, se reinicia el proceso.

Es importante mencionar que un modelo matemático no es completamente exacto con problemas de la vida real, de hecho, se trata de una idealización. Hay una gran cantidad de funciones que representan relaciones observadas en el mundo real.

El tipo de problemas, y la necesidad de encontrar la mejor forma de resolverlos, proporcionaron el surgimiento de la Investigación de Operaciones, que aspira determinar la mejor solución (optima) para un problema de decisión con la restricción de recursos limitados.

En la Investigación de Operaciones utilizaremos herramientas que nos permiten tomar una decisión a la hora de resolver un problema tal es el caso de los modelos, que se emplean según sea la necesidad.

2.1.2. Características de la Investigación de operaciones

- Usa el método científico para investigar el problema en cuestión. En particular, el proceso comienza por la observación cuidadosa y la formulación del problema incluyendo la recolección de datos pertinentes.

- Adopta un punto de vista organizacional. De esta manera intenta resolver los conflictos de interés entre los componentes de la organización de forma que el resultado sea el mejor para la organización completa.
- Intenta encontrar una mejor solución (llamada solución óptima), para el problema bajo consideración. En lugar de contentarse con mejorar el estado de las cosas, la meta es identificar el mejor curso de acción posible.
- Es necesario emplear el enfoque de equipo. Este equipo debe incluir personal con antecedentes firmes en matemáticas, estadísticas y teoría de probabilidades, economía, administración de empresas ciencias de la computación, ingeniería, etc. El equipo también necesita tener la experiencia y las habilidades para permitir la consideración adecuada de todas las ramificaciones del problema.
- La Investigación de Operaciones ha desarrollado una serie de técnicas y modelos muy útiles a la Ingeniería de Sistemas. Entre ellos tenemos: la Programación No Lineal, Teoría de Colas, Programación Entera, Programación Dinámica, entre otras.
- La Investigación de Operaciones tiende a representar el problema cuantitativamente para poder analizar y evaluar un criterio común.

2.1.3. Aplicaciones de la Investigación de operaciones

La investigación de operaciones es usada para diversas áreas entre ellas destacamos:

- Presupuestos de capital
- Localización del activo
- Selección de cartera
- Prevención de fraude, lavado de dinero
- Benchmarking.
- Optimización del canal de marketing
- Campañas de venta Directa
- Programación de la cadena logística
- Distribución
- Asignación de recursos

- Planteamientos del inventario
- Planeamiento al por menor, optimización de Merchandizing
- Mezcla y empaque de productos, reducción de basura industrial

2.1.4. Definición de Modelos

Un modelo es una abstracción o una representación idealizada de la vida real. El propósito del modelo es proporcionar un medio para analizar el comportamiento del sistema con el fin de mejorar su desempeño. O si el sistema no existe todavía, para definir la estructura ideal de este sistema futuro indicando las relaciones funcionales entre sus elementos. La realidad de la solución obtenida depende de la validez de él para representar el sistema real. Entre más grande sea la discrepancia entre la salida del modelo y el mundo real, más impreciso es el modelo para describir el comportamiento del sistema original.

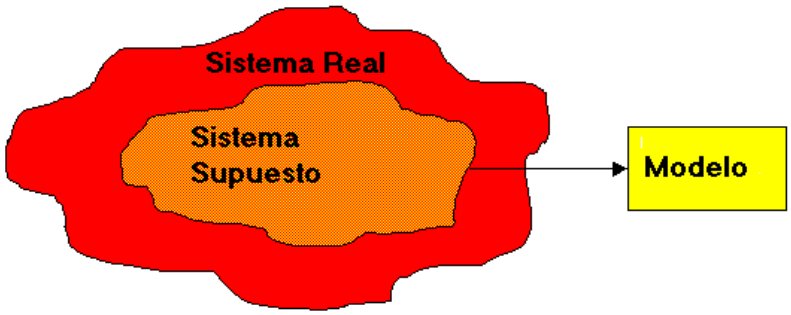

Figura II.10 Definición de modelo

2.1.5. Modelos Matemáticos

Un modelo matemático se define como una descripción desde el punto de vista de las matemáticas de un hecho o fenómeno del mundo real, desde el tamaño de la población, hasta fenómenos físicos como la velocidad, aceleración o densidad. El objetivo del modelo matemático es entender ampliamente el fenómeno y tal vez predecir su comportamiento en el futuro.

Características que deben presentar los modelos:

- Deben ser fáciles de entender y manejar.
- Deben ser simples y de costo no excesivo.

- Deben ser una buena aproximación del sistema real, que controle el mayor número posible de aspectos del mismo y que éstos contribuyan de forma significativa al sistema (hay relaciones en el sistema que no son significativas y pueden obviarse en el modelo).

Un modelo matemático comprende principalmente tres conjuntos básicos de elementos.

Estos son:

- Variables y parámetros de decisión. Las variables de decisión son las incógnitas (o decisiones) que deben determinarse resolviendo el modelo. Los parámetros son los valores conocidos que relacionan las variables de decisión con las restricciones y las funciones objetivo. Los parámetros del modelo pueden ser determinísticos o probabilísticos (estocásticos).
- Restricciones. Para tener en cuenta las limitaciones tecnológicas, económicas y otras del sistema, el modelo debe incluir restricciones (implícitas o explícitas) que restrinjan las variables de decisión a un rango de valores factibles.
- Función objetivo. La función objetivo define la medida de efectividad del sistema como función matemática de las variables de decisión. Una decisión óptima del modelo se obtiene cuando los valores de las variables de decisión producen el mejor valor de la función objetivo, sujeta a las restricciones.

Una formulación pobre o inapropiada de la función objetivo conduce a una solución pobre del problema. Un ejemplo común de esto ocurre cuando se desprecian algunos aspectos del sistema. Por ejemplo, para determinar el nivel óptimo de producción de un determinado producto, la función objetivo puede reflejar solamente metas de producción del departamento, despreciando las metas de mercado y finanzas.

2.1.6. Clasificación de los modelos Matemáticos

Un modelo Matemático en términos sencillos es un grupo de ecuaciones o inecuaciones que representan una realidad. El ingrediente principal en un modelo matemático, como es de esperarse, es la variable. Las variables, son la representación de los diferentes posibilidades

de un conjunto de datos y estos datos en su origen pueden ser de tipo Determinísticos o estocásticos.

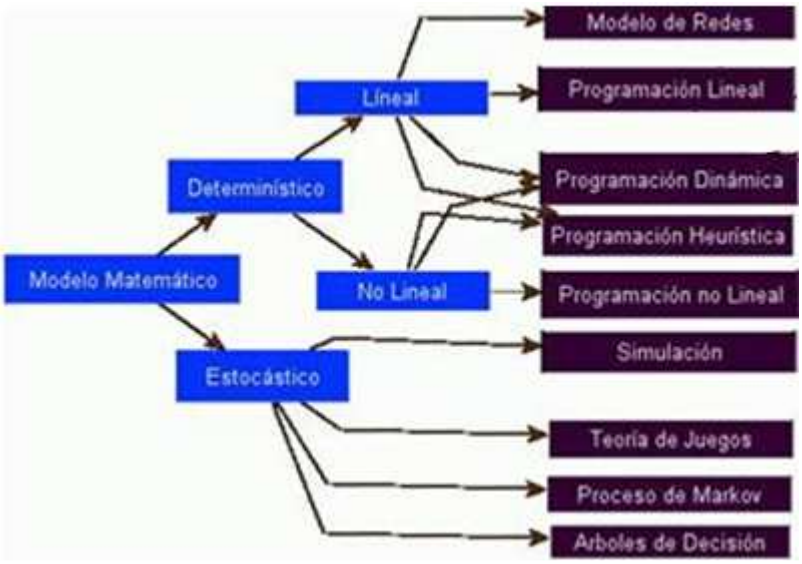

Figura II.11 Clasificación de los modelos matemáticos

Determinísticos

Los modelos determinísticos son los que hacen predicciones definidas de cantidades, dentro de cualquier distribución de probabilidades, también se les puede definir como aquellos que se aplican a problemas en los que hay un solo estado de la naturaleza, y dónde variables, limitaciones y alternativas son, después de que se aceptan los supuestos, conocidos, definibles, finitos y predecibles con confidencia estadística. Algunos modelos, herramientas o técnicas determinísticos son: programación lineal, análisis de Markov, costo/beneficio, etc. En otras palabras, un modelo determinístico se construye para una condición de certeza supuesta, y el modelo asume que solo hay un resultado posible (el cual es conocido) para cada acción o curso alternativo.

Los modelos determinísticos se clasifican a su vez en:

- Lineales
- No Lineales

Lineales.- Llamamos modelos lineales a aquellas situaciones que después de haber sido analizadas matemáticamente, se representan por medio de una función lineal. En algunos casos nuestro modelo coincide precisamente con una recta, en otros casos, a pesar de que las variables que nos interesan no pertenecen todas a la misma línea, es posible encontrar una función lineal que mejor se aproxime a nuestro problema, ayudándonos a obtener información valiosa. Nuestro modelo lineal se puede determinar de manera gráfica o bien, por medio de una ecuación. Existen ocasiones en que a una de nuestras variables le pedimos que cumpla varias condiciones a la vez, entonces surge un conjunto de ecuaciones donde el punto de intersección de dichas ecuaciones representa la solución de nuestro problema.

Dentro de la categoría de modelos lineales tenemos los siguientes:

- Modelo de Redes.
- Programación lineal.

- **Modelo de Redes.-** la familia de redes de los problemas de optimización incluye los siguientes prototipos de modelos: Problemas de asignación, camino crítico, flujo máximo, camino más corto, transporte y costo mínimo de flujos. Los problemas son establecidos fácilmente mediante el uso de arcos de redes y de los nodos.

Aplicabilidad de los modelos de redes

Los modelos de redes son aplicables a una extensa variedad de problemas de decisión, los cuales pueden ser modelados como:

- Problemas de optimización de redes que pueden ser eficiente y efectivamente resueltos.
- Algunos de estos problemas de decisión son realmente problemas físicos, tales como el transporte o flujo de bienes materiales.
- Sin embargo, muchos problemas de redes son más que una representación abstracta de procesos o actividades, tales como el camino crítico en las actividades entre las redes de un proyecto gerencial.
- Juegan un papel importante en la gerencia logística y en la cadena de insumos para reducir costos y mejorar servicios. Por lo tanto, el objetivo es encontrar la manera más efectiva en término de costos para transportar bienes.

- **Programación lineal.-** es un procedimiento o algoritmo matemático mediante el cual se resuelve un problema indeterminado, formulado a través de ecuaciones lineales, optimizando la función objetivo, también lineal.

Consiste en optimizar (minimizar o maximizar) una función lineal, que denominaremos función objetivo, de tal forma que las variables de dicha función estén sujetas a una serie de restricciones que expresamos mediante un sistema de inecuaciones lineales.

La Programación Lineal (PL) es un procedimiento matemático para determinar la asignación óptima de recursos escasos. La PL es un procedimiento que encuentra su aplicación práctica en casi todas las facetas de los negocios, desde la publicidad hasta la planificación de la producción. Problemas de transporte, distribución, y planificación global de la producción son los objetos más comunes del análisis de PL.

Aplicabilidad de la Programación Lineal

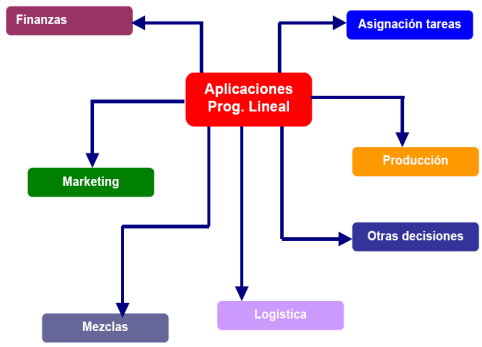

Figura II.12 Aplicación de la programación lineal

La Programación Lineal permite resolver problemas de:

- Mezclas
- Nutrición de animales
- Distribución de factorías
- Afectación de personal a distintos puestos de trabajo
- Almacenaje
- Planes de producción
- Escalonamiento de la fabricación

- Problemas de circulación
- Planes de optimización de semáforos
- Estudios de comunicaciones internas

No lineales.- Los modelos no lineales se caracteriza por ser muy diversos, de tal manera que incluso el caos forma una parte pequeña de los modelos no lineales; esto es, cuando el modelo no lineal va a predecir que el resultado no es predecible. Sin embargo, su interés se centra en predecir los resultados de la manera más exacta posible.

Dentro de la categoría de modelos no lineales tenemos los siguientes:

- Programación Dinámica
- Programación Heurística
- Programación no lineal

- **Programación Dinámica.-** La Programación Dinámica es un método de solución de problemas que permite descomponer un modelo matemático muy grande, en problemas más pequeños de fácil resolución, que al resolver cada uno de ellos se obtiene la solución apropiada al problema mayor del cual fueron generados los más pequeños. La programación dinámica se apoya como estrategia de solución en "divide y vencerás". En sí, la programación dinámica, no tiene una técnica específica que se aplique a todos los problemas para resolverlos como es el caso de la programación lineal. Más bien, es un método de resolución de problemas, que aunque mantiene algunas características comunes entre unos y otros, cada aplicación requiere un grado de práctica, creatividad y conocimientos para lograr dividir el problema grande en muchos pequeños lógicos y que se encuentren relacionados entre sí.

Aplicabilidad de la Programación Dinámica

La Programación dinámica se puede aplicara para resolver problemas como:

- Problema de la mochila (Snapsack Problem)
- Problema de la diligencia (Stagecoach Problem)
- Programación de producción e inventarios (Production and Inventory Scheduling)

- **Programación Heurística.-** Está basado en el modelo de comportamiento humano y su estilo para resolver problemas complejos, además, implica una forma de modelizar el problema en lo que respecta a la representación de su estructura, estrategias de búsqueda y métodos de resolución. Existen diversos tipos de programas que incluyen algoritmos heurísticos. Varios de ellos son capaces de aprender de su experiencia.

Aplicabilidad de la Programación Heurística

La programación heurística según varios autores es que el método heurístico forma parte vital, pero despreciada; del conjunto de herramientas del analista de investigación operacional. No se sugiere, sin embargo que todo problema sea resuelto por programación heurística. De hecho, la programación heurística se debe considerar únicamente si es obvio que los otros métodos fallarán. El punto es que este tipo de programación debe ser utilizada solamente cuando las técnicas exactas no están disponibles y/o no son económicas.

Los dos usos principales del enfoque de programación heurística son:

- Para resolver problemas de tal magnitud, que métodos más exactos y elegantes no pueden ser empleados.
- Para obtener un valor de iniciación aceptable, si no óptimo para los procedimientos más elegantes.

  Ejemplos:

  - Utilizando solo dos cubos sin marcas, uno de 6 litros y otro para 8 litros de capacidad; llenar un cubo con cuatro litros de agua.
  - Determinar la mejor asignación de 50 puestos de trabajo, con cada uno de los 30 operarios de una empresa.

- **Programación no Lineal.-** es el proceso de resolución de un sistema de igualdades y desigualdades sujetas a un conjunto de restricciones sobre un conjunto de variables reales desconocidas, con una función objetivo a maximizar, cuando alguna de las restricciones o la función objetivo no son lineales. Cuando el conjunto de restricciones, la función objetivo, o ambos, son no lineales, se dice que se trata de un problema de programación no lineal (PPNL).

Aplicabilidad de la Programación no Lineal

Podemos resolver con programación no lineal problemas:

- Bidimensionales
- Tridimensionales
- Optimización
- Estimación de estado en sistemas eléctricos
- Reparto optimo de carga
- Ejemplos geométricos
- Ejemplos mecánicos

Aleatorios o Estocásticos. Los modelos estocásticos contienen elementos aleatorios distribuidos dentro del modelo; de tal manera que predicen el valor previsto o una cantidad en términos de probabilidad de ocurrencia; también se les puede definir como aquellos modelos cuantitativos en los que hay más de un estado de la naturaleza y donde cada estado debe estimarse o definirse para permitir el cálculo de los resultados condicionales de cada alternativa de decisión en cada estado; cuando riesgo e incertidumbre están implicados en el problema de decisión, se emplean los modelos probabilísticas cuantitativos.

Dentro de la categoría de aleatorios o estocásticos tenemos los siguientes:

- Simulación
- Teoría de Juegos
- Procesos de Markov
- Arboles de decisión

- **Simulación.-** Simulación es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso y conducir experimentos con este modelo con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales se puede operar el sistema. Es imitar una situación del mundo real en forma matemática. La simulación constituye una técnica económica que nos permite ofrecer varios escenarios posibles de una

situación y nos permite equivocarnos sin provocar efectos sobre el mundo real (por ejemplo un simulador de vuelo o conducción).

Aplicabilidad de la Simulación

La simulación surge en tipos diferentes de problemas:

- Problemas que involucrarán una clase de proceso estocástico: Demanda por un articulo, tiempo de espera antes de empezar una producción, etc.
- Ciertos problemas matemáticos completamente determinísticos que no pueden resolverse fácilmente por métodos analíticos o determinísticos; sin embargo, es posible obtener soluciones aproximadas a estos simulando un proceso estocástico cuyos momentos, función de densidad o de distribución satisfacen las relaciones funcionales o los requisitos del problema determinístico: ejemplo: Ecuaciones diferenciales complejas.
- Cuando se está estudiando un sistema por medio de investigación de operaciones, es necesario usar la simulación en aquellas etapas que estén ocasionando dificultades.
- La simulación además puede realizarse para verificar soluciones analíticas.
- También permite estudiar los sistemas dinámicos, ya sea en tiempo real, comprimido o expandido.

- **Teoría de Juegos.-** La teoría de juegos es un área de la matemática aplicada que utiliza modelos para estudiar interacciones en estructuras formalizadas de incentivos (los llamados juegos) y llevar a cabo procesos de decisión. Sus investigadores estudian las estrategias óptimas así como el comportamiento previsto y observado de individuos en juegos. Tipos de interacción aparentemente distintos pueden, en realidad, presentar estructuras de incentivos similares y, por lo tanto, se puede representar mil veces conjuntamente un mismo juego.

Aplicabilidad de la teoría de juegos

Desarrollada en sus comienzos como una herramienta para entender el comportamiento de la economía, la teoría de juegos se usa actualmente en muchos campos, desde la biología a la filosofía.

Desde los setenta, la teoría de juegos se ha aplicado a la conducta animal, incluyendo el desarrollo de las especies por la selección natural. A raíz de juegos como el dilema del prisionero, en los que el egoísmo generalizado perjudica a los jugadores, la teoría de juegos se ha usado en economía, ciencias políticas, ética y filosofía. Finalmente, ha atraído también la atención de los investigadores en informática, usándose en inteligencia artificial y cibernética.
Los analistas de juegos utilizan asiduamente otras áreas de la matemática, en particular las probabilidades, las estadísticas y la programación lineal, en conjunto con la teoría de juegos.

- **Procesos de Márkov.-** Una cadena de Márkov, recibe su nombre del matemático ruso Andrei Andreevitch Márkov (1856-1922), es una serie de eventos, en la cual la probabilidad de que ocurra un evento depende del evento inmediato anterior. En efecto, las cadenas de este tipo tienen memoria. "Recuerdan" el último evento y esto condiciona las posibilidades de los eventos futuros. Esta dependencia del evento anterior distingue a las cadenas de Márkov de las series de eventos independientes, como tirar una moneda al aire o un dado. En los negocios, las cadenas de Márkov se han utilizado para analizar los patrones de compra de los deudores morosos, para planear las necesidades de personal y para analizar el reemplazo de equipo.

Aplicabilidad de los Procesos de Márkov

- Los procesos de Markov aparecen ampliamente en la física, en particular la mecánica estadística, siempre que las probabilidades se utilizan para representar detalles desconocidos del sistema.
- Teoría de colas y en estadística.
- Estado efectivo de estimación y reconocimiento de patrones.
- Reconocimiento de voz
- Bioinformática, por ejemplo para la codificación de región o de predicción de genes.
- Comportamiento de navegación web de los usuarios.
- Generación de secuencias de números aleatorios.
- Modelos biológicos.

- Geoestadística.
- Juegos de azar.

- **Árboles de decisión.-** El árbol de decisión es un diagrama que representan en forma secuencial condiciones y acciones, muestra qué condiciones se consideran en primer lugar, en segundo lugar y así sucesivamente. Este método permite mostrar la relación que existe entre cada condición y el grupo de acciones permisibles asociado con ella, además, modela funciones discretas, en las que el objetivo es determinar el valor combinado de un conjunto de variables, y basándose en el valor de cada una de ellas, determinar la acción a ser tomada. Los árboles de decisión son normalmente construidos a partir de la descripción de la narrativa de un problema. Ellos proveen una visión gráfica de la toma de decisión necesaria, especifican las variables que son evaluadas, qué acciones deben ser tomadas y el orden en la cual la toma de decisión será efectuada. Cada vez que se ejecuta un árbol de decisión, solo un camino será seguido dependiendo del valor actual de la variable evaluada.

Aplicabilidad de los Árboles de decisión

- Búsqueda binaria.
- Sistemas Expertos
- Árboles de juegos

2.1.7. Selección de Modelos Matemáticos

Una vez analizada la clasificación propuesta donde se detallan las características de cada uno de los modelos y su aplicación en problemas reales, seleccionamos los tres modelos siguientes que dentro de su aplicabilidad resuelven problemas de planificación de producción.

- Programación lineal
- Programación Dinámica
- Programación No lineal

2.1.8. Conclusión de la selección de los modelos

De los modelos listados anteriormente, que bibliográficamente hemos observados resuelven problemas de planificación de producción, hemos tomado como ejemplo para nuestro trabajo de investigación, la programación lineal, con el método simplex para su resolución. Este modelo se implementará en los tres paradigmas de programación para su posterior comparación.

A continuación mostramos una de las ventajas por las que tomamos este modelo para nuestro estudio.

- La programación lineal trata de la planeación de las actividades para obtener un resultado optimo, esto es, el resultado que mejor alcance la meta especificada (según el modelo matemático) entre todas alternativas de solución.
- El modelo lineal ofrece una planeación lineal, la misma que ofrece un conjunto de técnicas matemáticas que intentan obtener el mayor provecho posible de sistemas económicos, sociales y tecnológicos; cuyo funcionamiento se puede describir matemáticamente de modo adecuado.
- Los modelos lineales ofrecen una gran variedad de métodos de resolución, siendo uno de los más eficientes y utilizados el método simplex, para resolver problemas incluso los de gran tamaño.
- Muchas empresas a través de su aplicación han logrado grandes ahorros de recursos.

El modelo matemático para la planta de lácteos de la ESPOCH, será construido en base al modelo matemático seleccionado, el mismo que después será implementado en el desarrollo de la aplicación para dicha planta antes mencionada.

2.2. Construcción del modelo matemático para la planta de lácteos de la ESPOCH.

Antecedentes

La Planta de Lácteos ESPOCH, perteneciente a la Facultad de Ciencias Pecuarias, es una Unidad de Producción de la Escuela Superior Politécnica de Chimborazo; se encuentra ubicada en Tunshi, iniciando su funcionamiento en el año 2000. Cuenta con equipos de

pasteurización adquiridos por una donación mediante el Convenio de la ESPOCH con la Embajada del Japón.

La planta además sirve de apoyo académico a los estudiantes de la facultad, ya que los mismos pueden realizar prácticas de elaboración de lácteos y de esta manera adquirir destrezas y fortalecer sus conocimientos.

Entre los objetivos que tiene la Planta es el generar recursos económicos autofinanciados para la ESPOCH, con la producción y comercialización de sus productos lácteos, entre los productos están: **Leche Pasteurizada en Funda, Queso y Yogurt**, siendo el yogurt producido con menos frecuencia debido a su menor demanda.

A pesar de que la planta de lácteos brinda facilidades para los estudiantes y genera recursos económicos por ser una unidad de producción, es necesario que la misma brinde la utilidad máxima, determinando los requerimientos de insumos necesarios para la elaboración de los productos, controlando la utilización de los mismos, calculando las cantidades de producto terminado a fabricar, así como los componentes necesarios y las materias primas a comprar para poder satisfacer la demanda del mercado; lograremos lo propuesto con la creación de un sistema que Planifique y Controle la producción y el inventario de insumos de la planta, el mismo que será desarrollado en base al modelo matemático diseñado para la misma, que represente las condiciones reales que interviene en la elaboración de los productos y contemple cada uno de los elementos y restricciones necesarias para obtener la meta deseada.

Determinación de las variables y restricciones existentes en la planta

Variables

Las variables que hemos identificado son las siguientes:

d = demanda

X1 = cantidad de leche

X2 = cantidad de queso

X3 = cantidad de yogurt

mp1 = cantidad de materia prima para elaborar 1 litro de leche en funda

mp2 = cantidad de materia prima para elaborar 1 queso

mp3 = cantidad de materia prima para elaborar 1 litro de yogurt

mpm = materia prima receptada mensualmente

cpl = capacidad de producción actual por hora de la Leche en funda por hora

cpq = capacidad de producción actual por hora del queso

cpy = capacidad de producción actual por hora del yogurt

cpml = capacidad de producción mensual de leche

cpmq = capacidad de producción mensual de queso

cpym = capacidad de producción mensual de yogurt

tp1 = tiempo de producción de 1 litro de leche en funda

tp2 = tiempo de producción de 1 queso

tp3 = tiempo de producción de 1 litro de yogurt

tpm = tiempo de producción mensual

idpa => inventario disponible al final del periodo anterior

stockb => stock en bodega

pvp => Precio de venta

Leche

Producción mensual

$$\boldsymbol{PM}$$

Sueldo mensual

$$\boldsymbol{SM}$$

Porcentaje de producción

$$\boldsymbol{PProd}$$

Sueldo mensual

$$\boldsymbol{SMes}$$

Precio litro leche

$$\boldsymbol{PLL}$$

Costo Mensual energía eléctrica

$$\boldsymbol{CMEE}$$

Costo Mano de obra directa

$$\boldsymbol{CMOD} = SMes * PProd * X1$$

Costo Mano de obra indirecta

$$\boldsymbol{CMOI} = SMes * PProd * X1$$

Costo de Materia prima directa

$$\boldsymbol{CMPD} = PM / PLL$$

Costo Materia prima indirecta

$$\boldsymbol{CMPI} = \sum_{i=1}^{n} (\text{Precioinsumo}\mathbf{i} * \mathbf{PM}\ ) /\text{PM}$$

donde i= 1, 2 ,3,…n n>0

Costo Servicios básicos

$$\boldsymbol{CSB} = (CMEE\ * PProd)_{/PM}$$

Costo unitario de producción leche

$$\boldsymbol{CUPleche} = \sum CMOD + CMOI + CMPD + CMPI + CSB$$

Costo de producción total leche

$$\boldsymbol{CPL} = CULeche\ * PM$$

Total ventas

$$\boldsymbol{TV}$$

Utilidad bruta

$$\boldsymbol{U1} = TV - CP$$

Depreciación anual leche

$$\boldsymbol{DAL}$$

Depreciación mensual leche

$$\boldsymbol{DML} = DAL/12$$

Gasto mensual transporte

$$\boldsymbol{GMT}$$

Gasto mensual combustible y lubricantes

$$\boldsymbol{GMCYL}$$

Gastos de operación

$$\boldsymbol{GO} = DML + GMT + GMCYL$$

Utilidad neta

$$\boldsymbol{UN} = UB - GO$$

Utilidad leche

$$\boldsymbol{UL} = UN/PM$$

Queso

Costo Mano de obra directa

$$\boldsymbol{CMOD} = SMes * PProd * X2$$

Costo Mano de obra indirecta

$$\boldsymbol{CMOI} = SMes * PProd * X2$$

Costo de Materia prima directa

$$\boldsymbol{CMPD} = PM / PLL$$

Costo Materia prima indirecta

$$\boldsymbol{CMPI} = \sum_{i=1}^{n} (\text{Precioinsumo}\mathbf{i} * \text{PM}) / \text{PM}$$

donde i= 1, 2 ,3,…n n>0

Costo Servicios básicos

$$\boldsymbol{CSB} = (CMEE * PProd)/_{PM}$$

Costo unitario de producción queso

$$\boldsymbol{CUPqueso} = \sum CMOD + CMOI + CMPD + CMPI + CSB$$

Costo de producción total queso

$$\boldsymbol{CPQ} = CUqueso * PM$$

Total ventas

$$\boldsymbol{TV}$$

Utilidad bruta

$$\boldsymbol{U2} = TV - CP$$

Depreciación anual queso

$$\boldsymbol{DAQ}$$

Depreciación mensual queso

$$\boldsymbol{DMQ} = DAQ/12$$

Gasto mensual transporte

$$\boldsymbol{GMT}$$

Gasto mensual combustible y lubricantes

$$\boldsymbol{GMCYL}$$

Gastos de operación

$$\boldsymbol{GO} = DMQ + GMT + GMCYL$$

Utilidad neta

$$\boldsymbol{UN} = UB - GO$$

Utilidad queso

$$\boldsymbol{UQ} = UN/PM$$

Yogurt

Costo Mano de obra directa

$$\boldsymbol{CMOD} = SMes * PProd * X3$$

Costo Mano de obra indirecta

$$\boldsymbol{CMOI} = SMes * PProd * X3$$

Costo de Materia prima directa

$$\boldsymbol{CMPD} = PM\ /PLL$$

Costo Materia prima indirecta

$$\boldsymbol{CMPI} = \sum_{i=1}^{n} (\text{Precioinsumo}\mathbf{i} * \mathbf{PM}\ )\ /\text{PM}$$

donde i= 1, 2 ,3,…n n>0

Costo Servicios básicos

$$\boldsymbol{CSB} = (CMEE\ * PProd)/_{PM}$$

Costo por unidad de producción

$$\boldsymbol{CUyogurt} = \sum CMOD + CMOI + CMPD + CMPI + CSB$$

Costo de producción total yogurt

$$\boldsymbol{CPY} = CUyogurt * PM$$

Total ventas

$$\boldsymbol{TV}$$

Utilidad bruta

$$\boldsymbol{U3} = TV - CP$$

Depreciación anual yogurt

$$\boldsymbol{DAY}$$

Depreciación mensual yogurt

$$\boldsymbol{DMY} = DAY/12$$

Gasto mensual transporte

$$\boldsymbol{GMT}$$

Gasto mensual combustible y lubricantes

$$\boldsymbol{GMCYL}$$

Gastos de operación

$$\boldsymbol{GO} = DMY + GMT + GMCYL$$

Utilidad neta

$$\boldsymbol{UN} = UB - GO$$

Utilidad Yogurt

$$\boldsymbol{UY} = UN/PM$$

Consideramos como costos los insumos, depreciación de maquinaria y todos los costos involucrados al momento de elaborar los productos.

Mientras que los gastos se refiere a los costos involucrados en la comercialización.

Determinación de la estructura del modelo matemático

Para lograr maximizar las utilidades de la planta de lácteos, es necesario determinar cuántas leches en funda, quesos y yogurt se debe fabricar mensualmente; pero primero estableceremos una función objetivo.

Los factores limitantes que normalmente provienen del exterior son: las limitaciones de materia prima (esta limitación proviene de la cantidad de leche que entregan los ganaderos), la capacidad de producción que tiene la planta (cuantas leches, quesos y yogurt puede producir como máximo) y el tiempo que se requiere para la elaboración de determinado producto (tiempo que se requiere para elaborar una leche, queso y yogurt).

En lo que refiere a materia prima, mensualmente se receptan como promedio 10255 litros de leche cruda, diariamente para producir una leche en funda se necesita 1 litro de leche cruda, 4.5 litros para un queso y 1 litro para una botella de un litro de yogurt.

En lo referente a capacidad de producción, la capacidad máxima de producción de la leche pasteurizada en funda, de acuerdo a la maquinaria que se posee es de 1000 litros por hora, es decir 8000 litros diarios; 800 quesos y 550 litros de yogurt diarios, que al mes resulta una capacidad de producción de 280500.

Otros de los factores limitantes que tiene la planta de lácteos radica en el tiempo que toma la realización de cada producto, dado que la planta tiene un tiempo de producción de 8 horas

diarias y tomando en cuenta la capacidad que tiene para producir un determinado producto tenemos que la planta cuneta con una capacidad de producción de 1000 litros de leche por hora, 120 unidades de queso por día y un total de 550 litros de yogurt cada 2 días. En consecuencia, la formulación utilizando Programación Lineal es la siguiente:

Función Objetivo (FO)

Dado que el propósito de la modelación de los procesos de Producción de la Planta de Lácteos de la ESPOCH es mejorar la utilidad, se ha planteado la siguiente función objetivo:

Maximizar:

FO: Zmax: u1X1 + u2X2 + u3X3

Sujeta a:

Restricciones

R1: Cantidad de leche cruda: mp1**X1** + mp2**X2** + mp3**X3** ≤ **mpm**

R2: Capacidad de producción de la leche: cpl1**X1** + 0 **X2** + 0 **X3** ≤ **cpml**

R2: Capacidad de producción del queso: 0 **X1** + cpq2**X2** + 0 **X3** ≤ **cpmq**

R2: Capacidad de producción del yogurt: 0 **X1** + 0 **X2** + cpy3**X3** ≤ **cpmy**

R3: Tiempo de producción: tp1**X1** + tp2**X2** + tp3**X3** ≤ **tpm**

Donde:

X1, X2, X3 ≥ 0 o no negativas

Considerando que a **mayor producción => mayor utilidad**

menor producción => menor utilidad

2.3. Validación del modelo

La validación consiste en comprobar que existe una correspondencia entre los datos reales y los datos o resultados que arroja el modelo propuesto. Además para comprobar la validez de un modelo se debe ver cómo el modelo puede predecir un comportamiento futuro ante unas determinadas entradas.

Para realizar la validación de nuestro modelo matemático, recolectamos toda la información necesaria para calcular las utilidades que actualmente genera la planta por la comercialización de una leche en funda, un queso y un litro de yogurt.

Recolección de datos

Para obtener las utilidades que se necesitan como entradas para el sistema vamos a realizar la recolección de los datos históricos de la planta de lácteos en el año 2008, de cada una de las cantidades detalladas diariamente, obtuvimos las cantidades mensuales, para obtener datos más exactos y que se ajusten mas a la realidad tomamos en cuenta para los cálculos posteriores las cantidades promedio mensuales. De entre las cuales anotamos: cantidad de litros de leche receptada diariamente, cantidad de litros de leche para producir leche en funda, cantidad de litros de leche para producir quesos, cantidad de litros de leche para producir yogurt, entre otros que son de importancia, los mismos que presentamos en la Tabla II.12

.

		Producción						Ventas Leche					Ventas Queso				Ventas Yogurt	
		Leche	leche Queso	Queso	Promedio Leche	Yogurt	Estudiantes	0.25 - 0.3	0.40 - 0.45	0,5	0,5	Total Ventas				Total Ventas		Ventas Total
Mes	Total Leche							Estudiantes	Tiendas	Hospital	Comedor		Tiendas	Hospital	Comedor		Hospital	
Enero	10761,00	5793,00	4043,00	882,00	4,58	20,00	905,00	$ 226,25	$ 764,80	$ 594,00	$ 1.420,20	$ 3.005,25	$ 715,50	$ 384,00	$ 192,00	$ 1.291,50	$ 30,00	$ 30,00
Febrero	7590,00	3367,00	3065,00	674,00	4,55	10,00	1148,00	$ 287,00	$ 676,00	$ 598,50	$ 265,00	$ 1.826,50	$ 616,50	$ 432,00	$ 48,00	$ 1.096,50	$ 15,00	$ 15,00
Marzo	6449,00	3965,00	2464,00	544,00	4,53	20,00	0,00	$ -	$ 787,20	$ 665,00	$ 538,50	$ 1.990,70	$ 483,00	$ 423,00	$ -	$ 906,00	$ 15,00	$ 15,00
Abril	8259,00	6255,00	1994,00	443,00	4,50	10,00	0,00	$ -	$ 168,75	$ 680,00	$ 2.038,50	$ 2.887,25	$ 183,60	$ 486,00	$ -	$ 669,60	$ 15,00	$ 15,00
Mayo	14174,00	7329,00	5329,00	1171,00	4,55	20,00	1490,00	$ 447,00	$ 792,00	$ 715,00	$ 2.014,00	$ 3.968,00	$ 1.270,80	$ 486,00	$ 54,00	$ 1.810,80	$ 30,00	$ 30,00
Junio	15832,00	7105,00	7097,00	1643,00	4,32	20,00	1610,00	$ 483,00	$ 711,00	$ 715,00	$ 2.038,50	$ 3.947,50	$ 644,40	$ 432,00	$ 97,20	$ 1.173,60	$ 30,00	$ 30,00
Julio	16734,00	5531,00	8093,00	1651,00	4,90	20,00	3090,00	$ 927,00	$ 805,50	$ 710,00	$ 1.133,00	$ 3.575,50	$ 1.110,60	$ 486,00	$ 45,00	$ 1.641,60	$ 30,00	$ 30,00
Agosto	14562,00	2621,00	9511,00	2016,00	4,72	20,00	2410,00	$ 723,00	$ 618,75	$ 725,00	$ -	$ 2.066,75	$ 846,00	$ 486,00	$ -	$ 1.332,00	$ 30,00	$ 30,00
Septiembre	17371,00	5351,00	6995,00	1423,00	4,92	50,00	5025,00	$ 1.507,50	$ 714,60	$ 790,00	$ -	$ 3.012,10	$ 1.252,40	$ 432,00	$ -	$ 1.684,40	$ 75,00	$ 75,00
Octubre	15316,00	7166,00	4362,00	989,00	4,41	20,00	3788,00	$ 1.136,40	$ 864,00	$ 735,00	$ -	$ 2.735,40	$ 1.659,20	$ 540,00	$ -	$ 2.199,20	$ 30,00	$ 30,00
Noviembre	13146,00	4965,00	3947,00	857,00	4,61	20,00	4254,00	$ 1.276,20	$ 891,00	$ 680,00	$ 1.022,50	$ 3.869,70	$ 1.131,20	$ 486,00	$ 36,00	$ 1.653,20	$ 30,00	$ 30,00
Diciembre	12060,00	2517,00	4073,00	885,00	4,60	10,00	5470,00	$ 1.641,00	$ 472,50	$ 565,00	$ 55,00	$ 2.733,50	$ 584,00	$ 486,00	$ -	$ 1.070,00	$ 30,00	$ 30,00
Total:	152254,00	61965,00	60973,00	13178,00	4,60	240,00	29190,00	$ 8.654,35	$ 8.266,10	$ 8.172,50	$ 10.525,20	$ 35.618,15	$ 10.497,20	$ 5.559,00	$ 472,20	$ 16.528,40	$ 360,00	$ 360,00
Promedio:	12687,83	5163,75	5081,08	1098,17		20,00	2432,50	$ 721,20	$ 688,84	$ 681,04	$ 877,10	$ 2.968,18	$ 874,77	$ 463,25	$ 39,35	$ 1.377,37	$ 30,00	$ 30,00

Tabla II.12 Datos históricos del 2008 de la planta de lácteos de la ESPOCH

El promedio de leche receptada mensualmente es **10.255**, sin contar con la leche que es designada para práctica de los estudiantes.

El promedio de producción de leche, queso y yogurt mensualmente es **5164**, **1098** y **20** respectivamente.

Además es necesario considerar la **capacidad de producción** y el **tiempo de producción** de los productos que se elaboran en la planta:

Tabla II.13 Capacidad de Producción

Producto	Leche	Queso	Yogurt
Capacidad de Producción máxima (1 hora)	750 litros	61 unidades	17.188 litros
Capacidad de Producción utilizada (1 hora)	24.83	5.28	0.10

Tabla II.14 Tiempo de producción

Producto	Leche	Queso	Yogurt
Tiempo de Producción	750 litros/hora	61 unidades/hora	34.375 litros/hora
Tiempo de Producción por unidad	0.0013 hora	0.0164 hora	0.0582 hora

Determinación del Costo de Producción

Para determinar el costo de producción, se debe tomar en cuenta cada uno de los costos que incurren al momento de la elaboración ya sea de forma directa o indirecta.

Los elementos que se considera para calcular el costo de producción son:

- Mano de Obra Directa
- Materia Prima Directa
- Mano de Obra Indirecta
- Materia Prima Indirecta
- Costos Indirectos de Fabricación

Mano de Obra Directa: es la mano de obra consumida en las áreas que tienen una relación directa con la producción o la prestación de algún servicio. Es la generada por los obreros y operarios calificados de la empresa.

Materia Prima: Son los materiales que serán sometidos a operaciones de transformación o manufactura para su cambio físico y/o químico, antes de que puedan venderse como productos terminados

Mano de Obra Indirecta: es la mano de obra consumida en las áreas administrativas de la empresa que sirven de apoyo a la producción y al comercio.

Materia Prima Indirecta: Son todos los materiales sujetos a transformación, que no se pueden identificar o cuantificar plenamente con los productos terminados

Costos Indirectos de Fabricación: Son aquellos que se requieren para poder producir y están relacionados con la función producción.

$$\boldsymbol{Costo\ de\ Producción} = Mano\ de\ Obra\ Directa + Materia\ Prima\ Directa + Mano\ de\ Obra\ Indirecta + Materia\ Prima\ Indirecta$$

$$\boldsymbol{Costo\ de\ Producción\ por\ unidad} = Costo\ de\ Producción / unidades\ producidas$$

$$\boldsymbol{Utilidad\ Bruta} = Ventas - Costo\ de\ Producción$$

$$\boldsymbol{Utilidad\ Neta} = Utilidad\ Bruta - Gastos\ de\ Operación$$

Ahora para calcular el Costo de producción tenemos:

El costo de materia prima directa según datos recolectados en la planta de lácteos, con el Ing. Marco Manzano, técnico de la misma es $ 0.30 centavos de dólar por litro de leche.

El costo mensual de mano de obra directa es $ 780 dólares y el costo por mano de obra indirecta es $630 y $550 dólares.

Tabla II.15 Pagos mensuales en la planta de lácteo ESPOCH

MANO DE OBRA DIRECTA	
Sr. Jaime Loja	$ 780
MANO DE OBRA INDIRECTA	
Ing. Marco Manzano	$ 630
Ing. Edison Villacrés	$ 550

Para detallar el costo mensual de los insumos (materia prima indirecta) para elaboración de la leche, queso y yogurt, elaboramos la Tabla II.16, Tabla II.17 y Tabla II.18, que además describe el costo de cada insumo por litro de leche.

Tabla II.16 Detalle de los insumos utilizados para la elaboración de la leche

MATERIA PRIMA INDIRECTA Leche	Uso de Insumos				Costos			Cantidad por 1 litro Leche		
	Cant.	unidad	Cant.	unidad	Precio	Cant.	Unidad	Cant.	Unidad	Precio
Saborizante	2	cm3	100	lt	$ 35,00	1000	cm3	0,0200	cm3	$ 0,0007
Fundas	1	Kg	325	fundas	$ 3,80	1	Kg	0,0031	Kg	$ 0,0117
Azúcar	2	Kg	500	lt	$ 0,45	1	Lb	0,0040	Kg	$ 0,0018
Conservante	35	cm3	100	lt	$ 2,60	1000	cm3	0,3500	cm3	$ 0,0009

Tabla II.17 Detalle de los insumos utilizados para la elaboración del queso

Materia PRIMA INDIRECTA Queso	Uso de Insumos				Costos			Cantidad por 1 litro Leche		
	Cant	Unidad	Cant	Unidad	Precio	Cant	Unidad	cant	Unidad	Precio
Cuajo	10	cm3	100	lt	$ 38,00	1000	cm3	0,1000	cm3	$ 0,0038
Sal	1	Kg	100	lt	$ 0,50	2	Kg	0,0100	Kg	$ 0,0025
Calcio	20	g	100	lt	$ 3,50	1000	G	0,2000	g	$ 0,0007
Fermento	1	Sobre			$ 4,50	50	Lt			$ 0,0900
Fundas	1	Funda	1	Queso	$ 0,01	1	Fundas	1,0000	Funda	$ 0,0100

Tabla II.18 Detalle de los insumos utilizados para la elaboración de yogurt

Materia PRIMA INDIRECTA Yogurt	Uso de Insumos				Costos			Cantidad por 1 litro Leche		
	Cant	Unidad	Cant	Unidad	Precio	Cant	Unidad	cant	Unidad	Precio
Fermento	1	Sobre	100	lt	$ 7,00	1	sobre	0,0100	Sobre	$ 0,0700
Saborizante	15	cm3	100	lt	$ 35,00	1000	cm3	0,1500	cm3	$ 0,0053
Colorante	10	g	100	lt	$ 20,00	453,5924	G	0,1000	g	$ 0,0044
Azúcar	15	Kg	100	lt	$ 42,00	50	kg	0,1500	Kg	$ 0,1260
Envases					$ 0,35	1	Embase 2lt			$ 0,18

Leche

En base a los valores recolectados vamos a calcular el costo de producción y costo de producción por unidad de **leche**.

Tabla II.19 Costo de producción de la leche considerando costo de mano de obra

	Gasto Mensual	Unidades Producidas	Valor Utilizado	
MANO DE OBRA DIRECTA				$ 0,08
Sr. Jaime Loja	$ 419,90	5163,75	$ 0,08	
MATERIA PRIMA DIRECTA				$ 0,30
Leche Cruda	$ 1.549,13	5163,75	$ 0,30	
COSTOS INDIRECTOS FABRICACION				
MANO DE OBRA INDIRECTA				$ 0,09
Ing. Marco Manzano	$ 339,15	5163,75	$ 0,07	
Ing. Edison Villacrés	$ 148,04	5163,75	$ 0,03	
MATERIA PRIMA INDIRECTA Leche				$ 0,02
Saborizante	$ 3,61	5163,75	$ 0,00	
Fundas	$ 60,38	5163,75	$ 0,01	
Azúcar	$ 9,29	5163,75	$ 0,00	
Conservante	$ 4,70	5163,75	$ 0,00	
Servicios Básicos				$ 0,02
Energía Eléctrica	$ 118,43	5163,75	$ 0,02	
Costo por Unidad de Producción				$ 0,51
Costo de Producción				$ 2.652,63

Ahora determinaremos el costo de producción, pero sin considerar el costo de mano de obra directa e indirecta.

Tabla II.20 Costo de producción de la leche sin considerar costo de mano de obra

	Gasto Mensual	Unidades Producidas	Valor Utilizado	
MANO DE OBRA DIRECTA				$ -
Sr. Jaime Loja	$ -	5163,75	$ -	
MATERIA PRIMA DIRECTA				$ 0,30
Leche Cruda	$ 1.549,13	5163,75	$ 0,30	
COSTOS INDIRECTOS FABRICACION				
MANO DE OBRA INDIRECTA				$ -
Ing. Marco Manzano	$ -	5163,75	$ -	
Ing. Edison Villacrés	$ -	5163,75	$ -	
MATERIA PRIMA INDIRECTA Leche				$ 0,02
Saborizante	$ 3,61	5163,75	$ 0,00	
Fundas	$ 60,38	5163,75	$ 0,01	
Azúcar	$ 9,29	5163,75	$ 0,00	
Conservante	$ 4,70	5163,75	$ 0,00	
Servicios Básicos				$ 0,02
Energía Eléctrica	$ 118,43	5163,75	$ 0,02	
Costo por Unidad de Producción				$ 0,34
Costo de Producción				$ 1.745,54

Queso

Ahora para calcular el costo de producción del queso, consideramos primero el costo de mano de obra directa e indirecta y después sin considerarlo.

Tabla II. 21 Costo de producción del queso considerando costo de mano de obra

	Gasto Mensual	Unidades Producidas	Valor Utilizado	
MANO DE OBRA DIRECTA				$ 0,27
Sr. Jaime Loja	$ 293,05	1098,17	$ 0,2669	
MATERIA PRIMA DIRECTA				$ 1,38

Leche Cruda	$ 1.515,08	1098,17	$ 1,3796	
COSTOS INDIRECTOS FABRICACION				
MANO DE OBRA INDIRECTA				$ 0,31
Ing. Marco Manzano	$ 236,70	1098,17	$ 0,2155	
Ing. Edison Villacrés	$ 103,32	1098,17	$ 0,0941	
MATERIA PRIMA INDIRECTA Queso				$ 0,04
Cuajo	$ 19,19	1098,17	$ 0,0175	
Sal	$ 12,63	1098,17	$ 0,0115	
Calcio	$ 3,54	1098,17	$ 0,0032	
Fermento	$ 454,53	1098,17	$ 0,4139	
Fundas	$ 0,01	1098,17	$ 0,0100	
Servicios Básicos				$ 0,08
Energía Eléctrica	$ 82,66	1098,17	$ 0,0753	
Costo por Unidad de Producción				$ 2,07
Costo de Producción				$ 2.277,14

Tabla II.22 Costo de producción del queso sin considerar costo de mano de obra

	Gasto Mensual	Unidades Producidas	Valor Utilizado	
MANO DE OBRA DIRECTA				$ -
Sr. Jaime Loja	$ -	1098,17	$ -	
MATERIA PRIMA DIRECTA				$ 1,38
Leche Cruda	$ 1.515,08	1098,17	$ 1,3796	
COSTOS INDIRECTOS FABRICACION				
MANO DE OBRA INDIRECTA				$ -
Ing. Marco Manzano	$ -	1098,17	$ -	
Ing. Edison Villacrés	$ -	1098,17	$ -	
MATERIA PRIMA INDIRECTA Queso				$ 0,04
Cuajo	$ 19,19	1098,17	$ 0,0175	
Sal	$ 12,63	1098,17	$ 0,0115	
Calcio	$ 3,54	1098,17	$ 0,0032	
Fermento	$ 454,53	1098,17	$ 0,4139	
Fundas	$ 0,01	1098,17	$ 0,0100	
Servicios Básicos				$ 0,08
Energía Eléctrica	$ 82,66	1098,17	$ 0,0753	
Costo por Unidad de Producción				$ 1,50
Costo de Producción				$ 1.644,07

Yogurt

Para el yogurt detallamos también el cálculo del costo de producción considerando el costo de la mano de obra.

Tabla II.23 Costo de producción del yogurt considerando costo de mano de obra

	Gasto Mensual	Unidades Producidas	Valor Utilizado	
MANO DE OBRA DIRECTA				$ 0,07
Sr. Jaime Loja	$ 1,45	20,00	$ 0,0725	
MATERIA PRIMA DIRECTA				$ 0,30
Leche Cruda	$ 6,00	20,00	$ 0,3000	
COSTOS INDIRECTOS FABRICACION				
MANO DE OBRA INDIRECTA				$ 0,08
Ing. Marco Manzano	$ 1,17	20,00	$ 0,0585	
Ing. Edison Villacrés	$ 0,51	20,00	$ 0,0256	
MATERIA PRIMA INDIRECTA Yogurt				$ 0,38
Fermento	$ 1,40	20,00	$ 0,0700	
Saborizante	$ 0,11	20,00	$ 0,0053	
Colorante	$ 0,09	20,00	$ 0,0044	
Azúcar	$ 2,52	20,00	$ 0,1260	
Envases	$ 3,50	20,00	$ 0,1750	
Servicios Básicos				$ 0,02
Energía Eléctrica	$ 0,41	20,00	$ 0,0204	
Costo por Unidad de Producción				$ 0,86
Costo de Producción				$ 17,15

Calculamos además el costo de producción del yogurt sin tener en cuenta el costo de mano de obra.

Tabla II.24 Costo de producción del yogurt sin considerar costo de mano de obra

	Gasto Mensual	Unidades Producidas	Valor Utilizado	
MANO DE OBRA DIRECTA				$ -
Sr. Jaime Loja	$ -	20,00	$ -	
MATERIA PRIMA DIRECTA				$ 0,30
Leche Cruda	$ 6,00	20,00	$ 0,3000	
COSTOS INDIRECTOS FABRICACION				
MANO DE OBRA INDIRECTA				$ -
Ing. Marco Manzano	$ -	20,00	$ -	
Ing. Edison Villacrés	$ -	20,00	$ -	
MATERIA PRIMA INDIRECTA Yogurt				$ 0,38
Fermento	$ 1,40	20,00	$ 0,0700	
Saborizante	$ 0,11	20,00	$ 0,0053	
Colorante	$ 0,09	20,00	$ 0,0044	
Azúcar	$ 2,52	20,00	$ 0,1260	
Envases	$ 3,50	20,00	$ 0,1750	
Servicios Básicos				$ 0,02
Energía Eléctrica	$ 0,41	20,00	$ 0,0204	
Costo por Unidad de Producción				$ 0,70
Costo de Producción				$ 14,02

Una vez establecido el costo de producción de la leche, queso y yogurt, es necesario calcular la utilidad para cada uno de ellos, cabe recalcar que en nuestro caso tomaremos en cuenta como costo de producción el valor resultante sin considerar mano de obra, por el hecho de que los pagos a los trabajadores y técnico de la planta lo realiza la Espoch, lo que significa que no representa un costo para la planta.

Calculo de la Utilidad

Utilidad: Utilidad, satisfacción o beneficio que se puede obtener al realizar una transacción económica; la utilidad es la base del valor que un individuo confiere a los bienes y servicios que consume.

Utilidad Bruta

Es el resultado de restar el total de ventas obtenidas menos los costos de producción en un periodo dado.

Con lo antes expuesto tenemos que

$$\boldsymbol{Utilidad\ Bruta} = Ventas - Costo\ de\ Producción$$

Para determinar las ventas, es necesario indicar que en este caso se considerará que todo lo producido es vendido, únicamente restando en las ventas de la leche, las unidades que se pierden ya sea por roturas o canjes.

Entonces:

Tabla II.25 Cálculo de la utilidad bruta

	Leche	Queso	Yogurt
Unidades Producidas	5164	1098	20,00
Unidades Perdidas [3-5]%	52	0	0
Total unidades	5112	1098,17	20,00
Costo de Producción	1728,09	1644,07	14,02
Costo de Ventas	2556,06	1976,70	30,00
Utilidad Bruta	827,97	332,63	15,98
Utilidad / Producto	0,16	0,30	0,80

Como podemos apreciar en la tabla, hemos determinado las utilidades con las que trabajaremos en nuestro modelo matemático de maximización de utilidad, es decir estas cantidades serán ingresadas al sistema junto con las restricciones establecidas de a cuerdo a las limitaciones que posee la planta, para de esta manera determinar la cantidad de leches, quesos y yogurt a elaborar, para que la planta obtenga la mayor utilidad.

Gastos de Operación

Los gastos de operación se refiere a los gastos que se involucran en la comercialización de los productos y la depreciación de la maquinaria/equipos y otros bienes que posee la planta, es

necesario definir estos gastos para calcular la utilidad neta que se obtiene mensual y anualmente.

Tabla II.26 Gastos de operación

Gastos de operación	Leche	Queso	yogurt
Depreciación	$ 397,35	$ 57,50	$ 40,51
Transporte	$ 66,67	$ 66,67	$ 66,67
Combustible y lubricantes	$ 66,67	$ 66,67	$ 66,67
Total Gastos Operación	$ 530,69	$ 190,83	$ 173,85

Utilidad Neta

La utilidad neta es la modificación observada en el capital contable de la entidad, después de su mantenimiento, durante un periodo contable determinado, originada por las transacciones efectuadas, hechos y otras circunstancias, excepto las distribuciones y los movimientos relativos al capital contribuido.

La utilidad neta será calculada una vez, determinados la cantidad de leche, queso y yogurt, que deberá elaborar la planta. La misma que será igual a la utilidad máxima resultante, menos los gastos de operación.

$$\boldsymbol{Utilidad\ Neta} = Utilidad\ máxima\ resultante - Gastos\ de\ Operación$$

Función Objetivo y restricciones del Modelo matemático

A continuación mostramos el modelo matemático con los datos ingresados:

F.O: $0.16\,\boldsymbol{X1} + 0.30\,\boldsymbol{X2} + 0.80\,\boldsymbol{X3}$

Rest:

- $\boldsymbol{X1} + 4.5\,\boldsymbol{X2} + \boldsymbol{X3} \leq \boldsymbol{10255}$
- $24.83\,\boldsymbol{X1} + 0\,\boldsymbol{X2} + 0\,\boldsymbol{X3} \leq \boldsymbol{156000}$
- $0\,\boldsymbol{X1} + 5.28\,\boldsymbol{X2} + 0\,\boldsymbol{X3} \leq \boldsymbol{12688}$
- $0\,\boldsymbol{X1} + 0\,\boldsymbol{X2} + 0.10\,\boldsymbol{X3} \leq \boldsymbol{3575}$
- $0.0013\,\boldsymbol{X1} + 0.0164\,\boldsymbol{X2} + 0.0582\,\boldsymbol{X3} \leq \boldsymbol{208}$

Al resolver el modelo matemático, a través de programación lineal con el método de resolución Simplex tenemos los siguientes resultados:

Para alcanzar una utilidad mensual de 3761.59 se debe producir por mes:

Leches en funda: X1 = 6282.72

Quesos: X2 = 127.715

Yogurt: X3 = 3397.56

2.4. El Proceso de planificación

La planificación es el fundamento de la gestión administrativa. Sin un plan no hay bases para establecer cuáles deben ser las acciones que la empresa ha de tomar en el futuro, ni existen referencias que permitan comparar lo conseguido con lo que se hubiera deseado conseguir.

Por tanto, todo plan debe constar de los siguientes elementos:

- Los **objetivos** que la empresa se propone alcanzar en el futuro.
- Los **medios** con los que empresa va a contar para alcanzar esos objetivos.
- El **tiempo** durante el cual la empresa va a disponer de dichos medios. Se conoce como «horizonte temporal de la planificación».

No obstante estos objetivos y por tanto los medios correspondientes, no tienen por qué ser los mismos, cualesquiera que sean los horizontes temporales cubiertos en la planificación. Por este motivo, se divide el tiempo de planificación en intervalos durante los cuales existe una cierta permanencia de los objetivos, lo que permite a su vez una continuidad de los medios dispuestos.

Es norma común que la empresa establezca tres intervalos u horizontes temporales:

1. Largo plazo, también se denomina planificación estratégica.
2. Medio plazo o planificación táctica.

3. Corto plazo. Que, aunque no tiene un nombre especifico, veremos que coincide con lo que en Producción se conoce como Programación.

2.4.1. La planificación estratégica.

En esta etapa de la planificación, es donde la empresa fija globalmente sus grandes objetivos. En algunos casos se establecen como cometidos de carácter genérico, que con posterioridad darán paso a otros más concretos, referidos ya a cada uno de los departamentos de la empresa.

El Plan de Negocios es el documento en el que la empresa establece sus líneas de actuación a largo plazo sobre el mercado, sus productos y los medios de producción necesarios, que ha de disponer para conseguir los objetivos marcados.

2.4.2. La planificación Táctica

La planificación debe ser el vínculo de unión entre los objetivos fijados por la dirección de la empresa con las disponibilidades para conseguirlos; a medida que avanzamos en el tiempo esos objetivos deben hacerse realidad. Es por tanto el factor tiempo el que da a la planificación un sentido práctico a medida que hacemos presente el futuro.

Si los objetivos de la planificación estratégica de la producción se relacionaban más con términos económicos, ahora se aproximan más a realidad del proceso. Es decir:

1. Cuánto hay que producir de cada uno de los productos comercializados.
2. En qué fecha hay que producir esas cantidades.

2.4.3. Proceso de planificación Agregada.

La planificación estratégica implica, entre otras cosas, la generación de un plan comercial o plan de ventas a largo plazo, calculado a partir de la demanda prevista y de los objetivos empresariales para dicho horizonte temporal. Este plan marca las cantidades a vender, las cuales una vez realizados los ajustes necesarios, teniendo en cuenta la capacidad y los

objetivos estratégicos, constituirán el Plan de Producción a Largo Plazo, que puede alterar el Plan de ventas inicialmente previsto. A partir de los mencionados planes se generará el Plan Financiero a Largo Plazo, cuya elaboración puede traer consigo modificaciones en los dos planes anteriormente citados. El conjunto formado por estos tres planes constituye la base del Plan Estratégico o Plan de Empresa; aprobado este, la dirección de operaciones será la responsable del Plan de Producción a Largo Plazo. Para desarrollarlo, deberá pasarse a nivel táctico y concretar dicho plan para el medio plazo mediante la elaboración del denominado Plan Agregado de Producción, este puede considerarse como la misión a cumplir por el Departamento de Operaciones para apoyar la consecución del Plan de Empresa.

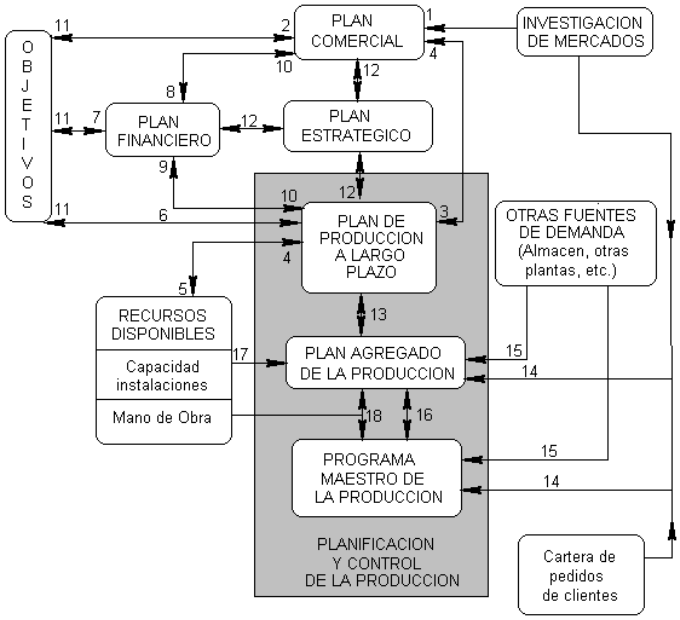

Figura II.13 Del Plan de Empresa al Programa Maestro de Producción (diagrama simplificado).

El establecimiento del Plan Agregado de Producción es altamente complejo, viniendo condicionado por múltiples factores, tales como las distintas fuentes generadoras de la demanda, los objetivos estratégicos, las disponibilidades de recursos materiales y financieros, etc. Podemos definirlo, pues, como un Plan de Producción a medio plazo, factible desde el punto de vista de la capacidad, que permita lograr el plan estratégico de la forma más eficaz posible en relación con los objetivos tácticos del Subsistema de operaciones.

Por otra parte, hemos podido apreciar sus funciones básicas, tales como:

- Permitir la conexión y comunicación del departamento de operaciones con la Alta Dirección y con el resto de las Áreas funcionales.
- Ser el origen del proceso de planificación y control de producción a desarrollar por la dirección de operaciones.
- Ser uno de los instrumentos de control del plan estratégico, en cuyo marco las distintas áreas acuerdan, en términos agregados, lo que va a producirse y lo que va a estar disponible para la venta.

2.4.4. Plan Maestro de Producción.

Introducción.

El plan maestro de producción se utiliza para planificar partes o productos que tienen una gran influencia en los beneficios de la empresa o que asumen recursos críticos y que, por tanto, deben planificarse con especial atención.

Funciones básicas:

- Concretar el plan agregado, tanto en cantidades (de productos finales que deberán ser concluidas) como en el tiempo (estableciendo los momentos de conclusión de los mismos en una base temporal más concreta).
- Facilitar, por su mayor desagregación, la obtención de un plan aproximado de capacidad, el cual permitirá establecer la viabilidad del plan maestro y con ello, la del plan agregado.

Obtención del PMP

El proceso de desagregación pretende convertir las cantidades que constan en el plan agregado (en unidades de familia por mes) en cantidades de productos concretos por semana, de forma que:

- Se cubran las necesidades de fabricación contenidas en el plan agregado.
- Se dividen los retrasos en el servicio de las necesidades de productos.
- Se lleve a cabo todo ello con el menor coste posible.

Par desarrollar este proceso se han elaborado algunos modelos analíticos y de simulación, que consideran incluso las condiciones de capacidad y realizan todo el proceso de una vez. Sin embargo estos métodos no siempre suelen dar las soluciones más óptimas o las mejores, debido a que puede haber varios factores que difieran la planificación de lo real. Por ello, en la práctica, las empresas suelen apoyarse en los procedimientos de prueba y error de diversos tipos, basándose, pues, en la habilidad de los planificadores. Estos últimos pretenden lograr un PMP prospectivo aceptable a través de un proceso tipo que, aunque podría variar en función de la empresa concreta suelen estructurarse en cinco fases:

- Descomposición de las familias del Plan agregado.
- Periodificación de las unidades de producto en los cubos de tiempo.
- Dimensionamiento de los lotes de pedido y determinación de la fecha de obtención de los mismos: PMP inicial.
- Ajuste del PMP inicial en función de la demanda: PMP propuesto e inventarios finales por periodo.
- Determinación de las disponibilidades a comprometer con los clientes.

2.4.5. Planificación de Materiales.

El paso siguiente en la planificación consiste en determinar las fechas en las que habrá que solicitar las materias primas y productos semi-elaborados, que en general llamaremos materiales, para completar la producción de las cantidades fijadas en la Programación Maestra.

Previo a esto debemos conocer primero como elaborar la lista de materiales que se requiere para la elaboración de ciertos productos.

2.4.6. Planificación de Requerimientos de materiales MRP

Los sistemas de planificación de requerimientos de materiales (MRP) integran las actividades de producción y compras. Programan las adquisiciones a proveedores en función de la producción programada. El MRP, es un sistema de planificación de la producción y de gestión de stocks (o inventarios) que responde a las preguntas: ¿qué? ¿Cuánto? y ¿cuándo?, se debe fabricar y/o aprovisionar. El objetivo del MRP es brindar un enfoque más efectivo, sensible y disciplinado para determinar los requerimientos de materiales de la empresa.

2.4.7. Datos de entrada para el desarrollo del MRP Originario.

La técnica comúnmente usada en la industria es la denominada MRP (estas siglas proceden de las palabras «Material Requirement Planning» o su traducción Planificación de las Necesidades de Materiales), existen numerosos paquetes comerciales de desarrollo del MRP basados en el uso de ordenadores, que permiten acortar y racionalizar el procedimiento de planificación. A continuación veremos una de las posibles maneras de llevar a cabo la MRP -1, pero hay que decir que hay otras muchas distintas, aunque sus principios son siempre iguales.

Está técnica determina las necesidades de materiales a partir de las cantidades requeridas de los productos finales. Por tanto hay que fijar en primer lugar las necesidades de éstos y la fecha de entrega, para pasar a continuación a la determinación de los productos intermedios y así ir descendiendo en la estructura de elaboración del producto hasta llegar a la materia prima.

Como veremos, este modo de planificar presenta indudables beneficios en cuanto a costes y organización, frente a otros tradicionales. Previamente deberemos hablar de la manera en que se lleva a cabo esta técnica. Para ello se debe partir de ciertos documentos elaborados por la propia empresa, estos son:

- Programa maestro de producción
- Lista de materiales
- Registros de los inventarios.

- **Lista de materiales.**

La lista de materiales es un compendio de todos los productos, independientemente de su grado de elaboración, que intervienen en la fabricación de un producto. La lista debe contener las especificaciones necesarias para que pueda completarse el producto y las cantidades que interviene en la producción del producto final.

Tomemos el caso de un disquete de computador

El disquete está formado por las siguientes piezas y la relación de estas piezas es la siguiente:

Tabla II.27 Lista de Materiales Tipo Dentada

1065	Conjunto disquete de 3 1/2. (Producto final) (1)
2026	Tapa superior. (1)
2036	Tapa inferior. (1)
2078	Obturador lectura-escritura. (1)
2042	Protector retráctil. (Subconjunto). (1)
7856	Chapa de protección. (1)
9810	Muelle de retracción.
3022	Disco magnético. (Subconjunto). (1)
6501	Parte activa. (1)
6890	Cabezal de arrastre. (1)
9010	Tornillos. (4)

Esta forma que acabamos de describir se conoce como *Lista Dentada*, permite distinguir cada uno de los subconjuntos y sus componentes.

No es esta la única manera de representar la lista de materiales, hay otra denominada *Lista Jerarquizada*, cuya utilidad es mayor que la anterior para el MRP originario. En el caso concreto del disquete anterior dicha representación corresponde al siguiente organigrama.

Los números indicados representan las cantidades en que intervienen los diversos componentes que conforman el producto. Por ejemplo en cada disquete intervienen cuatro tornillos o cada protector contiene un muelle.

La relación entre un producto y su componente es siempre unitaria, en ese sentido.

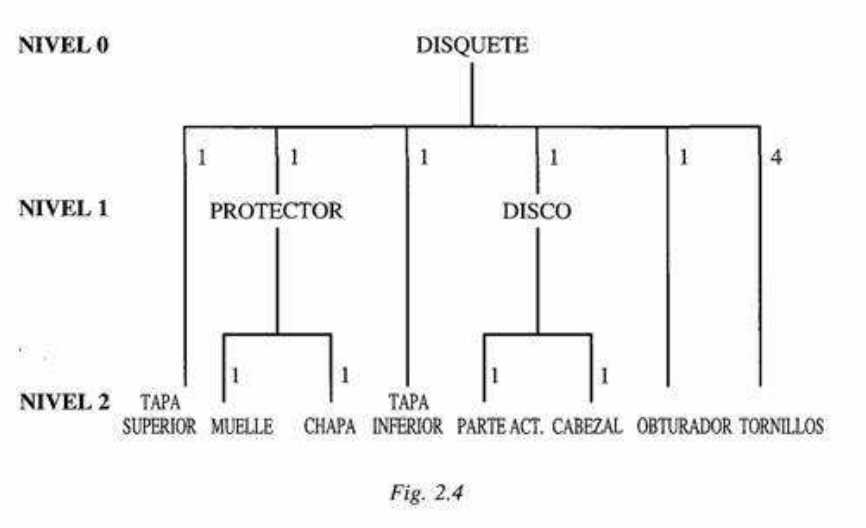

Figura II.14 Lista Jerarquizada

Otro aspecto que destacar es lo referente a los «Niveles». Estos marcan el grado de elaboración de productos y permiten la localización rápida de estos dentro de la lista jerarquizada. Hay que notar que los productos no manipulados por el proceso, lo que constituye la materia prima, siempre están en el nivel más elevado o en la base del triángulo, este es el caso de las tapas o el obturador en el ejemplo anterior.

La confección y actualización de la lista de materiales compete al departamento de Ingeniería, perteneciente a Producción; no obstante la lista de materiales suele ser un documento manejado por diversos departamentos, entre ellos el Comercial, ya que le permite conocer las alternativas que un mismo producto ofrece.

Una lista de materiales donde se especifiquen en porcentajes las opciones del producto final se conoce como SUPERLISTA DE MATERIALES. Su utilización es a efectos de previsión.

Como ejemplo de esto último consideremos el caso de los disquetes, donde se establecen tres opciones: el color de las tapas, la capacidad de información y la comprobación previa.

Ahora la lista de materiales se específica en opciones y cada una ellas viene determinada por el porcentaje de demanda que ha habido de esa opción respecto al total. Ahora la lista de materiales es más una previsión de las posibles opciones demandadas por el mercado, que un instrumento de producción.

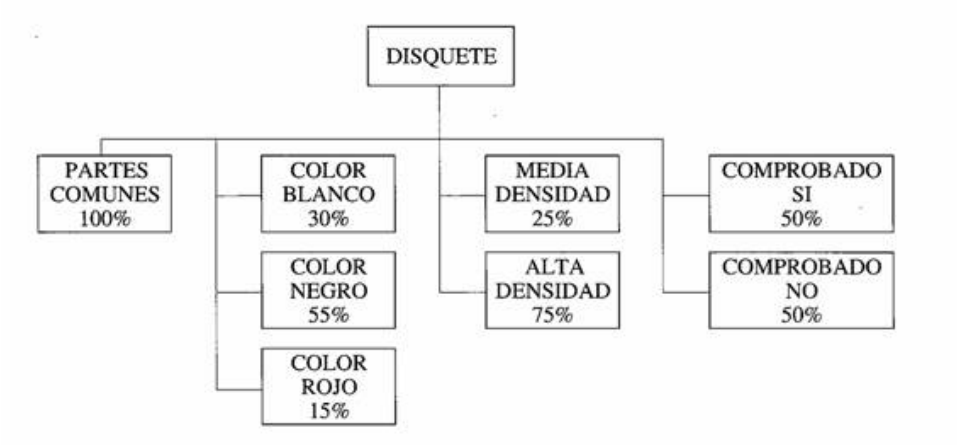

Figura II.15 Super Lista de Materiales

Puesto que los porcentajes indicados también están sujetos a los errores de previsión es necesario definir un stock de seguridad que absorba estas diferencias. Por ejemplo imaginemos que hay una previsión de producción (no una venta real) de 10000 unidades de disquetes. El departamento de Planificación informará al de Abastecimientos que curse la compra de 3000 tapas de color blanco, 5500 de color negro y 1500 de color rojo. Si con posterioridad la demanda real no cumple esta «mezcla» y difiere de las proporciones indicadas, sea el caso de que los clientes soliciten 2000 del tipo rojo, se carecería de 500 unidades, por lo que se comprenderá la procedencia de prevenirse de estas situaciones estableciendo un stock de seguridad, que absorba las diferencias entre las previsiones en los porcentajes de las opciones y la demanda real de éstas.

En otros casos conviene saber cuántos subconjuntos forman el producto final. Se trata de procesos donde las opciones que los clientes pueden solicitar en el producto final son muy numerosas, situación típica de producciones donde se monta por pedido, por ejemplo las opciones en la industria del automóvil, donde el cliente solicita vehículos con dos o tres puertas, cierta motorización, elementos de seguridad, etc. o bien el empaquetado de ciertos

productos formados por una amplia gama de artículos, tal como vajillas, piezas de equipos de cocina, etc.

En estos casos, el producto sólo permanece en situación de componentes modulares, por ejemplo los platos de la vajilla o los elementos de seguridad del coche que no son de serie. Es a partir de esta situación cuando se completa el producto final y la lista de materiales sirve de guía para conseguir la finalización del producto, lo que se conoce como Programación del Ensamblado Final.

- **Registros de los inventarios**

La planificación de la producción requiere de informaciones auxiliares sobre la situación de partida y de aquellos datos que con certeza se cumplirán en el futuro.

Estos datos son en definitiva todos los que atañen al proceso pero en el caso específico de la planificación de materiales los datos necesarios hay que buscarlos en los registros que de los productos y proveedores que tengan en los departamentos de Aprovisionamientos y Control de Inventarios.

En concreto se necesitan los siguientes datos para la Planificación de Materiales:

- Existencias de los productos.
- Stocks de seguridad fijados por el departamento de Planificación.
- Tiempo de suministro, conocido también como plazo de entrega, que en definitiva es el plazo fijado por el suministrador para entregar ciertas cantidades de los productos.
- Ordenes abiertas, o pedidos pendientes de suministro. En realidad se trata de un mismo concepto, aunque con dos puntos de vista diferentes.

Previamente diremos, que cuando hablemos de suministradores, nos referiremos tanto a los proveedores ajenos a la empresa, como a otra parte de la empresa que en cierto momento actúa como proveedora.

En el primer caso, cuando hagamos un pedido a un proveedor y establezcamos una programación de entregas, diremos que son Pedidos Pendientes. En el segundo caso, cuando cursemos una orden de producción a una sección de la propia empresa e igualmente, programemos las entregas, las unidades no entregadas aun forman parte de una Orden Abierta.

En cualquiera de los dos casos para nosotros serán «Entregas pendientes»

- Métodos de pedido, este concepto se refiere a la agrupación de las cantidades. Ya vimos que ciertos tipos de procesos no producen unidades individuales, sino que agrupan para formar lotes, por razones económicas. Se trata por tanto de definir qué cantidades se producen o compran cuando sea necesario.

- Costes de producción, o de compra. Que son datos empleados para rentabilizar económicamente los pedidos.

2.4.8. Esquema Básico de MRP.

El sistema MRP parte de un conjunto de informaciones básicas:

- Las cantidades del producto final a elaborar con indicación de las fechas previstas de entrega, lo cual no es más que el Programa Maestro de Producción.

- La estructura de fabricación y montaje del artículo en cuestión, que recibe el nombre de Lista de Materiales.

- Datos sobre los distintos ítems, como, por ejemplo, los tiempos de suministro y otros que, por lo simplificado del ejemplo, no tuvimos en cuenta (existencias disponibles en almacén, recepciones programadas, etc.). Todos ellos se recogen en el denominado Fichero de Registro de Inventarios.

Dichas entradas eran procesadas por el programa de MRP que, mediante la explosión de necesidades, daba lugar al denominado Plan de Materiales, indicativo de los pedidos a fábrica y a compras, según que el origen del ítem demandado fuese interno o externo. Dicho plan

forma parte de los denominados informes primarios, los cuales constituyen una de las salidas del MRP. Las otras son los denominados informes secundarios o residuales y las transacciones de inventarios. Estas últimas sirven para actualizar el Fichero de Registros de Inventarios en función de los datos obtenidos en el proceso de cálculo desarrollado por el MRP.

La Figura IV.3. Muestra gráficamente el esquema básico de funcionamiento expuestos en los párrafos anteriores. No se considera aquí la capacidad, pues estamos tratando ahora el MRP originario; este aspecto será abordado en el próximo capítulo cuando tratemos el MRP de bucle cerrado. En los próximos apartados desarrollaremos cada uno de los elementos mencionados.

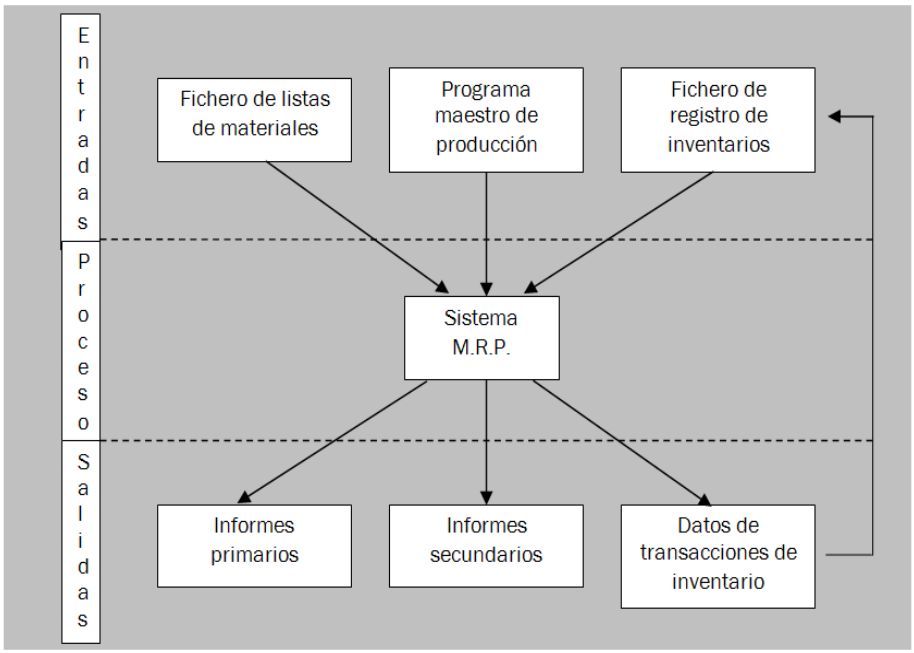

Figura II.16 Esquema básico del MRP originario.

Con lo hasta aquí expuesto, podemos ya definir el sistema M.R.P. originario y enumerar sus características básicas. Quizá la definición más difundida es la que conceptualiza como un sistema de planificación de componentes de fabricación que, mediante un conjunto de procedimientos lógicamente relacionados, traduce un Programa Maestro de Producción en necesidades reales de componentes, con fechas y cantidades.

En cuanto a las características del sistema, se podrían resumir en:

1. Está orientado a los productos, dado que, a partir de las necesidades de éstos, planifica las de componentes necesarios.

2. Es prospectivo, pues la planificación se basa en las necesidades futuras de los productos.

3. Realiza un decalaje de tiempo de las necesidades de ítems en función de los tiempos de suministro, estableciendo las fechas de emisión y entrega de pedidos. En relación con este tema, hay que recordar que el sistema MRP toma el TS como un dato fijo, por lo que es importante que éste sea reducido al mínimo antes de aceptarlo como tal.

4. No tiene en cuenta las restricciones de capacidad, por lo que no asegura que el plan de pedidos sea viable.

Es una base de datos integrada que debe ser empleada por las diferentes áreas de la empresa.

2.4.9. Salida del sistema MRP Originario.

Los outputs del sistema MRP forman un abanico muy amplio y variado, presentándose además en distintos formatos, ya sean informes o mensajes individuales visualizados en la pantalla de los ordenadores o listados obtenidos a través de las impresoras. Todo este conjunto de salidas puede agruparse en salidas primarias y salidas secundarias, que pasamos a describir a continuación.

Salidas primarias del sistema MRP Originario

Se trata del conjunto de informes básicos relativos a necesidades y pedidos a realizar de los diferentes ítems para hacer frente al Programa Maestro de Producción, asó como a las acciones que hay que emprender para conseguirlo. Constituyen la salida fundamental de todo sistema MRP y se pueden concretar en el Plan de Materiales y en los Informes de Acción.

- **El Plan de Materiales**

El Plan de Materiales (compras y fabricación) denominado también Informe de Pedidos Planificados o Plan de Pedidos, es una salida fundamental del sistema MRP, pues contiene los pedidos planificados de todos los ítems. Por regla general, los sistemas MRP suelen tener dos maneras de presentar esta información: modalidad de Cubos de Tiempos (The time-bucket Approach) y modalidad de Fecha/Cantidad (The Date/Quantity Approach). En ambos casos, aparecen en la cabecera los datos descriptivos e identificativos y los factores de planificación de los distintos ítems, al objeto de facilitar la labor del planificador de materiales. En cuanto a la información sobre los pedidos, la primera modalidad la presenta de forma matricial. En cambio, el formato de fecha/cantidad es vertical, por orden de fecha; cada fila, que corresponda a un día laborable dentro de un período, refleja la cantidad correspondiente a un concepto determinado: necesidades brutas, lanzamiento de pedidos planificados, etc. Este segundo formato asocia los datos a la fecha del día que les corresponda dentro del período, de ahí que suministre al usuario una información mucho más precisa que el primer formato. Los sistemas MRP que utilizan esta modalidad se denominan ‹‹sistemas MRP sin cubos. (Bucketless MRP Systems) (T. F. Wallace, 1984).

Creemos que sería de bastante utilidad para los usuarios el poder disponer de ambos formatos de información, y que el sistema procesase y almacenase internamente los datos, asociándolos a fechas de días dentro del cubo de tiempo. Si el coste de procesamiento de dicha información es elevado, sólo restaría, como está ocurriendo en la práctica, utilizar datos agregados asignados a cubos de tiempo, normalmente semanales.

Tabla II.28 Formato del Plan de Materiales (Modalidad de cubos de tiempo)

Descripción ítem……Bomba 21 Clase inventario A
Código identificativo.....21253
Tiempo suministro…….2 semanas Cantidad asignada 0
Stock seguridad……..0
Algoritmo tamaño lote….Lote a lote

Semanas	23	24	25	26	27
Necesidades brutas	50	0	100	60	**10**
Recepciones programadas			150		
Disponibilidades	50	0	0	50	**0**
Necesidades netas				10	**10**
Recepción pedidos planificados				10	**10**
Lanzamiento pedidos planificados		**10**	**10**		

Tabla II. 29 Formato del Plan de Materiales (Modalidad de fechas/cantidad)

Descripción ítem…….Bomba 21 Clase inventario A
Código identificativo……21253
Tiempo suministro……..2 semanas Cantidad asignada 0
Stock seguridad……..0
Algoritmo tamaño lote…Lote a lote

Semana	Día laboral	Tipo de dato	Cantidad	Disponible
23	123	-	-	**50**
23	124	NB	30	**20**
23	126	NB	20	**0**
24	129	PPL	10	**0**
25	132	RP	150	**150**
25	133	NB	60	**90**
25	134	NB	40	**50**
25	135	PPL	10	**50**
26	137	RPPL	10	**60**
26	138	NB	40	**20**
26	139	NB	20	**0**
27	142	RPPL	10	**10**
27	**143**	**NB**	**10**	**0**

NB = Necesidades Brutas,
PPL = Lanzamiento pedidos planificados,
RPPL = Recepción de Pedidos Planificados,
RP = Recepciones Programadas.

Un correcto Plan de Materiales no sólo beneficia al Departamento de Operaciones que, por un lado, podrá disminuir enormemente el tiempo dedicado a aceleración de pedidos y, por otro, el empleo extraordinario de recursos para hacer frente a una producción insuficiente en relación con los compromisos de los clientes. También el Departamento de Compras podrá reducir al mínimo la aceleración de pedidos a proveedores que, tradicionalmente, viene ocupando mucho tiempo al personal de dicha área.

- **Los informes de acción**

Esta salida indica, para cada uno de los ítems, la necesidad de emitir un nuevo pedido o de ajustar la fecha de llegada o la cantidad de algún pedido pendiente. Se pueden visualizar en las pantallas de los terminales así como a través de listados. Aunque es el ordenador quien genera estos informes, es el planificador quien debe tomar las decisiones a la vista de los mismos. Así, cuando el primer período del horizonte de planificación, denominado ‹‹cubo de acción››, aparece el lanzamiento de un pedido planificado, se emitirá el correspondiente pedido siempre que se disponga de sus componentes en la cantidad necesaria. Hecho esto, se llevará a cabo una transacción para actualizar convenientemente el Fichero de Registros de Inventarios. En la próxima programación, el pedido emitido aparecerá en la fila de recepciones programadas y habrá desaparecido de la fila de lanzamiento de pedidos planificados, pues ya no existirá el cubo de tiempo en que estaba situado.

Puede ocurrir, sin embargo, que el planificador decida modificar la cantidad del lote o el tiempo de suministro (TS) especificado para un cierto componente. Así, puede suceder que, al emitir un pedido y asignarle el TS normal, la llegada prevista del correspondiente pedido planificado no sea suficiente (conjuntamente con las disponibilidades del período) para cubrir las necesidades brutas y el stock de seguridad del cubo de tiempo en cuestión. Este hecho puede darse por diversas circunstancias, como, por ejemplo, una cantidad de defectuosos inesperada, cambios de última hora en el PMP, información errónea en el Fichero de Registros de Inventarios, etc. Ante esta eventualidad, el planificador debe determinar si el pedido puede incrementarse o, al menos, acelerarse y obtenerse en un TS inferior al estándar; para ello deberá consultar los informes de capacidad y hablar con el responsable de fabricación (caso de

pedidos internos) o contrastar con el proveedor (si el pedido es externo). Si estas circunstancias son favorables incrementará el pedido o, en su caso, acortará el TS; en caso contrario, podrá optarse por romper el pedido en lotes más pequeños y acelerar parte del mismo.

Salidas secundarias del sistema MRP

Junto con las salidas primarias, tradicionales del MRP, pueden existir otras de utilidad que dependen del paquete de software empleado. A continuación comentamos rápidamente algunas de ellas. Las dos primeras servirán de ayuda al buen desarrollo del proceso de MRP; en cuanto a las restantes, si se valora la información suministrada por el sistema en unidades monetarias, pueden ser de sumar utilidad para distintos Departamentos de la firma.

- **Mensajes individuales excepcionales.**

Son generados como respuesta a las transacciones de inventario introducidas en el Sistema y sólo aparecen en las pantallas de los terminales. Entre estos mensajes se encuentran los siguientes: código identificativo no existente, código de la transacción no existente, exceso en el número de dígitos de la cantidad de un pedido pendiente de recibir o de la cantidad de disponible, etc. Estos mensajes desarrollan un papel fundamental al dotar al sistema de una capacidad de autodetección de errores que ayuda enormemente a mantener la exactitud de los datos.

- **Informe de las Fuentes de Necesidades**.

Este informe (Pegged Requirement Report) relaciona las necesidades brutas de cada ítem con las fuentes que las producen, ya sean demandas de piezas de repuesto o lanzamiento de pedidos planificados de ítems de niveles superiores.

CAPÍTULO III

ESTUDIO COMPARATIVO.

3.1. Introducción.

En este capítulo realizaremos el estudio comparativo de los paradigmas de programación, alineándonos para ello al estándar internacional ISO 9126 que define dentro de sus capítulos los parámetros necesarios a evaluar y que aplicaremos a los paradigmas que están en estudio en esta tesis, en el capítulo inicial de la misma.

3.2. Características del estándar internacional ISO 9126

ISO 9126 es un estándar internacional para la evaluación del Software. Está supervisado por el proyecto SQuaRE, ISO 25000:2005, el cual sigue los mismos conceptos.

El estándar está dividido en cuatro partes las cuales dirigen, respectivamente, lo siguiente: modelo de calidad, métricas externas, métricas internas y calidad en las métricas de uso.

El modelo de calidad establecido en la primera parte del estándar, ISO 9126-1, clasifica la calidad del software en un conjunto estructurado de características y subcaracterísticas de la siguiente manera:

Funcionalidad.- Un conjunto de atributos que se relacionan con la existencia de un conjunto de funciones y sus propiedades específicas. Las funciones son aquellas que satisfacen lo indicado o implica necesidades.

- Idoneidad
- Exactitud
- Interoperabilidad
- Seguridad
- Cumplimiento de normas.

Fiabilidad.- Un conjunto de atributos relacionados con la capacidad del software de mantener su nivel de prestación bajo condiciones establecidas durante un período de tiempo establecido.

- Madurez
- Recuperabilidad
- Tolerancia a fallos
- Conformidad de Fiabilidad

Usabilidad.- Un conjunto de atributos relacionados con el esfuerzo necesitado para el uso, y en la valoración individual de tal uso, por un establecido o implicado conjunto de usuarios.

- Aprendizaje
- Comprensión

- Operatividad
- Atractividad
- Conformidad de Usabilidad

Eficiencia.- Conjunto de atributos relacionados con la relación entre el nivel de desempeño del software y la cantidad de recursos necesitados bajo condiciones establecidas.

- Comportamiento en el tiempo
- Comportamiento de recursos
- Cumplimiento de la Eficiencia

Mantenibilidad.- Conjunto de atributos relacionados con la facilidad de extender, modificar o corregir errores en un sistema software.

- Estabilidad
- Facilidad de análisis
- Facilidad de cambio
- Facilidad de pruebas
- Conformidad de facilidad de mantenimiento

Portabilidad.- Conjunto de atributos relacionados con la capacidad de un sistema software para ser transferido desde una plataforma a otra.

- Capacidad de instalación
- Capacidad de reemplazamiento
- Adaptabilidad
- Co-Existencia

- Conformidad de Portabilidad

La subcaracterística Conformidad no está listada arriba ya que se aplica a todas las características.

Cada subcaracterística (como comportamiento en el tiempo) está dividida en atributos.

Un **atributo** es una entidad la cual puede ser verificada o medida en el producto software. Los atributos no están definidos en el estándar, ya que varían entre diferentes productos software.

El estándar provee un entorno para que las organizaciones definan un modelo de calidad para el producto software. Haciendo esto así, sin embargo, se lleva a cada organización la tarea de especificar precisamente su propio modelo. Esto podría ser hecho, por ejemplo, especificando los objetivos para las métricas de calidad las cuales evalúan el grado de presencia de los atributos de calidad.

Métricas internas.- son aquellas que no dependen de la ejecución del software (medidas estáticas).

Métricas externas.- son aquellas aplicables al software en ejecución.

La calidad en las métricas de uso están sólo disponibles cuando el producto final es usado en condiciones reales.

Idealmente, la calidad interna determina la calidad externa y esta a su vez la calidad en el uso.

Este estándar proviene desde el modelo establecido en 1977 por McCall y sus colegas, los cuales propusieron un modelo para especificar la calidad del software. El modelo de calidad McCall está organizado sobre tres tipos de Características de Calidad:

- Factores (especificar): Describen la visión externa del software, como es visto por los usuarios.

- Criterios (construir): Describen la visión interna del software, como es visto por el desarrollador.

- Métricas (controlar): Se definen y se usan para proveer una escala y método para la medida.

ISO 9126 distingue entre fallo y no conformidad.

Fallo.- es el incumplimiento de los requerimientos previos.

No conformidad.- es el incumplimiento de los requerimientos especificados.

Una distinción similar es la que se establece entre validación y verificación.

3.3. Definición de los parámetros a utilizar para el Estudio Comparativo.

De acuerdo a lo expuesto en lo anterior, el estándar 9126, proporciona un conjunto de características y subcaracterísticas para evaluar el software, en nuestro caso de estudio tomaremos en cuenta la característica **Eficiencia** y las subcaracterísticas de la misma para medir la cantidad de tiempo y recursos del computador por cada modelo matemático construido en los tres paradigmas.

- **Eficiencia**

Según (ISO 9126: 1991, 4.4), se refiere a un conjunto de atributos que tienen que ver con la relación entre el nivel de rendimiento del software y la cantidad de recursos utilizados, bajo condiciones señaladas.

Las subcaracterísticas definidas dentro de Eficiencia son:

***Comportamiento en el tiempo*:** Atributos del software que tienen que ver con la **respuesta y los tiempos de tramitación** y sobre las **tasas de rendimiento en el desempeño** de su función. (ISO 9126: 1991, A.2.4.1)

Es decir es la capacidad del producto software para proporcionar tiempos de respuesta, tiempos de proceso y potencia apropiados, bajo condiciones determinadas.

Comportamiento de recursos: Atributos del software que tienen que ver con la **cantidad de recursos utilizados y la duración de esos usos en el desempeño** de su función. (ISO 9126: 1991, A.2.4.2)

Es la capacidad del producto software para usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su función bajo condiciones determinadas.

Cumplimiento de la eficiencia: Capacidad del producto software para adherirse a normas o convenciones relacionadas con la eficiencia.

Las características de calidad no permiten su medición directa debido a la manera de cómo han sido definidas, por tal motivo, se deben establecer métricas que se correlacionen con características que puedan ser cuantificables, cada característica e interacción cuantificable del software en un determinado entorno puede ser establecido como una métrica.

Por esta razón surge la necesidad de establecer parámetros que puedan ser expuestos a medición, a continuación detallamos cada uno de los ellos, que vienen a ser los parámetros que utilizaremos para nuestro estudio comparativo.

Parámetros

- **Líneas de Código**

La definición de línea de código, es básica para la mayor parte del software. El significado de línea de código varía de un lenguaje a otro, pero también dentro de un mismo lenguaje de programación.

Son cada una de las líneas de un archivo de código fuente de un programa de software. Habitualmente en cada línea se ejecuta una instrucción que tiene que ejecutar el programa.

En ocasiones los programadores hablan del número de líneas de código que tiene cierto programa para hablar de la magnitud o complejidad de este.

- **Tiempo de Ejecución**

Se denomina Tiempo de ejecución (Runtime en inglés) al intervalo de tiempo en el que un programa de computadora se ejecuta en un sistema operativo. Este tiempo se inicia con la puesta en memoria principal del programa, por lo que el sistema operativo comienza a ejecutar sus instrucciones. El intervalo finaliza en el momento en que éste envía al sistema operativo la señal de terminación, sea ésta una terminación normal, en que el programa tuvo la posibilidad de concluir sus instrucciones satisfactoriamente, o una terminación anormal, en el que el programa produjo algún error y el sistema debió forzar su finalización.

En tiempo de ejecución pueden darse errores inesperados llamados runtime errors, que pueden ser controlados a través de mecanismos llamados manejos de excepciones.

- **Memoria Utilizada**

Es un recurso importante de la computadora ya que determina el tamaño y el número de programas que pueden ejecutarse al mismo tiempo, como también la cantidad de datos que pueden ser procesados instantáneamente, las instrucciones del programa son copiados en memoria que después se las extrae para su análisis y ejecución. Para calcular la cantidad de memoria, utilizaremos el monitor del sistema, el mismo que nos permitirá monitorear el uso de este recurso.

- **Tiempo de Procesador**

Se refiere al tiempo que utiliza un proceso o programa para realizar una tarea determinada.

3.4. Definición de las escalas de Evaluación.

Este valor no muestra el nivel de satisfacción. Con este fin, estas escalas deben ser divididas en rangos correspondientes a los distintos grados de satisfacción de los requerimientos.

Dado que la calidad se refiere a las necesidades dadas, se pueden establecer niveles de calificación que no están generalizados, es decir una escala. Los mismos deben definirse específicamente para cada evaluación.

El resultado, es decir, el valor medido se asigna en la escala.

Para el análisis de los resultados que se obtendrán se tomó como base comparaciones cuali-cuantitativas definidas bajo nuestro propio criterio, envase a la importancia de los parámetros a evaluar los cuales ya han sido definidos con anterioridad, para el caso presentamos a continuación la tabla II.30 que detalla los valores

Tabla III.30 Parámetros de medición con su respectivo valor porcentual

Parámetro	Valor Porcentual
Líneas de Código	10%
Tiempo de Ejecución	40%
Memoria	20%
Procesador	30%
Total	**100%**

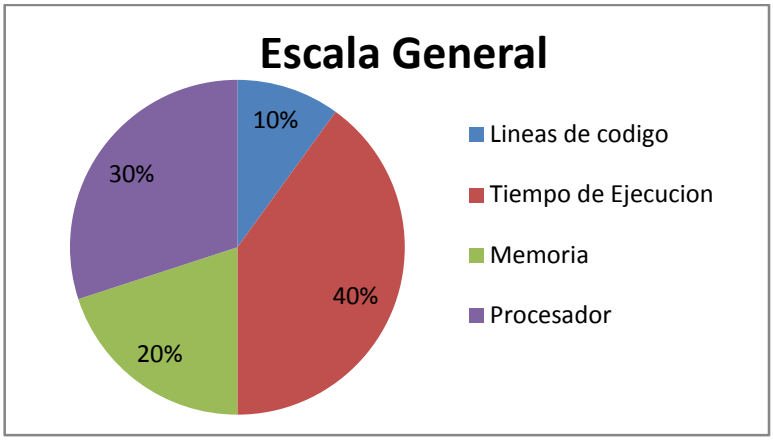

Figura III. 17 Escala General de los parámetros de medición

Con la escala detallada y una vez evaluados los paradigmas de programación obtendremos como resultado el paradigma que mejor optimice los recursos computacionales.

Para que la medición sea más precisa detallaremos escalas para cada uno de los parámetros, donde los valores que se sigan obteniendo serán finalmente sumados y mapeados en la escala general definida.

- **Líneas de código.**

Este parámetro suele ser importante al momento de estimar recursos utilizados, se pretende comparar el número de líneas de código que se deben escribir para construir un modelo matemático para producción en el Paradigma de Programación Estructurada, Orientado a Objetos y en el Paradigma Funcional, los mismos que fueron implementados en los lenguajes de programación Pascal, C# y F#, respectivamente.

Para determinar este valor utilizaremos un proceso de conteo directo, luego se procederá a comparar la cantidad de líneas de código resultantes, en donde el menor número que puede pertenecer a cualquiera de los paradigmas, será el que más optimice este recurso.

Para el conteo de las líneas de código utilizaremos la herramienta **LocMetrics**, la misma que cuenta el total de líneas de código (LOC), líneas de código en blanco (BLOC), líneas

de código comentadas (CLOC), líneas de código y comentarios (C & SLOC), líneas de código fuente lógicas (SLOC-L) y el número de palabras comentadas (CWORDS).

Para la medición de las líneas de código utilizaremos además la siguiente tabla que nos servirá para obtener el valor y el calificativo de cada uno de los demos a evaluar.

Tabla III.31 Escala Cuali-cuantitativa para el parámetro líneas de código

Calificativo	Parámetro (Líneas de código)	Valor
Excelente	[50 - 80]	100
Muy bueno	[81 - 160]	80
Bueno	[161 - 210]	60
Regular	[211 - 290]	40
Malo	[291 - mas]	20

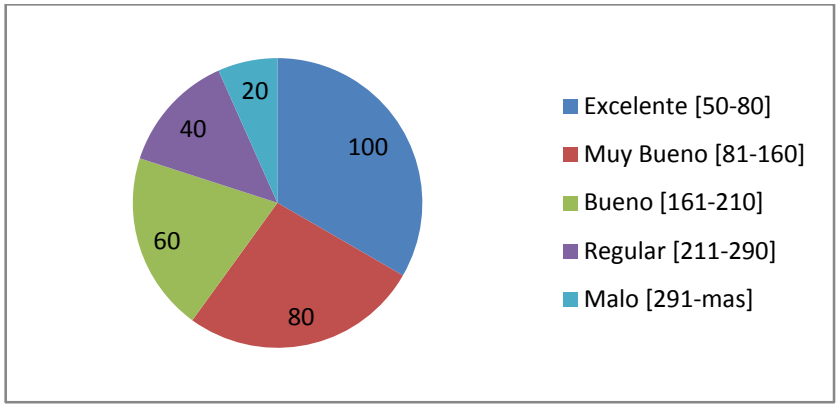

Figura III.18 Escala definida para líneas de código.

Con este conteo indicaremos el que contenga un menor número de líneas de código será excelente, en lo que se refiere a la escritura de un modelo matemático lineal para producción, ya que en el paradigma que resulte excelente, será aquel que permitió reducir líneas de código pero donde el grado de abstracción para implementar dicho modelo será

mucho mayor del que utilizó mayor número de líneas de código ya que el programador deberá haber aprendido nuevas técnicas de programación propias del paradigma.

- **Tiempo de Ejecución**

Con este parámetro determinaremos el tiempo de respuesta, es decir cuál es el tiempo estimado para completar una tarea. Para obtener el tiempo de ejecución, utilizaremos una librería propia de los lenguajes de programación en los que hemos desarrollado los modelos matemáticos.

En Pascal que representa al paradigma estructurado, para realizar el cálculo del tiempo de ejecución utilizamos la librería **DOS**, que nos permite utilizar la función GetTime(), para determinar el tiempo que tarda en ejecutar el programa.

Mientras que en C# y F#, que representan al paradigma orientado a objetos y funcional respectivamente, dado que los dos lenguajes de programación pertenecen a la plataforma .Net, utilizaremos la librería **System.Diagnostics**, que nos proporciona las funciones Stopwatch y TimeSpan.

Obtendrá mayor calificación el paradigma cuyo tiempo de ejecución sea el menor.

t = menor tiempo de ejecución

Tabla III.32 Escala Cuali-cuantitativa para el parámetro Tiempo de ejecución

Parámetro	Valor	Calificativo
t	100	Excelente
(t + 30%t)	80	Muy Bueno
(t + 60%t)	60	Bueno
(t + 80%t)	40	Regular
> (t + 100%t)	0	Malo

- **Memoria Utilizada**

Se trata de medir la cantidad de memoria utilizada al momento de la ejecución del modelo matemático construido en los paradigmas de programación antes mencionados.

Para obtener los datos, utilizaremos el **Administrador de Tareas de Windows,** pestaña **Procesos** donde se listan todos los procesos que se están ejecutando, seleccionamos el proceso que se crea al ejecutar el demo en el paradigma respectivo, del que consideramos la opción **Uso de memoria,** la que viene dada en Kilobytes e indica la cantidad de bytes de memoria que se encuentran actualmente ocupados.

De igual manera obtendrá mayor calificación el paradigma que tenga la menor cantidad de memoria.

m = menor cantidad de memoria

Tabla III.33 Escala Cuali-cuantitativa para el parámetro memoria utilizada

Parámetro	Valor	Calificativo
m	100	Excelente
(m + 30%m)	80	Muy Bueno
(m + 60%m)	60	Bueno
(m + 80%m)	40	Regular
> (m + 100%m)	0	Malo

- **Tiempo de Procesador**

Otra métrica a tomar en cuenta es el uso del procesador, es decir cuánto recurso del procesador utiliza el modelo matemático en su ejecución.

Utilizaremos el **monitor del sistema** para saber la carga del procesador con el contador **% de tiempo de procesador** que es el porcentaje de tiempo que el procesador invierte para ejecutar un proceso.

Para establecer la escala de calificación, consideremos que el obtendrá mayor calificación el paradigma cuyo tiempo de uso de procesador sea menor.

tp = menor tiempo de uso del procesador

Tabla III.34 Escala Cuali-cuantitativa para el parámetro tiempo de procesador

Parámetro	Valor	Calificativo
tp	100	Excelente
(tp + 30%tp)	80	Muy Bueno
(tp + 60%tp)	60	Bueno
(tp + 80%tp)	40	Regular
> (tp + 100%tp)	0	Malo

3.5. Evaluación de cada parámetro por paradigma.

Una vez definida la escala de valoración y los parámetros a evaluar, mostramos enseguida los resultados obtenidos en cada una de las mediciones.

Cabe recalcar que los datos para los parámetros memoria y tiempo de procesador, obtuvimos modificando el programa para que tenga repeticiones continuas y de esta manera se pueda observar y medir el desempeño del mismo.

- **Líneas de Código**

Utilizando la herramienta LocMetrics, obtuvimos la cantidad de líneas de código, utilizadas por cada paradigma para resolver el mismo modelo matemático.

Tabla III.35 Resultados obtenidos de la medición de líneas de código.

Paradigmas de Programación	Número de Líneas de código
Estructurado	304
Orientado a Objetos	299
Funcional	120

- **Tiempo de Ejecución**

Con respecto al tiempo de ejecución que cada uno de los demos utilizan para resolver el modelo y retornar el resultado, obtuvimos los datos de la Tabla III.36.

Tabla III.36 Resultados obtenidos de la medición de Tiempo de ejecución.

Paradigmas de Programación	Tiempo de Ejecución
Estructurado	1420 ms
Orientado a Objetos	11 ms
Funcional	18 ms

- **Memoria Utilizada**

La cantidad de memoria utilizada, obtenida se refleja en la Tabla III.37, donde se indica por paradigma la cantidad de memoria que ocupa cada demo en su ejecución.

Tabla III.37 Resultados obtenidos de la medición de memoria utilizada

Paradigmas de Programación	Memoria
Estructurado	2936 Kb
Orientado a Objetos	4580 Kb
Funcional	10584 Kb

- **Tiempo de Procesador**

El porcentaje de tiempo que usa el procesador al ejecutar los demos lo apreciamos en la Tabla III.38.

Tabla III.38 Resultados obtenidos de la medición de Tiempo de procesador

Paradigmas de Programación	Procesador (%)
Estructurado	51.563
Orientado a Objetos	47.385
Funcional	42.195

3.6. Análisis de resultados y conclusiones.

En este apartado realizaremos el análisis de los datos recolectados, y el conteo de los puntos obtenidos por cada paradigma en cada uno de los parámetros definidos para la evaluación.

Tabla III.39 Resultados de la medición de cada uno de los parámetros por paradigma

Parámetro / Paradigma	Líneas de Código		Tiempo de Ejecución		Memoria		Procesador	
	valor	**calificación**	**valor**	**calificación**	**valor**	**calificación**	**valor**	**calificación**
Estructurado	20 Puntos	Malo	0	Malo	100	Excelente	80	Muy Buena
Orientado a Objetos	20 Puntos	Malo	100	Excelente	60	Bueno	80	Muy Buena
Funcional	80 Puntos	Muy Bueno	40	Regular	0	Malo	100	Excelente

Análisis

Los puntajes obtenidos por cada paradigma, luego de realizar el conteo y mapearlos en la escala general de calificación, se detallan a continuación.

Tabla III.40 Sumatoria de los puntos obtenidos por paradigma

	Paradigma estructurado (LP. Pascal)	Paradigma Orientado a Objetos (LP. C#)	Paradigma Funcional (LP. F#)
Líneas de Código	2	2	8
Tiempo de ejecución	0	40	16
Memoria	20	12	0
Procesador	24	24	30
Total	**46**	**78**	**54**

- De acuerdo a la Tabla III.39 y a la Tabla III.40, además una vez realizado el conteo de los puntos obtenidos, podemos concluir que el paradigma más adecuado para implementar modelos matemáticos para planificación de producción es el Paradigma Orientado a Objeto con su Lenguaje de Programación C#.
- El paradigma Orientado a Objetos obtuvo 78 sobre un total de 100 puntos, obteniendo de esta manera el mayor puntaje.
- El paradigma Funcional sigue a la lista con un puntaje de 54 puntos y por último el Paradigma Estructurado con 46 puntos.
- Si bien es cierto que el paradigma ganador resulto ser el Orientado a Objetos con el LP. C#, sin embargo el paradigma funcional con el LP. F#, proporciona muchas bondades para cálculos matemáticos, debido a su notación algo matemática, la misma que parece naturalmente atractiva para aquellos profesionales cuyo domino principal es representado en notación matemática, dominios como el financiero, científico y la computación técnica.
- Una de las cosas que deben ser tomadas muy en cuenta al momento de programar en F#, es tener las suficientes destrezas y conocimiento acerca de este lenguaje de programación, ya que solo de este modo, podremos utilizar al máximo cada una de las potencialidades de F#.

- F# suele utilizar funciones netamente recursivas, agilitando de este modo el proceso de cálculo y disminuyendo el número de líneas de código, por tal motivo, debido a que la programación lineal con su método de resolución Simplex, depende de otros vectores y funciones para la obtención del resultado, no se utiliza recursividad para el cálculo del mismo.

- C#, es un lenguaje de programación muy poderoso, que ofrece muchas ventajas al momento de implementar modelos matemáticos, pero cabe recalcar que lo que le hace especial es el dominio que la mayoría de los programadores de la plataforma .net, tienen sobre él, permitiendo de esta manera que C# sea aprovechado al máximo. Y se pueda optimizar los recursos computacionales que este consuma al momento de su ejecución.

- El Paradigma de Programación ganador del estudio comparativo es el Paradigma Orientado a Objetos, con su lenguaje de programación representante C#.

3.7. Comprobación de la hipótesis

Una vez realizado el estudio de los paradigmas de programación y tomando en cuenta características a nivel de optimización de recursos computacionales, basadas en el estándar internacional ISO 9126, se puede concluir que el paradigma utilizado en la implementación del modelo matemático, construido para la planta de lácteos de la ESPOCH, es el que obtuvo los mejores resultados, para la hipótesis planteada para la presente investigación:

CAPÍTULO IV

SISTEMA PARA LA PLANIFICACIÓN DE PRODUCCIÓN Y CONTROL DE INVENTARIOS PARA LA PLANTA DE LÁCTEOS ESPOCH

4.1. Introducción

La planta de lácteos de la ESPOCH ofrece el servicio de producción de leche pasteurizada en funda, queso rectangular y yogurt.

Sirve además de apoyo académico a los estudiantes de la facultad, ya que los mismos pueden realizar prácticas de elaboración de lácteos y de esta manera adquirir destrezas y fortalecer sus conocimientos.

La planta fue creada con el objetivo de brindar facilidades para los estudiantes y generar recursos económicos pero, es necesario que la misma genere utilidades, determinando los requerimientos de insumos necesarios para la elaboración de los productos, controlando la utilización de los mismos, calculando las cantidades de producto terminado a fabricar, así

como los componentes necesarios y las materias primas a comprar para poder satisfacer la demanda del mercado.

4.2. Presentación

El sistema propuesto tiene como objetivo ayudar a administrar el inventario, producción y ventas de la planta de lácteos de la ESPOCH, con la creación de una aplicación de escritorio que contenga un modulo de control de inventario, un modulo de producción, ventas y planificación; que permita mantener la organización de las entradas y salidas de insumos, y de productos elaborados, llevar el control de ventas y planificar la cantidad de leche en funda, queso y yogurt a elaborar con el fin de obtener la máxima utilidad.

La implementación de SystemLac, permitirá al personal de la planta de lácteos de la ESPOCH:

- Contar con una base de datos solida
- Se espera que el desarrollo del sistema SystemLac, logre reducir el tiempo de elaboración de reportes.
- Llevar de forma organizada el control de insumos, producción y ventas.
- Conocer de antemano la cantidad de productos que se debe elaborar como una alternativa para que la utilidad sea mayor.

4.3. Metodología de Desarrollo

La metodología usada para el desarrollo del Sistema de Planificación de producción y control de inventarios SYSTEMLAC es Microsoft Solución Framework, específicamente el modelo de procesos.

MSF proporciona prácticas comprobadas para planificar, crear y poner en marcha soluciones exitosas. En oposición a la metodología prescriptiva, MSF proporciona una estructura flexible y escalable para conocer las necesidades de una organización o equipo encargado de un proyecto de cualquier tamaño. La orientación de MSF consiste en

proporcionar principios, modelos y disciplinas para manejar personas, procesos y elementos de tecnología con los que se encuentra la mayoría de proyectos.

El modelo de procesos de MSF combina los mejores principios de los modelos en cascada y espiral ya que considera la planeación basada en puntos de revisión o hitos del modelo en cascada y los beneficios de la retroalimentación y creatividad del modelo en espiral.

4.4. Visión

En esta fase el equipo y el cliente definen los requerimientos del negocio y los objetivos generales del proyecto. La fase culmina con el hito Visión y Alcance aprobados.

Objetivo:

Obtener una visión del proyecto compartida, comunicada, entendida y alineada con los objetivos del negocio. Además, Identificar los beneficios, requerimientos funcionales, sus alcances y restricciones; y los riesgos inherentes al proceso.

Posteriormente a la implementación del sistema para la planificación de producción y control de inventarios, SystemLac, la planta de lácteos de la ESPOCH proporcionará un enfoque más objetivo, sensible y disciplinado al momento de determinar los requerimientos de materiales de la planta.

El sistema calculará las cantidades de producto terminado a fabricar, los componentes necesarios y emitirá mensajes que indican que se debe adquirir un determinado insumo, necesario para que la producción se realice.

4.5. Metas de Diseño

Se refiere a la descripción de las características operacionales del sistema.

4.5.1. Disponibilidad

El producto debe ser disponible únicamente para los usuarios que se encuentren registrados. El sistema cubre los requerimientos de disponibilidad.

4.5.2. Confiabilidad

El producto debe ser confiable, para poder realizar auditorías. Mantenibilidad: El sistema debe ser fácil de mantener, además se debe dejar documentación para realizar posibles actualizaciones.

4.5.3. Amigable

El sistema debe presentar interfaces amigables, intuitivas y de fácil interacción con el usuario.

4.5.4. Flexibilidad

El sistema debe ser flexible a los crecientes cambios de la planta de lácteos de la ESPOCH.

4.5.5. Seguridad

Se deben aplicar políticas de seguridad que garanticen la confidencialidad de la base de datos y de la aplicación.

4.6. Fase de Envisionamiento

4.6.1. Introducción

Esta fase tiene como objetivo conseguir una idea clara de las metas y objetivos del sistema, basando su solución en la problemática que se desea resolver. Definiremos también los problemas que el sistema procura solucionar.

4.6.2. Análisis de la Problemática

4.6.2.1. Antecedentes

En la actualidad la planta realiza el control de la entrada y salida de insumos, de forma manual, y de la misma manera el control de las ventas.

El técnico encargado de la planta, lleva el control diario de la producción, de la materia prima e insumos en libros, emite informes mensuales de las ventas, detallando por producto las ventas realizadas en cada mes.

Mantiene además respaldos en Excel de la recepción de leche cruda, producción de leche, queso y yogurt y también de las ventas detalladas por cliente y producto, debidas a que los precios de venta, varía según el cliente.

Cabe aclarar que en la actualidad no se cuenta con una planificación, que organice que producir y en qué cantidades, y de esta manera la planta genere más utilidad y mejore los beneficios que esta presta.

La manera como se lleva el control hoy en día, suele ser tediosa y demorada, sin contar que la integridad de los datos se ve amenazada.

4.6.2.2. Control actual de inventario, producción y ventas

La planta de lácteos de la ESPOCH, en la actualidad lleva el control de materia prima e insumos para la producción de leche pasteurizada en funda, queso y yogurt, en documentos Excel, los mismos que detallan la cantidad de litros de leche cruda receptada diariamente y la asignación de la leche cruda a la producción ya sea de leche en funda, queso, yogurt o para prácticas de los estudiantes.

Una de las políticas de la planta es producir leche y queso de forma alternada, es decir se produce un producto por día a la vez, salvo escasas excepciones en las que se produce leche y queso, en el caso del yogurt únicamente se produce bajo pedido y esto suele darse sin mucha frecuencia, debido a esta característica la producción de yogurt es mucho menor

a la producción de leche y queso, a pesar de que el yogurt genera mayor utilidad y cuenta con una capacidad de producción que puede cumplir con mayor producción a la actual.

Tabla IV.41 Forma actual de Control de recepción de leche cruda y producción (Enero 2008)

Fecha	Recepción				Producción			Estudian	Yogurt	Crema	
	Mañana	Tarde	Compra	Total	Leche pas	Leche que	Quesos	Litros			
1	187			187	187						
2	176	195		371		371	81				
3	212	193		405	212	193	43				
4	240	153		393	240	153	34				
5	230	172		402	230	172	38				
6	213	167		380	360				20		
7	240	177		417		417	91				
8	180	200		380		380	82				
9	206	148		354	354						
10	196	162		358	358						
11	186	179		365	135	230	50				
12	180	200		380		380	81				
13	200	184		384	384						
14	184	173		357	357						
15	176	157		333	333						
16	184	170		354	174			180			
17	160	164		324	324						
18	180	138		318	318						
19	199	165		364	214			150			
20	184	160		344		344	76				
21	190	144		334	334						
22	185	173		358		358	79				
23	170	160		330	330						
24	143	166		309	9			300			
25	163	130		293	293						
26	164	150		314	314						
27	180	160		340		340	72				
28	173	140		313		313	69				
29	180	150		330	180	150	33				
30	178	150		328	153			175			
31	198	144		342		242	53	100			
TOTAL	5837	4924		10761	5793	4043	882	905	20		
Los quesos rectangulares tienen un peso de 720 a770 gr. Utilizando 4.4 a 4.6 lt por queso y esto depende de localidad de la leche.											

Para elaborar cada uno de los productos que la planta pone a disposición, adicional de la leche cruda se necesitan insumos extras, los mismos son adquiridos cada vez que se están quedando sin existencia de los mismos, sin embargo cabe recalcar que a partir del siguiente año todo los insumos, equipos, material de oficina, maquinaria, es decir todo de lo que la planta necesite abastecerse se lo realizará a través de los organismos POA y PAC.

Los insumos que se necesita para elaborar leche, queso y yogurt, se describen en el capítulo 2, apartado 2.3, que detalla el insumo, el precio y la cantidad de insumo por litro de leche.

En lo que se refiere a ventas, los reportes se llevan también en Excel, donde se detalla la cantidad de cada producto vendida por cliente, el precio de venta por cliente debido a que este varía y el total de ventas diaria.

Si bien es cierto se ha llevado por años esta forma de control y organización de los datos de producción y ventas de la planta, ocasionando muchas veces perdida de información valiosa, datos poco íntegros y gran cantidad de tiempo invertido en la elaboración de informes mensuales, por la manera manual de llevar el control de la información.

Por esta y otras razones se plantea el desarrollo de SystemLac, un aplicación de escritorio que permitirá llevar el control de la materia prima e insumos existentes en inventario, control de la producción conociendo las cantidades necesarias por cantidad de productos a elaborar, control en las ventas diferenciando por producto y cliente, y un modulo de planificación que permitirá saber en qué cantidades debo producir leche en funda, queso y yogurt para maximizar la utilidad de la planta, en base a la utilidad que en la actualidad genera la planta por unidad de producto.

4.6.2.3. Alcance del sistema

En esta sección se describen algunas de las principales características del sistema, es decir las necesidades que serán cubiertas con el desarrollo del mismo:

Tabla IV.42 Características del sistema SystemLac

Características
Ingresar el detalle de la compra de insumos por factura, especificando la cantidad, nombre del insumo y el precio.
Ingresar datos del proveedor y mostrar la lista de proveedores.
Visualizar los ECG de un paciente desde cualquier lugar.
Control de la entrada y salida de insumos (Kardex).
Permite dar de baja un insumo, por perdida, rotura y fecha de vencimiento.
Ingresar los productos que se elaboran en la planta, con su detalle y lista de materiales necesarios para su elaboración.
Ingresar una orden de producción.
Control de productos elaborados (Kardex).
Ingreso de clientes
Ingreso de detalle de venta.
Reporte de ventas
Determinar la cantidad de leche, queso y yogurt a elaborar para maximizar la utilidad.

4.7. Fase de Planificación

4.7.1. Introducción

La Fase de Planificación tiene como objetivo enumerar los requerimientos del sistema ya sean estos tecnológicos, funcionales, no funcionales, de interfaces externas, es decir, especificar la funcionalidad que ofrecerá el sistema SystemLac.

4.7.2. Componentes Principales

La aplicación consta de los módulos descritos a continuación:

- Modulo de Administración.
- Modulo de Inventario.
- Modulo de Producción.
- Modulo de Ventas.

- Modulo de Planificación.

Modulo de Administración.- permite crear las cuentas de los usuarios que accederán al sistema, además modificar o eliminar cuentas de usuario ya existentes.

Modulo de Inventario.- este modulo se encuentra dividido en: Proveedor, Insumos, Facturas y Bajas de Insumos.

Proveedor: permite ingresar los datos de los proveedores, visualizar y obtener la lista de proveedores, ingresados en el sistema.

Insumos: proporciona opciones para ingreso y lista de insumos, control mínimo de insumos (Kardex) y reporte de compra de insumos.

Facturas: en esta opción podemos realizar el ingreso de facturas de compra, además generar una lista de compras y el reporte de las mismas

Bajas de Insumos: brinda la opción de dar de baja un insumo por perdida, rotura o fecha de vencimiento y listar los insumos dados de baja

Modulo de Producción.- consta de las opciones Producto, Producción y Costos.

Producto: admite el ingreso de un nuevo producto que se requiera elaborar, así como también permite mostrar la lista de productos con los que cuenta la planta.

Producción: permite realizar una Orden de producción y Fin de producción, también obtener un Listado de órdenes de producción y Órdenes de producción terminadas, además llevar un Control de productos elaborados (Kardex).

Costos: permite obtener el costo de producción de una determinada orden de producción y el costo de producción por unidad.

Utilidad: muestra la utilidad que genero una producción determinada y además la utilidad unitaria por producto.

Modulo de Ventas.- cuenta con opciones para Clientes, Ventas y Reportes.

Modulo de Planificación.- provee la opción de planificar la cantidad de leche, queso y yogurt que se debe elaborar, si se desea maximizar la utilidad.

4.7.3. Recursos hardware y software

Es necesario especificar el recurso hardware y software necesario para la implementación de nuestro sistema.

Las características hardware necesarias podemos apreciarlas en la tabla IV.45.

Tabla IV.43 Características Hardware para la implementación del sistema

Uso de Equipos	Características
Maquina 1	Intel Core 2Duo 1.8 GHz 1 Gb de RAM 160 GB Disco Duro Monitor LCD 17 pulgadas DVD Writer Teclado USB Mouse Ps2
Maquina 2	Laptop Intel Core 2Duo Ghz 3 Gb de Ram 160 Gb de Disco Duro Monitor de 15 pulgadas

Recurso software necesarios

Tabla IV.44 Recursos Software para la implementación del sistema

Programas
Sistema Operativo Windows XP Professional Service Pack 2
Visual Studio. Net 2005

Microsoft SQL Server 2005
Microsoft Office 2007
Rational Rose 2002
Adobe Photoshop CS4
Adobe Acrobat 9 Pro

4.7.4. Cronograma de trabajo

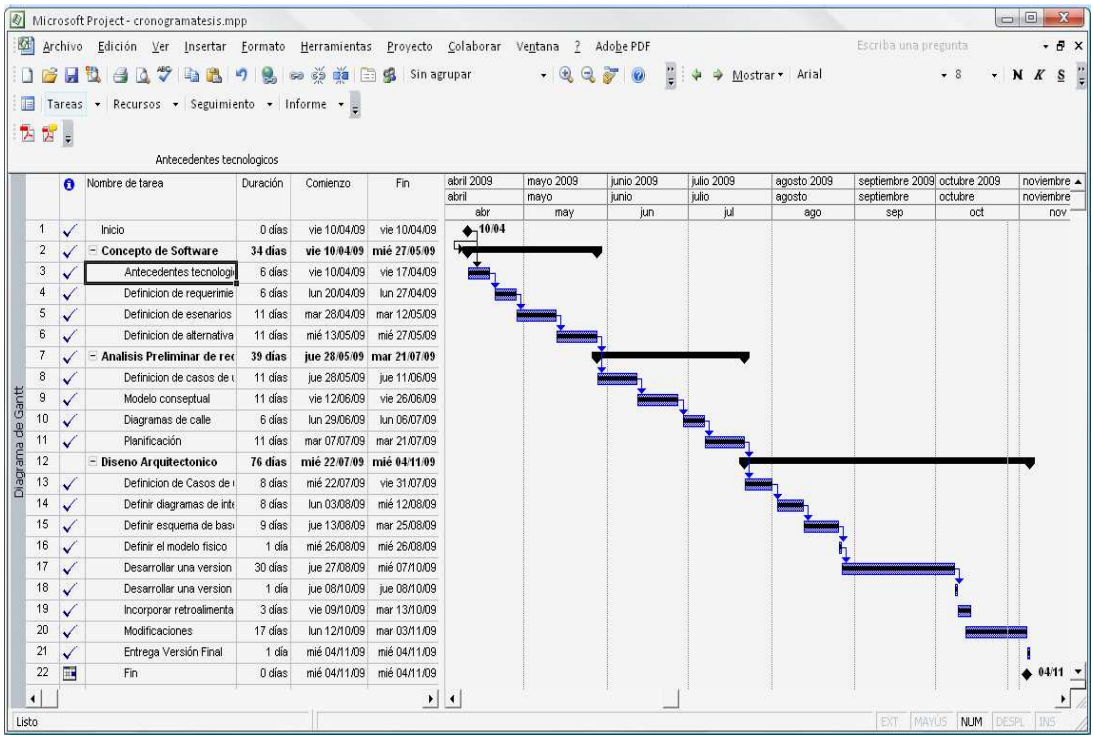

Figura IV.19 Cronograma de trabajo para la implementación de SystemLac

4.7.5. Requerimientos

Después de haber realizado la recopilación de la información a continuación ponemos a consideración del lector todos los requerimientos que se tomaron en cuenta para el desarrollo del sistema SystemLac.

Requerimientos Tecnológicos

Se refiere a aquellos requerimientos que el sistema necesita para su buen desempeño y funcionamiento.

RT1: La aplicación de escritorio desarrollada junto con su base de datos, deberá ser instalada sobre un equipo de cómputo que pertenezca a la planta de lácteos de la ESPOCH.

Requerimientos Funcionales

- **Administrador**

RF1: el sistema debe permitir al administrador, crear cuentas de usuario, junto con los datos adicionales del usuario para que el mismo pueda acceder al sistema.

Entradas: CI, nombres, apellidos, dirección, teléfono, fecha nacimiento, sexo, estado, usuario y password.

Procesos: el sistema guardará los datos del usuario en la Base de Datos dbKardex.

Salidas: mensaje de aceptación: Usuario registrado exitosamente.

RF2: el sistema debe permitir modificar los datos del usuario.

Entradas: CI, nombres, apellidos, dirección, teléfono, fecha nacimiento, sexo, estado, usuario y password.

Procesos: El sistema modificará los datos del usuario la BDatos.

Salidas: Mensaje de aceptación: Modificación exitosa.

RF3: El sistema debe permitir eliminar los datos del usuario que ya no tienen el respectivo permiso para el uso del sistema.

Entradas: Ninguna.

Procesos: El sistema mostrará los datos del usuario, y mostrará la opción de aceptar o cancelar el usuario, si selecciona Aceptar, el usuario será eliminado de la DB.

Salidas: Mensaje de confirmación: Usuario eliminado exitosamente.

- **Usuario**

RF4: El sistema debe permitir al usuario ingresar un nuevo proveedor.

Entradas: RUC, Nombre, Teléfono, Dirección, Representante, Teléfono, E-mail, Página Web.

Procesos: el sistema mostrará una pantalla para el ingreso de los datos del proveedor, una vez llenos los datos muestra el botón Guardar para guardar los datos en la DB o Cerrar para cancelar el ingreso.

Salidas: Mensaje de confirmación: Datos ingresados correctamente.

RF5: el sistema debe permitir visualizar la lista de proveedores y sus respectivos datos.

Entradas: Ninguna

Procesos: se muestra una pantalla con la lista de los proveedores existentes, con la opción de visualizar el detalle de sus datos.

Salidas: Ninguna

RF6: el sistema debe permitir modificar los datos de un determinado proveer.

Entradas: RUC, Nombre, Teléfono, Dirección, Representante, Teléfono, E-mail, Página Web.

Procesos: el sistema muestra los datos del proveedor para que estos puedan ser modificados, si se desea modificar, se debe dar clic sobre el botón Modificar, a continuación aparece un cuadro de dialogo, para confirmar la acción con las opciones Si, No o Cancelar.

Salidas: Mensaje de confirmación: los datos han sido actualizados correctamente.

RF7: el sistema permite insertar un nuevo insumo, para la elaboración de un nuevo producto o de uno ya existente que requiera un nuevo insumo.

Entradas: Código, Nombre, Medida, Stock Max, Stock Min, Descripción, Tipo.

Procesos: muestra una pantalla donde se debe ingresar los datos del insumo, especificando la unidad de medida del mismo, el stock máximo, mínimo, entre otros; presenta el botón Guardar para guardar en la DB y el botón Cancelar para cancelar la acción.

Salidas: Mensaje de confirmación: los datos han sido ingresados correctamente, Mensaje de error: Verifique que los datos estén ingresados correctamente, Mensaje de error: Ya existe el código o nombre de insumo.

RF8: el sistema permite listar todos los insumos, del insertar un nuevo insumo, para la elaboración de un nuevo producto o de uno ya existente que requiera un nuevo insumo.

Entradas: Código, Nombre, Medida, Stock Max, Stock Min, Descripción, Tipo.

Procesos: muestra una pantalla donde se debe ingresar los datos del insumo, especificando la unidad de medida del mismo, el stock máximo, mínimo, entre otros; presenta el botón Guardar para guardar en la DB y el botón Cancelar para cancelar la acción.

Salidas: Mensaje de confirmación: los datos han sido ingresados correctamente, Mensaje de error: Verifique que los datos estén ingresados correctamente, Mensaje de error: Ya existe el código o nombre de insumo.

RF9: el sistema permite modificar la especificación de la cantidad de insumo que se debe utilizar por cantidad de litros de leche.

Entradas: Insumo, cantidad de insumo, unidad de medida del insumo, cantidad de medida de la materia prima, unidad de medida de la materia prima.

Procesos: muestra una pantalla que permite escoger el insumo que se va a modificar, presenta la opción de cambiar la cantidad de insumo a utilizar y en qué cantidad de materia prima o leche cruda.

Salidas: Mensaje de confirmación: los datos han sido modificados correctamente, Mensaje de error: Verifique que los datos estén ingresados correctamente.

RF10: el sistema muestra la lista de insumos que se encuentran bajo el stock mínimo en inventario.

Entradas: Ninguna.

Procesos: muestra una pantalla con la lista de los insumos bajo stock mínimo, detallando el nombre del insumo, stock mínimo permitido, cantidad en bodega.

Salidas: Ninguno.

RF11: el sistema permite ingresar una nueva factura de compras.

Entradas: RUC, Código, Empresa (Proveedor), Nº Factura de Compra, Fecha, Código del insumo, Detalle, cantidad, precio unitario, precio total.

Procesos: aparece una pantalla para ingresar cada uno de los datos necesarios de la factura de compras, y el detalle de los insumos que pertenecen a esta factura de compras, seleccione Guardar para grabar los datos en la BD y cancelar si desea cancelar la operación.

Salidas: Mensaje de confirmación: los datos han sido ingresados correctamente, Mensaje de error: Verifique que los datos estén ingresados correctamente, Mensaje de error: Factura ya existe.

RF12: el sistema permite llevar el control de las entradas y salidas de insumos, así también como de los insumos dados de baja ya sea por perdida, derrame u otra situación de pérdida del mismo.

Entradas: código del insumo.

Procesos: al seleccionar el insumo del que quiero conocer los movimientos, se muestra fecha, detalle, cantidad entrada, precio unitario entrada, valor total entrada, cantidad salida, precio unitario de salida, valor total de salida, cantidad saldo, precio unitario saldos y valor total saldos. Muestra además la opción para exportar los datos a Excel.

Salidas: ninguno.

RF13: el sistema permite registrar el insumo que ha sido o deseo dar de baja, por rotura pérdida u otra razón.

Entradas: Fecha, insumo, stock, cantidad perdida, motivo por el que fue dado de baja.

Procesos: muestra la pantalla para ingreso de los datos del insumos a ser dado de baja, para que los datos se guarden en la DB selecciono Dar Baja, caso contrario para cancelar selecciono Cerrar.

Salidas: Mensaje de confirmación: Datos ingresados correctamente, mensaje de error: No se puede dar de Baja el Insumo dado, no hay esa cantidad en Stock, mensaje de error: Verifique que los Datos estén correctos.

RF14: el sistema permite listar los insumos que han sido dados de baja.

Entradas: Ninguna.

Procesos: muestra la lista de insumos dados de baja, detallando: producto, fecha , detalle, motivo, cantidad, precio, total.

Salidas: Ninguno.

RF15: el sistema permite ingresar un nuevo producto.

Entradas: código producto, nombre producto, código producto, código insumo, nombre insumo, cantidad, unidad de medida.

Procesos: se ingresa los datos del producto y los insumos que intervienen en su elaboración, para guardar en la DB, y si de desea cancelar pulsamos Cerrar.

Salidas: Mensaje de confirmación: Datos ingresados correctamente, mensaje de error, Mensaje de error: Verifique que estén ingresados correctamente los datos.

RF16: el sistema permite listar los productos existentes.

Entradas: Código del producto.

Procesos: muestra la lista de productos, detallando: nombre producto, cantidad existente, medida.

Salidas: Ninguno.

RF17: el sistema permite ingresar una orden de producción.

Entradas: código de producción, fecha elaboración, producto a elaborar, hora inicio, detalle, cantidad de productos.

Procesos: se ingresa los datos de la orden de producción y la lista de insumos necesarios para la elaboración de la cantidad de productos ingresada, aceptamos para guardar en la DB, y si de desea cancelar pulsamos Cerrar.

Salidas: Mensaje de confirmación: Datos ingresados correctamente, mensaje de error, Mensaje de error: Verifique que los datos estén correctos.

RF18: el sistema permite listar las órdenes de producción en proceso.

Entradas: ninguna.

Procesos: muestra la lista de órdenes de producción en proceso, detallando: nombre producto, detalle, fecha elaboración, cantidad, costo de producción, hora inicio, hora fin.

Salidas: Ninguno.

RF19: el sistema permite ingresar el fin de producción.

Entradas: hora de fin, producción real, fecha fin.

Procesos: una vez ingresados los datos de finalización de producción, estos son guardados en la DB y dicha producción pasa a formar parte de la lista de producciones terminadas, caso contrario selecciono cerrar.

Salidas: Mensaje de confirmación: Datos ingresados correctamente, mensaje de error, Mensaje de error: Verifique que los datos estén correctos.

RF20: el sistema permite listar las producciones realizadas.

Entradas: Ninguna.

Procesos: muestra la lista de producciones realizadas, detallando: nombre producto, detalle, fecha de elaboración, cantidad, costo de producción, hora inicio, hora fin.

Salidas: Ninguno.

RF21: el sistema permite llevar el control de las entradas y salidas de productos terminados, es decir los productos que ingresan de las nuevas producciones y los productos que han salido a la venta.

Entradas: código del producto, fecha inicio, fecha fin.

Procesos: seleccionar el producto del que quiero conocer los movimientos y el periodo, muestra fecha, detalle, cantidad producción, precio unitario producción, total producción, cantidad venta, precio unitario venta, total venta, cantidad bodega, precio unitario bodega y total bodega. Muestra además la opción para exportar los datos a Excel.

Salidas: ninguno.

RF22: el sistema permite ingresar los elementos involucrados en el costo de producción y gastos de operación, con el fin de determinar la utilidad neta.

Entradas: código del parámetro, nombre del parámetro, valor, descripción, tipo de parámetro.

Procesos: una vez ingresado los datos, presionamos guardar para guardar en la DB o cerrar para cancelar la operación**.**

Salidas: Mensaje de confirmación: Datos ingresados correctamente, mensaje de error, Mensaje de error: Verifique que los datos estén correctos.

RF23: el sistema permite ingresar un nuevo cliente.

Entradas: código, tipo.

Procesos: se ingresa el código y el tipo de clientes, para guardar en la DB presiono Guardar o Cerrar para cancelar la opción**.**

Salidas: Mensaje de confirmación: Datos ingresados correctamente, mensaje de error, Mensaje de error: Verifique que los datos estén correctos.

RF24: el sistema permite registrar la cantidad de ganancia dependiendo el tipo de cliente, tomando en cuenta el costo de producción.

Entradas: código precio, tipo cliente, producto, Precio de producción del producto, ganancia y precio de venta.

Procesos: se ingresa los datos indicados en las entradas, para guardar en la DB presiono Guardar o Cerrar para cancelar la opción**.**

Salidas: Mensaje de confirmación: Datos ingresados correctamente, mensaje de error, Mensaje de error: el cliente con su respectivo producto ya han sido ingresados.

RF25: el sistema permite ingresar una venta.

Entradas: código de venta, cliente, fecha, detalle, producto, stock, cantidad producto, precio, total.

Procesos: se ingresa los datos indicados en las entradas, para guardar en la DB presiono Insertar o Cerrar para cancelar la opción**.**

Salidas: Mensaje de confirmación: Datos ingresados correctamente, mensaje de error, Mensaje de error: No se puede vender productos que superen el stock o no seleccionó la cantidad a vender.

RF26: el sistema permite listar las ventas registradas en el sistema.

Entradas: cliente, producto, fecha inicio, fecha final.

Procesos: especificando el cliente, el producto y la fecha de inicio y fin, el sistema muestra la lista de ventas de dicho periodo.

Salidas: Ninguno.

RF27: el sistema permite generar un reporte de ventas.

Entradas: cliente, producto, fecha inicio, fecha final.

Procesos: muestra el reporte de las ventas realizadas en un periodo, especificando las ventas por cliente y las cantidades obtenidas de las ventas ordenadas por productos, además muestra la cantidad total de productos vendidos, costo total vendido en el periodo.

Salidas: Ninguno.

RF28: el sistema permite generar el balance general de un periodo determinado.

Entradas: fecha inicio, fecha final.

Procesos: dado la fecha de inicio y fin del periodo que deseo mostrar el balance general, el sistema permite exportar el reporte de ventas a Excel.

Salidas: Balance de ventas.

RF29: el sistema permite determinar la cantidad de fundas de leche en funda, quesos y yogurt que se deben elaborar para maximizar la utilidad de la planta, una vez construido el modelo matemático utilizando el método simplex.

Entradas: utilidad unitaria de l leche en funda, queso y yogurt; materia prima necesaria para producir una unidad de leche en funda, queso y yogurt; capacidad de producción utilizada actualmente de la leche, capacidad de producción máxima de la leche, capacidad de producción utilizada actualmente del queso, capacidad de producción máxima del queso, capacidad de producción utilizada actualmente del yogurt, capacidad de producción máxima del yogurt; tiempo de producción para un litro de leche, tiempo de producción para un queso, tiempo de producción para un litro de yogurt.

Procesos: el presente modulo consume los datos de los modulo de inventario y producción para determinar las entradas del modelo matemático y de esta forma generar la utilidad máxima.

Salidas: cantidad de fundas de leche, cantidad de quesos, cantidad de litros de yogurt mensual a producir y utilidad máxima mensual.

Requerimientos de Interfaces Externas

- **Interfaz de Usuario**

RIU: el sistema contará con una interfaz de usuario intuitiva, fácil de aprender y manejar, utilizando el mouse y el teclado y además el manejo por ventanas.

- **Interfaz Software**

RIS: para el desarrollo del sistema se usará el DBMS SQL Server 2005 para la base de datos y la plataforma Visual Studio 2008. Net con el lenguaje de programación C# para desarrollar el sistema.

Requerimientos No funcionales

Disponibilidad: El producto debe ser disponible únicamente para los usuarios que se encuentren registrados. El sistema cubre los requerimientos de disponibilidad

Confiabilidad: El producto debe ser confiable, para poder realizar auditorías.

Mantenibilidad: El sistema debe ser fácil de mantener, además se debe dejar documentación para realizar posibles actualizaciones.

Amigable: El sistema debe presentar interfaces amigables, intuitivas y de fácil interacción con el usuario.

Flexibilidad: El sistema debe ser flexible a los crecientes cambios de la planta de lácteos de la ESPOCH.

Seguridad: Se deben aplicar políticas de seguridad que garanticen la confidencialidad de la base de datos y de la aplicación.

4.8. Fase de Análisis

4.8.1. Introducción

El objetivo de esta fase es describir los casos de uso del sistema y los actores que intervienen en el, además la definición del modelo conceptual de la Base de datos y la construcción de diagramas lógicos que intervienen en el sistema.

4.8.2. Identificación de actores

Tabla IV.45 Identificación de actores.

Nº	Nombre	Perfil	Tipo de Acceso	Descripción
1	Usuario (Técnico, administrador o personal encargado de la planta de lácteos)	Empleado de la planta de lácteos.	Aplicación de escritorio.	Usuario que puede utilizar el sistema con su respectivo login y passsword.
2	Administrador	Persona encargada de administrar la Base de datos y del sistema.	Aplicación de escritorio.	Usuario encargado de las gestiones de configuración y administración de SystemLac.

4.8.3. Definición de Casos de Uso

Tabla IV.46 Casos de uso: Acceso al Sistema

CASO DE USO:	Acceso al sistema
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Administración
VISIÓN GENERAL:	Seguridad
CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). El sistema después de la pantalla de presentación, solicita el ingreso de la cuenta de acceso (Usuario, Clave)
2). En la pantalla de acceso el Cliente tiene que decidir si accede al sistema o sale del mismo. 3). Si el cliente accede al sistema ingresa su cuenta de acceso (Usuario, Clave). 4). Si el Cliente sale del Sistema.	
	5). El sistema valida los datos. 6). Si los datos fueron correctos. 7). Busca en la base de datos si existe la cuenta de acceso. 8). Si encuentra los datos permite el acceso al sistema, presentando la pantalla principal del mismo. 9). En el caso de que no encuentra los datos en la base de datos emite un error (*Usuario no registrado*).

11). En el caso de que el sistema le indique un error el cliente verifica el mensaje obtenido lo acepta y regresa al paso 2.	10). En caso de que los datos no fueron correctos emite un mensaje de error (*Verifique que los datos estén correctos*).
CURSOS ALTERNATIVOS	
LÍNEA 4 (Salir): Salida del sistema.	

Tabla IV.47 Casos de uso: Ingreso de Usuario y cuentas de acceso al sistema

CASO DE USO:	Ingreso de Usuario y cuentas de acceso al sistema
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Administración
VISIÓN GENERAL:	Seguridad
CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). Presenta la pantalla principal del sistema (SystemLact).
2). El cliente selecciona la opción de Cuentas y usuarios	
	3)El sistema solicita el ingreso de datos
4). El cliente toma la decisión de ingresa los datos del usuario y la cuenta de acceso o salir del ingreso de datos	
	5). El sistema valida los datos
	6). Si los datos fueron correctos.
	7). Busca en la base de datos si existe la cuenta de acceso.
	8). Si no encuentra una cuenta similar en la base de datos los datos permite el ingreso del usuario y cuenta de acceso al sistema.
	9). En el caso de que encuentra los datos en la base de datos emite un error (*Usuario ya registrado*).
	10). En caso de que los datos no fueron correctos emite un mensaje de error (*Verifique que los datos estén correctos*).

11). En el caso de que el sistema le indique un error, el cliente verifica el mensaje obtenido lo acepta y regresa al paso 4.	
CURSOS ALTERNATIVOS	
LÍNEA 4 (Salir): Salida de la pantalla de ingreso de usuario y cuentas de acceso al sistema.	

Tabla IV.48 Casos de uso: Ingreso de Proveedor

CASO DE USO:	Ingreso de Proveedor
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Proveedores
VISIÓN GENERAL:	Proveedores de insumos para producción
CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). Presenta la pantalla principal del sistema (SystemLact).
2). El cliente selecciona la opción de Ingreso de Proveedor	
	3)El sistema solicita el ingreso de datos
4). El cliente toma la decisión de ingresa los datos del proveedor o salir del ingreso de proveedores	
	5). El sistema valida los datos
	6). Si los datos fueron correctos.
	7). Busca en la base de datos si existe el

	Proveedor. 8). Si no encuentra un proveedor similar en la base de datos, permite el ingreso del proveedor. 9). En el caso de que encuentra los datos en la base de datos emite un error (Proveedor *ya registrado*). 10). En caso de que los datos no fueron correctos emite un mensaje de error (*Verifique que los datos estén correctos*).
11). En el caso de que el sistema le indique un error, el cliente verifica el mensaje obtenido lo acepta y regresa al paso 4.	
CURSOS ALTERNATIVOS	
LÍNEA 4 (Salir): Salida de la pantalla de ingreso de usuario y cuentas de acceso al sistema.	

Tabla IV.49 Casos de uso: Datos del Proveedor

CASO DE USO:	Datos del Proveedor
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Proveedores
VISIÓN GENERAL:	Lista de Proveedores, Datos del proveedor y modificación de los mismos
CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). Presenta la pantalla principal del sistema (SystemLact).
2). El cliente selecciona la opción de Proveedores	

	3)El sistema presenta la lista de proveedores
4). El cliente selecciona un proveedor y verifica el detalles de sus datos	
	5). El sistema retorna los datos del proveedor
6). En el caso de que el cliente salga de la pantalla	
	7). Regresa al sistema
CURSOS ALTERNATIVOS	
LÍNEA 6 (Salir): Salida de la pantalla.	

Tabla IV.50 Casos de uso: Ingreso de Insumos y listo de Insumos

CASO DE USO:	Ingreso de Insumos y listo de Insumos
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Insumos
VISIÓN GENERAL:	Ingreso de insumos al sistema y visión de los mismos
CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). Presenta la pantalla principal del sistema (SystemLact).
2). El cliente selecciona la opción de Ingreso de Insumos	
	3)El sistema solicita el ingreso de datos
4). El cliente toma la decisión de ingresa los datos del insumo salir del ingreso de datos	

	5). El sistema valida los datos 6). Si los datos fueron correctos permite el ingreso del insumo. 7). Regresa a la pantalla principal 8). En caso de que los datos no fueron correctos emite un mensaje de error (*Verifique que los datos estén correctos*).
9). En el caso de que el sistema le indique un error, el cliente verifica el mensaje obtenido lo acepta y regresa al paso 4. 10). Luego del ingreso para su comprobación del ingreso de datos el cliente accede a la lista de insumos	
	11). El sistema Presenta los insumos ingresados
12). El cliente cierra la pantalla.	
CURSOS ALTERNATIVOS	
LÍNEA 4,12 (Salir): Salida de la pantalla de ingreso de usuario y cuentas de acceso al sistema.	

Tabla IV.51 Casos de uso: Control mínimo de insumos

CASO DE USO:	Control mínimo de insumos
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Insumos
VISIÓN GENERAL:	Control de insumos que han llegado al límite permitido para generar una producción.

CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). Presenta la pantalla principal del sistema (SystemLact)
2). El cliente selecciona la opción control mínimo de insumos	
	3). El sistema verifica y busca los insumos que se encuentran con el stock mínimo permitido
4). El cliente sale de la pantalla.	
CURSOS ALTERNATIVOS	
LÍNEA 4 (Salir): Salida de la pantalla.	

Tabla IV.52 Casos de uso: Control de Insumos

CASO DE USO:	Control de Insumos y reporte de compras
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Insumos
VISIÓN GENERAL:	Permite verificar los movimientos de los insumos en la adquisición y producción de los mismos.
CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). Presenta la pantalla principal del sistema (SystemLact).
2). El cliente selecciona la opción control de insumos.	
	3) El sistema presenta el kardex de los insumos con los datos del primero que está en lista.

4). El cliente selecciona otro Insumo	5) El sistema realiza la búsqueda de la información solicitada y presenta sus datos.
6). El cliente selecciona un periodo	7) El sistema presenta en Excel el reporte de las compras realizadas en el mes
8) Sale de la pantalla	
CURSOS ALTERNATIVOS	
LÍNEA 8 (Salir): Salida de la pantalla de ingreso de usuario y cuentas de acceso al sistema.	

Tabla IV.53 Casos de uso: Ingreso de factura de compras

CASO DE USO:	Ingreso de factura de compras
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Facturas
VISIÓN GENERAL:	Permite el ingreso de una factura de un proveedor existente e insumos de los mismos
CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). Presenta la pantalla principal del sistema (SystemLact).
2). El cliente selecciona la opción ingreso de factura.	
	3) El sistema el formato de factura y solicita que seleccione el proveedor.

4). El cliente selecciona el proveedor y selecciona nueva factura 5) Ingresa los insumos y las cantidades a registrar	
	6) El sistema valida los datos del insumo si no se presenta ningún error en la validación ingresa a la base de datos las cantidades del insumo. 7) en caso de emitirse un error por validación de datos presenta un mensaje (*Verifique datos en el insumo o seleccione el insumo*)
8). En el caso de que el sistema le indique un error, el cliente verifica el mensaje obtenido lo acepta y regresa al paso 5 9) sale de la pantalla	
CURSOS ALTERNATIVOS	
LÍNEA 9 (Salir): Salida de la pantalla de ingreso de usuario y cuentas de acceso al sistema.	

Tabla IV.54 Casos de uso: Lista de facturas

CASO DE USO:	Lista de facturas
ACTORES:	Cliente
TIPO:	Primario esencial
PROPÓSITO:	Facturas
VISIÓN GENERAL:	Permite visualizar todas las facturas que han sido registradas, debido a las compras realizadas de los insumos
CURSO TÍPICO DE EVENTOS	
ACTOR	**SISTEMA**
	1). Presenta la pantalla principal del sistema

	(SystemLact).
2). El cliente selecciona la opción Lista Facturas.	
	3) El sistema presenta todas las facturas ingresadas en el sistema.
4) Si el cliente selecciona Lista facturas generales.	
	5) El sistema presenta los todas las facturas
6) El cliente puede seleccionar buscar una factura, para lo cual ingresa el numero de factura a buscar	
	7) El sistema realiza la búsqueda de la factura en la base de datos.
	8) En el caso de no encontrar la factura presenta el formulario vacio caso presenta el detalle de la compra realizada con esa factura.
9) Sale de la pantalla	
CURSOS ALTERNATIVOS	
LÍNEA 10 (Salir): Salida de la pantalla de ingreso de usuario y cuentas de acceso al sistema.	

4.8.4. Diagramas de Casos de Uso

Caso de Uso: Ingreso al sistema

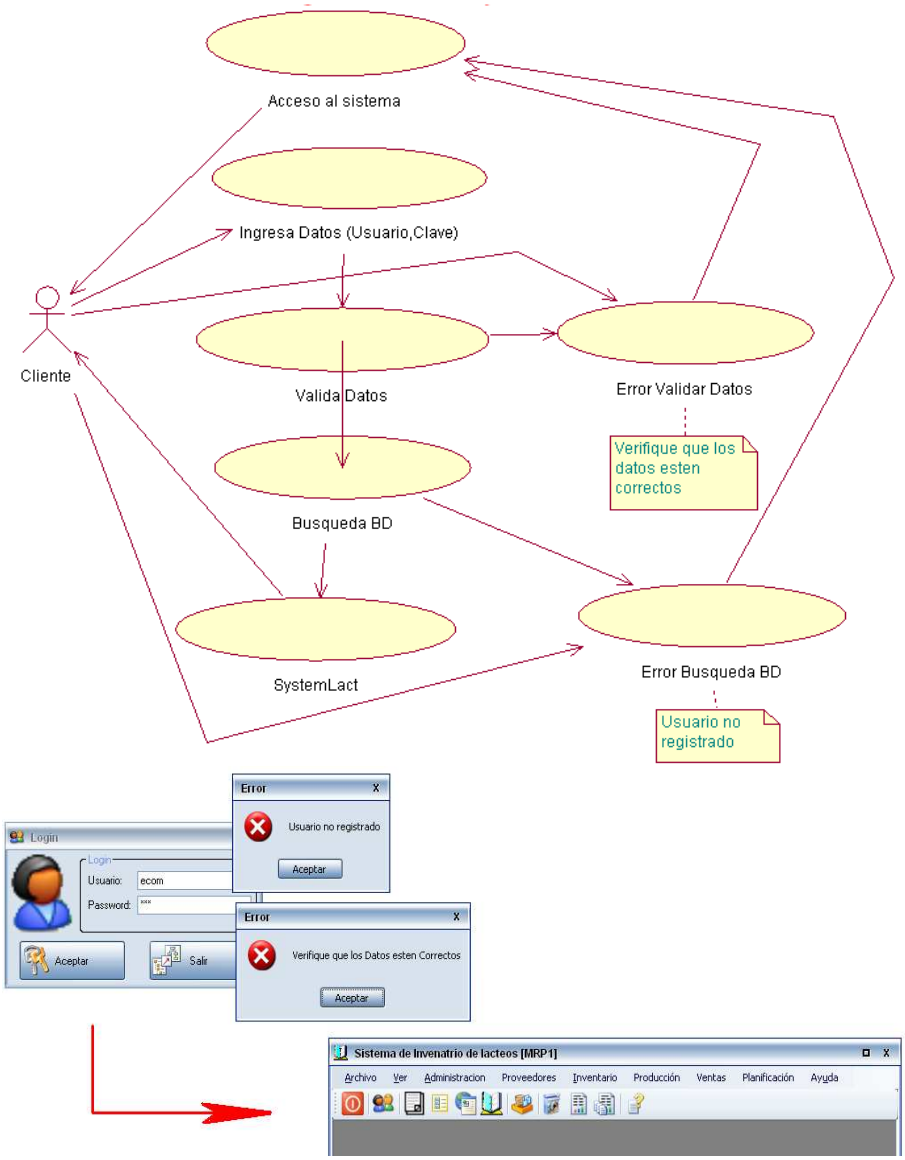

Figura IV.20 Diagrama de Caso de Uso. Ingreso al sistema SystemLac

Caso de Uso: Ingreso de usuario y cuentas de acceso al sistema

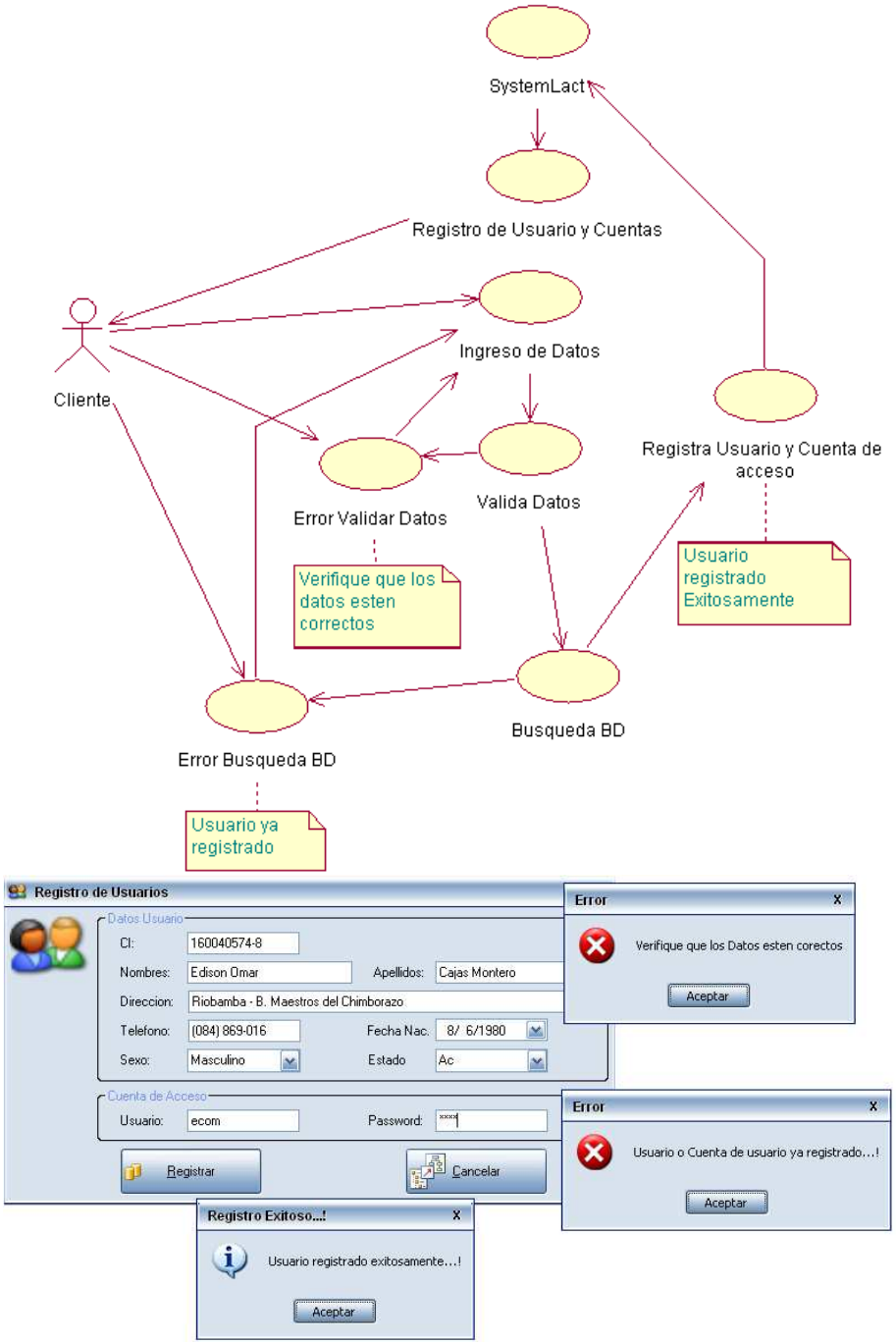

Figura IV.21 Diagrama de Caso de Uso. Ingreso de usuario y cuentas de acceso al sistema SystemLac

Caso de Uso: Ingreso de Proveedor

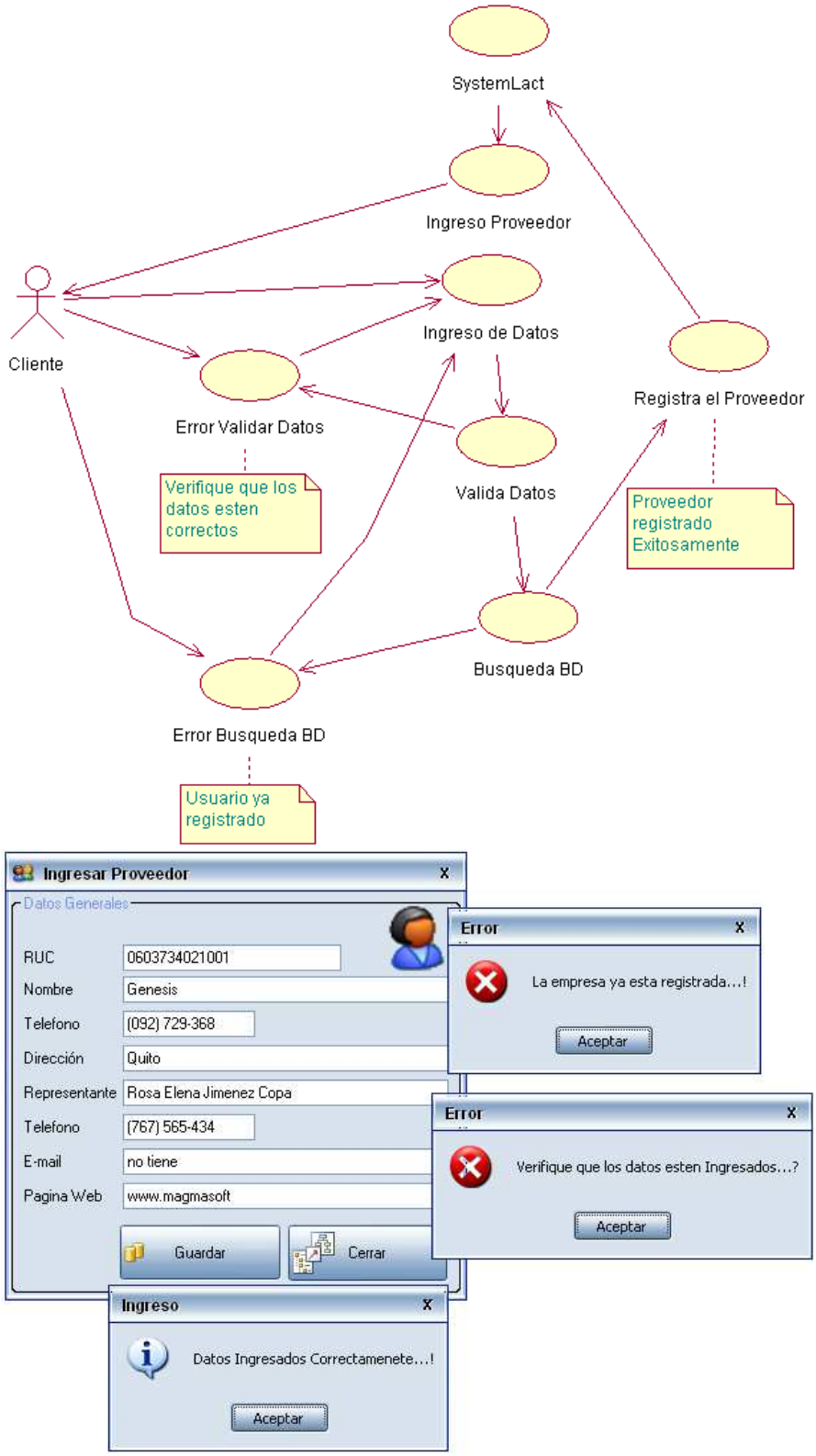

Figura IV.22 Diagrama de Caso de Uso. Ingreso de Proveedor

Caso de Uso: Lista de proveedores

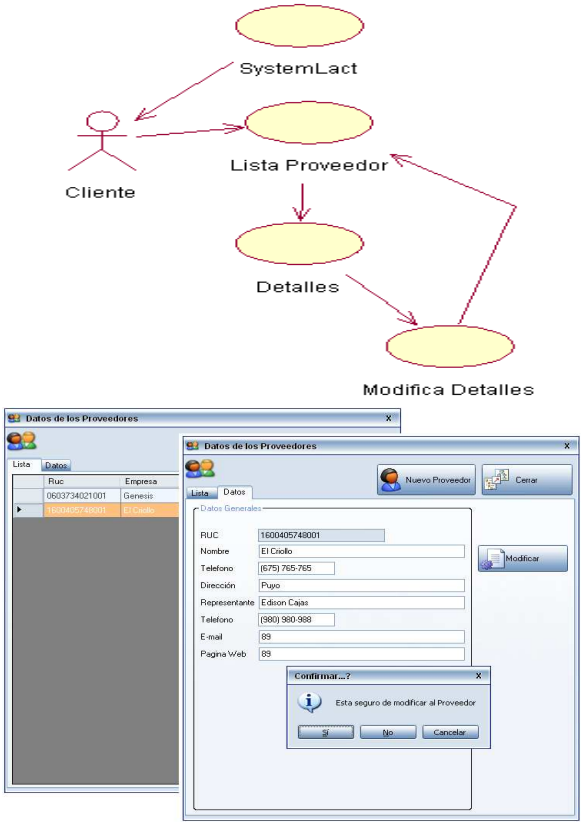

Figura IV.23 Diagrama de Caso de Uso. Lista de Proveedores

Caso de Uso: Lista de insumos bajo el stock mínimo

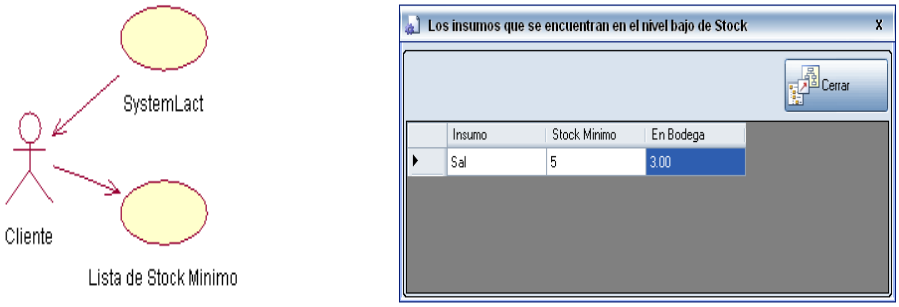

Figura IV. 24 Diagrama de Caso de Uso. Lista de insumos bajo el stock mínimo

Caso de Uso: Ingreso y lista de insumos

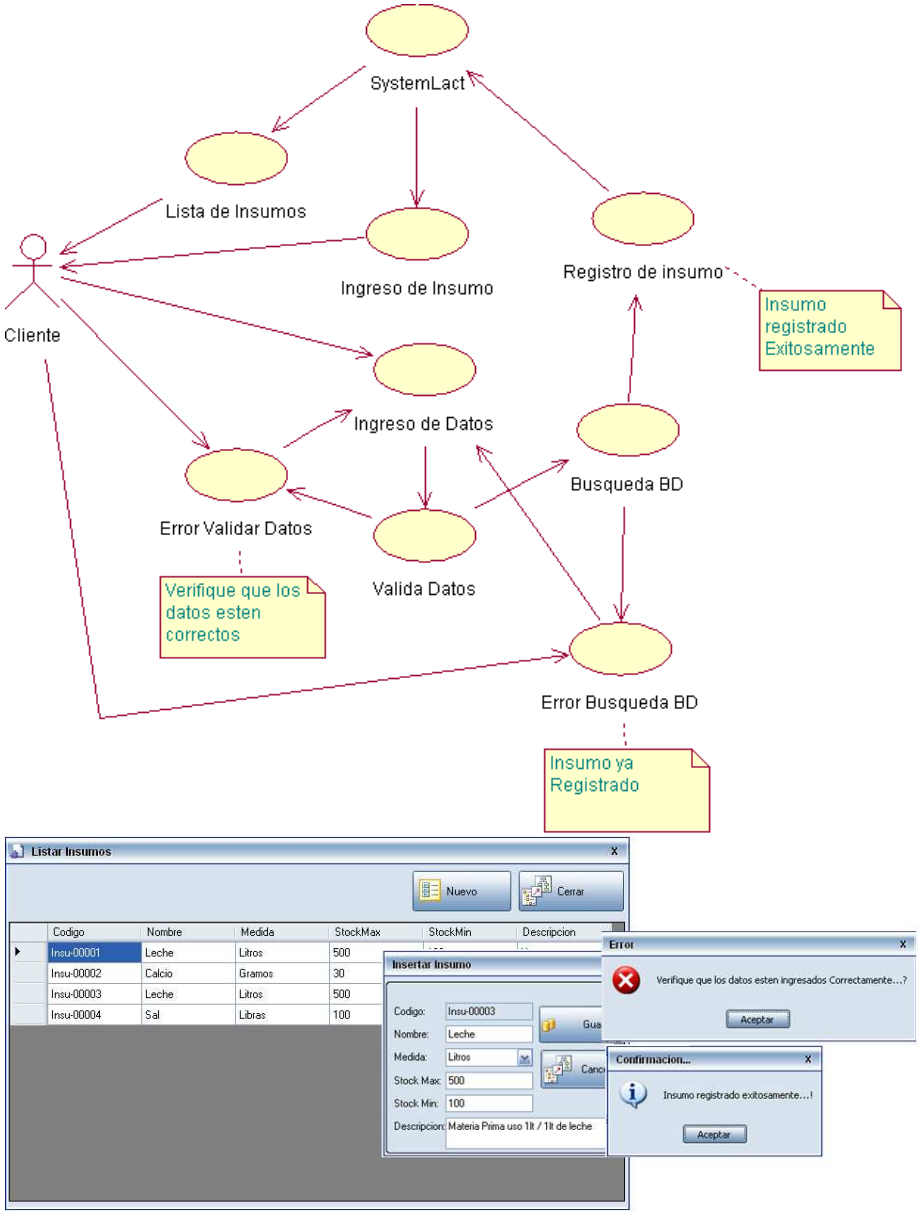

Figura IV.25 Diagrama de Caso de Uso. Ingreso y lista de Insumo

Caso de Uso: Control de Insumos

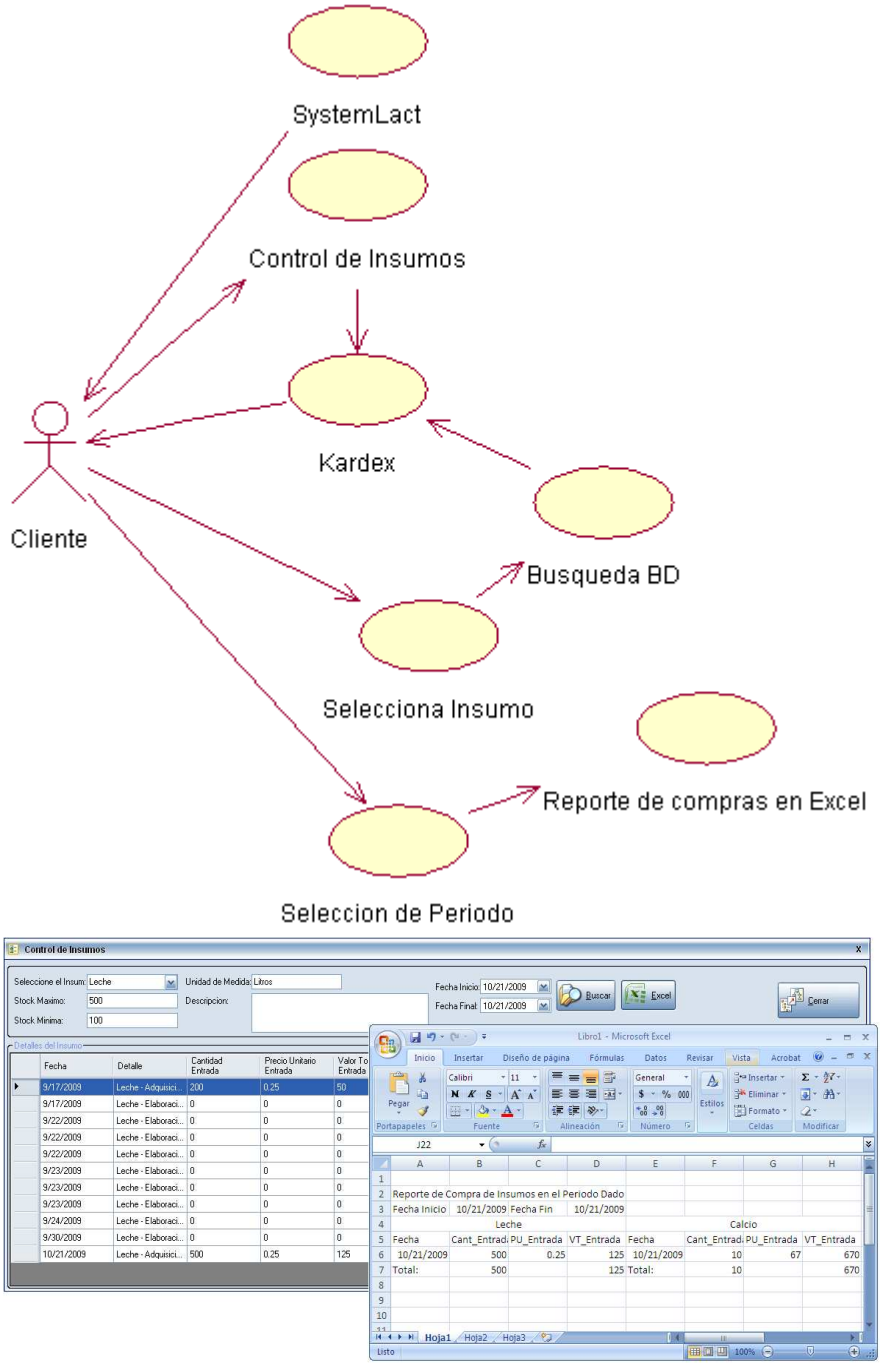

Figura IV.26 Diagrama de Caso de Uso. Control de Insumos

Caso de Uso: Ingreso factura de compras

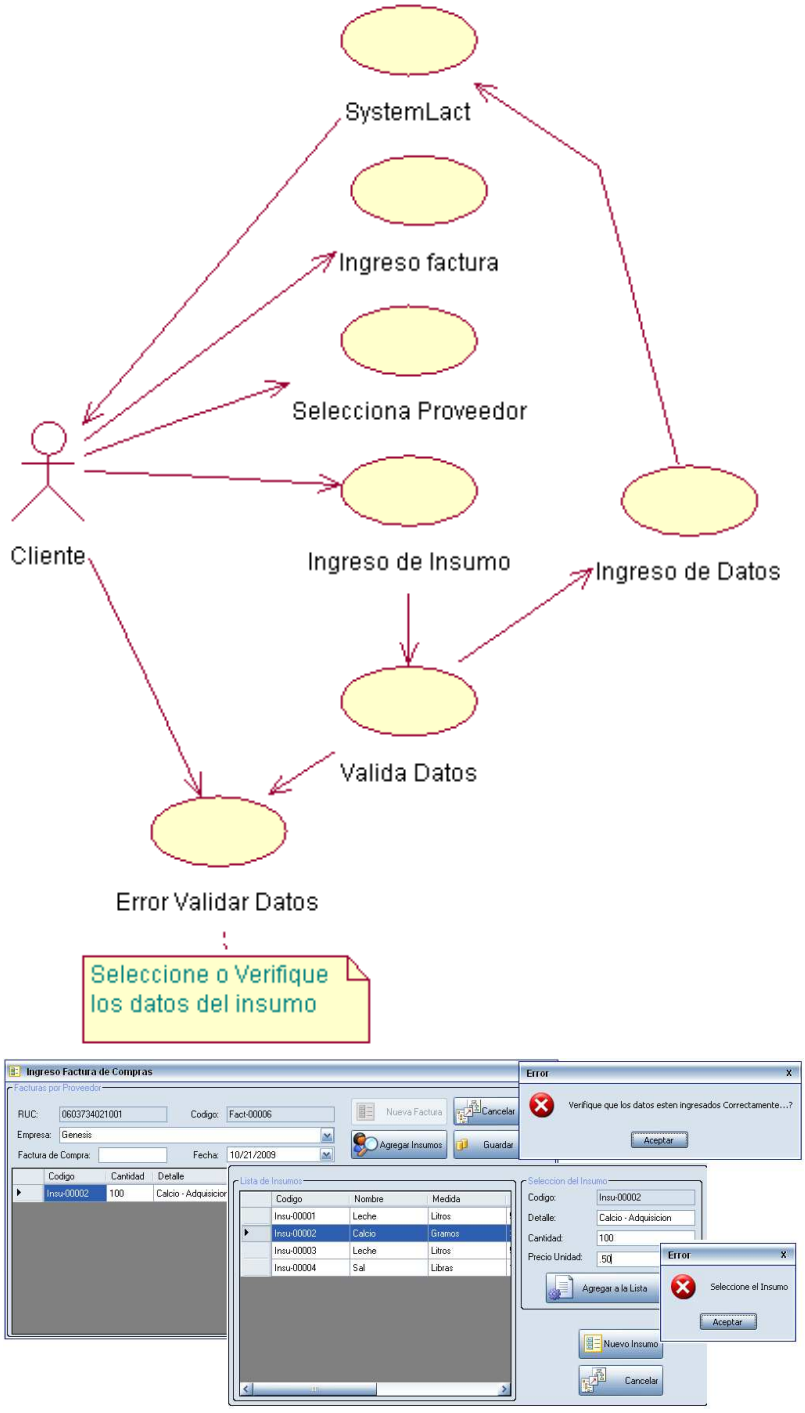

Figura IV.27 Diagrama de Caso de Uso. Ingreso factura de compras

Caso de Uso: Lista de facturas de compras

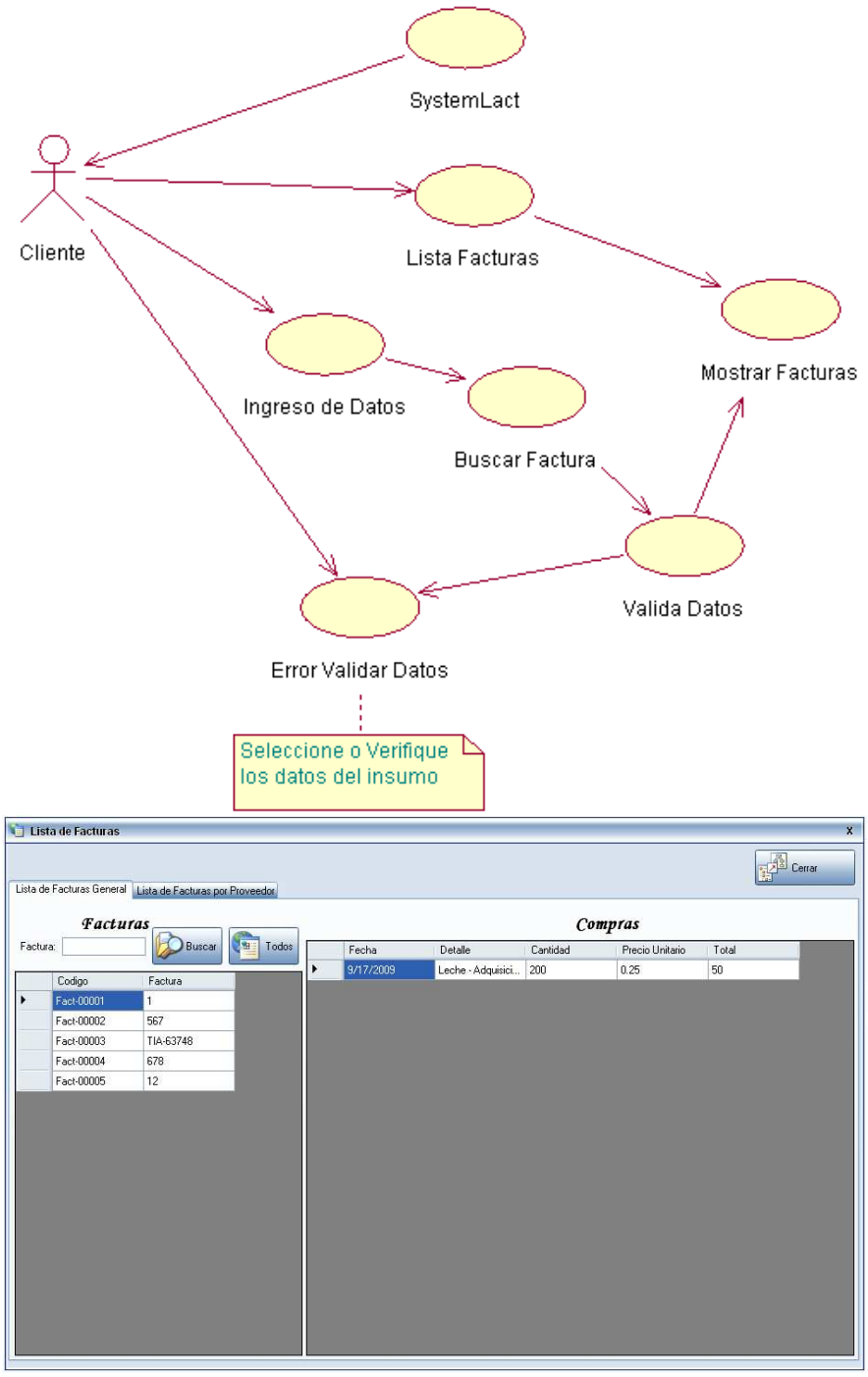

Figura IV.28 Diagrama de Caso de Uso. Lista de facturas de compras

4.8.5. Construcción del Modelo Conceptual

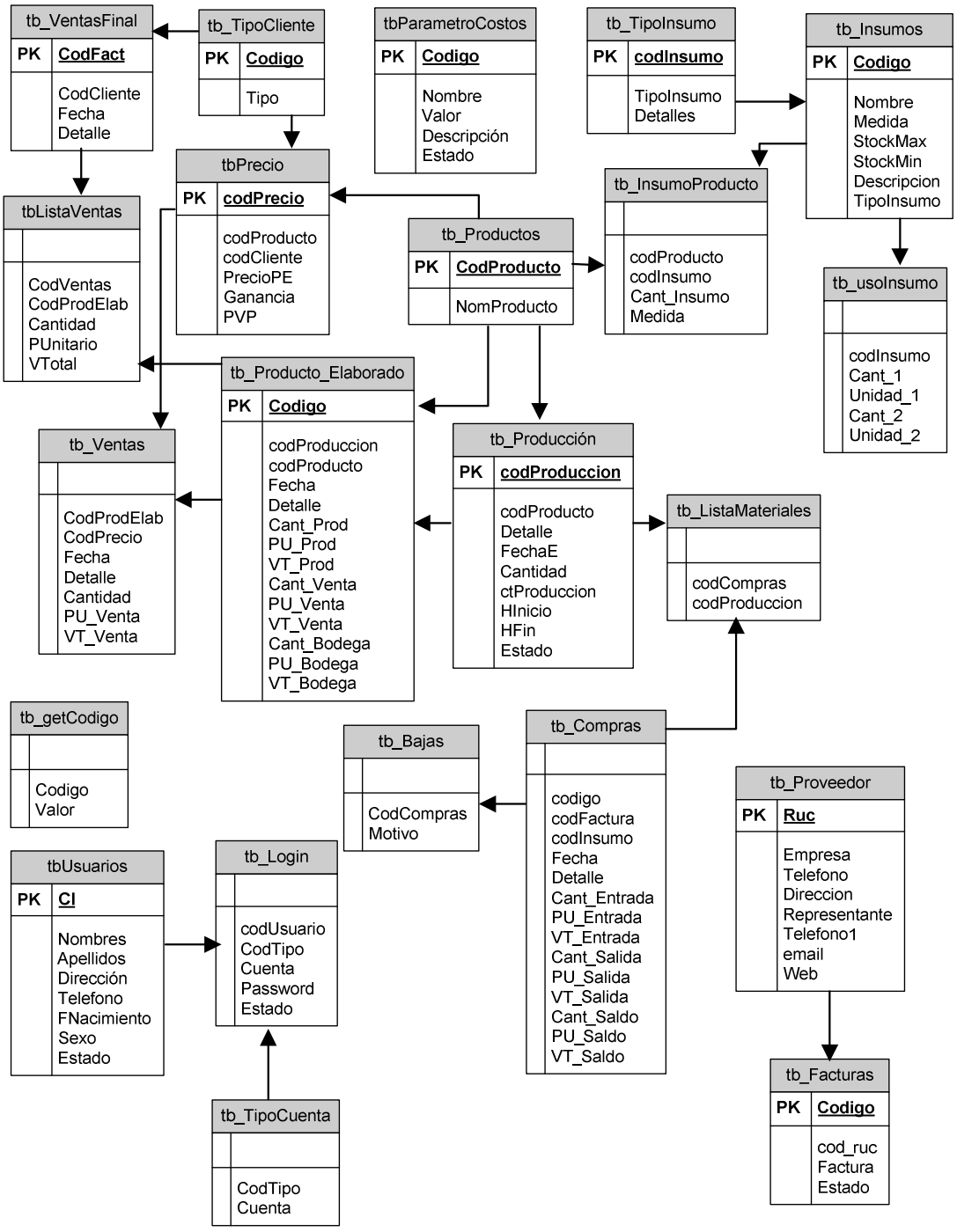

Figura IV.29 Diagrama conceptual de la Base de Datos dbKardex de SystemLac

4.8.6. Construcción del Modelo Lógico

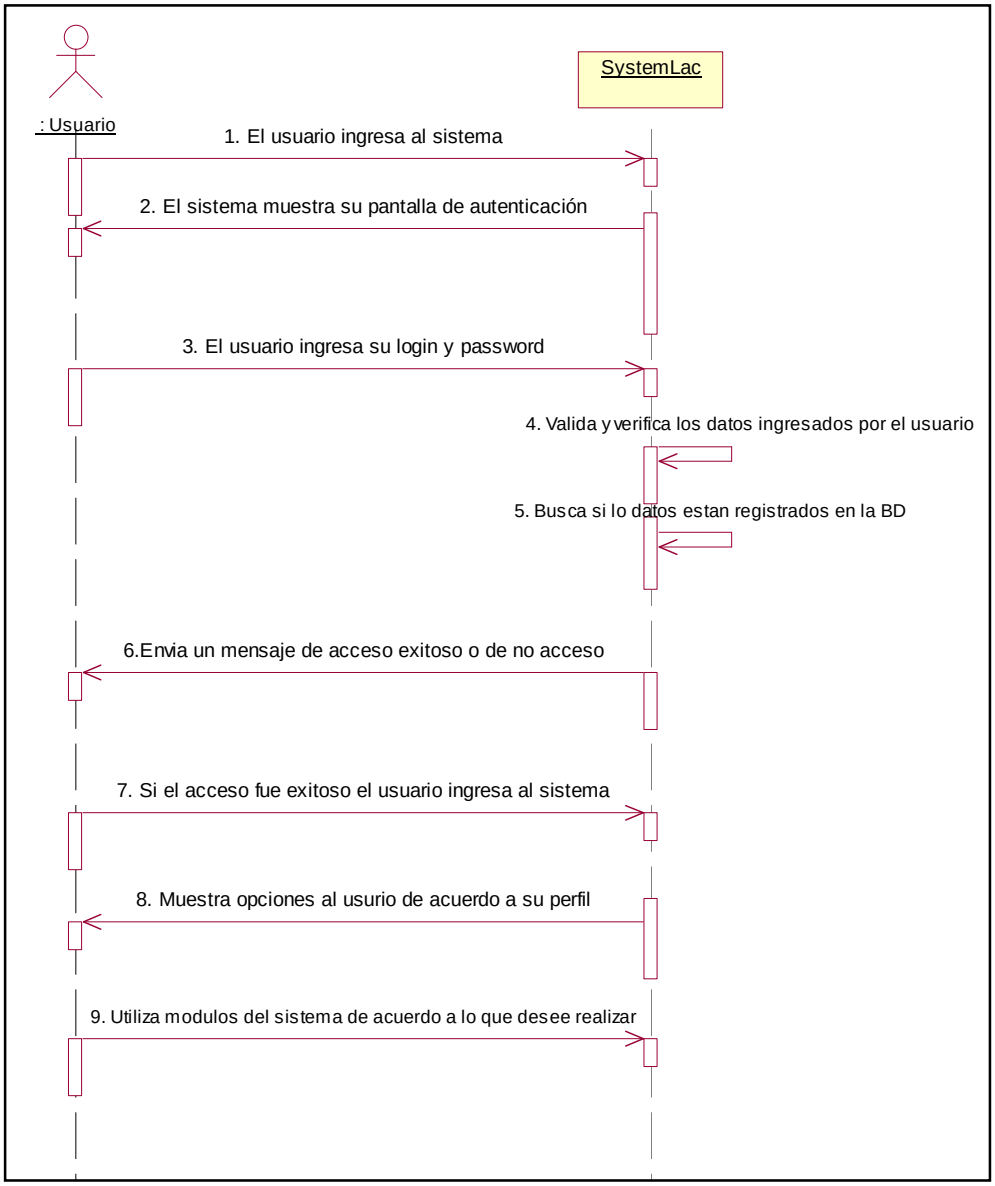

Figura IV.30 Diagrama de Secuencia de SystemLac

4.9. Fase de Diseño

4.9.1. Introducción

Tiene el objetivo de definir la arquitectura del sistema, los módulos y unidades lógicas que la conforman, los componentes de negocio y el diagrama de despliegue que describe la distribución de los componentes y módulos con los que interactúa el sistema.

4.9.2. Arquitectura del sistema

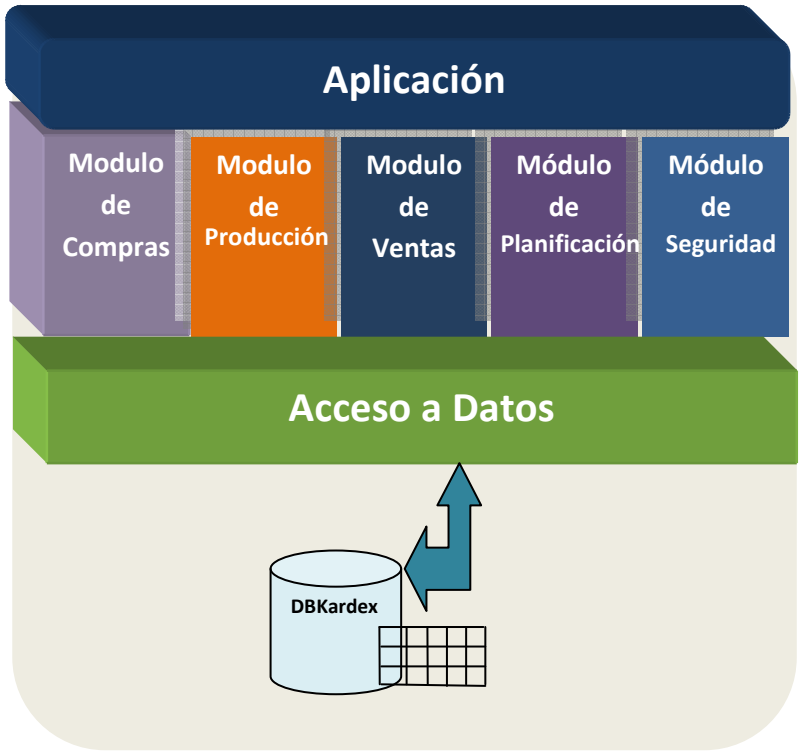

Figura IV.31 Arquitectura del sistema SystemLac

4.9.3 Diagrama físico de la Base de Datos

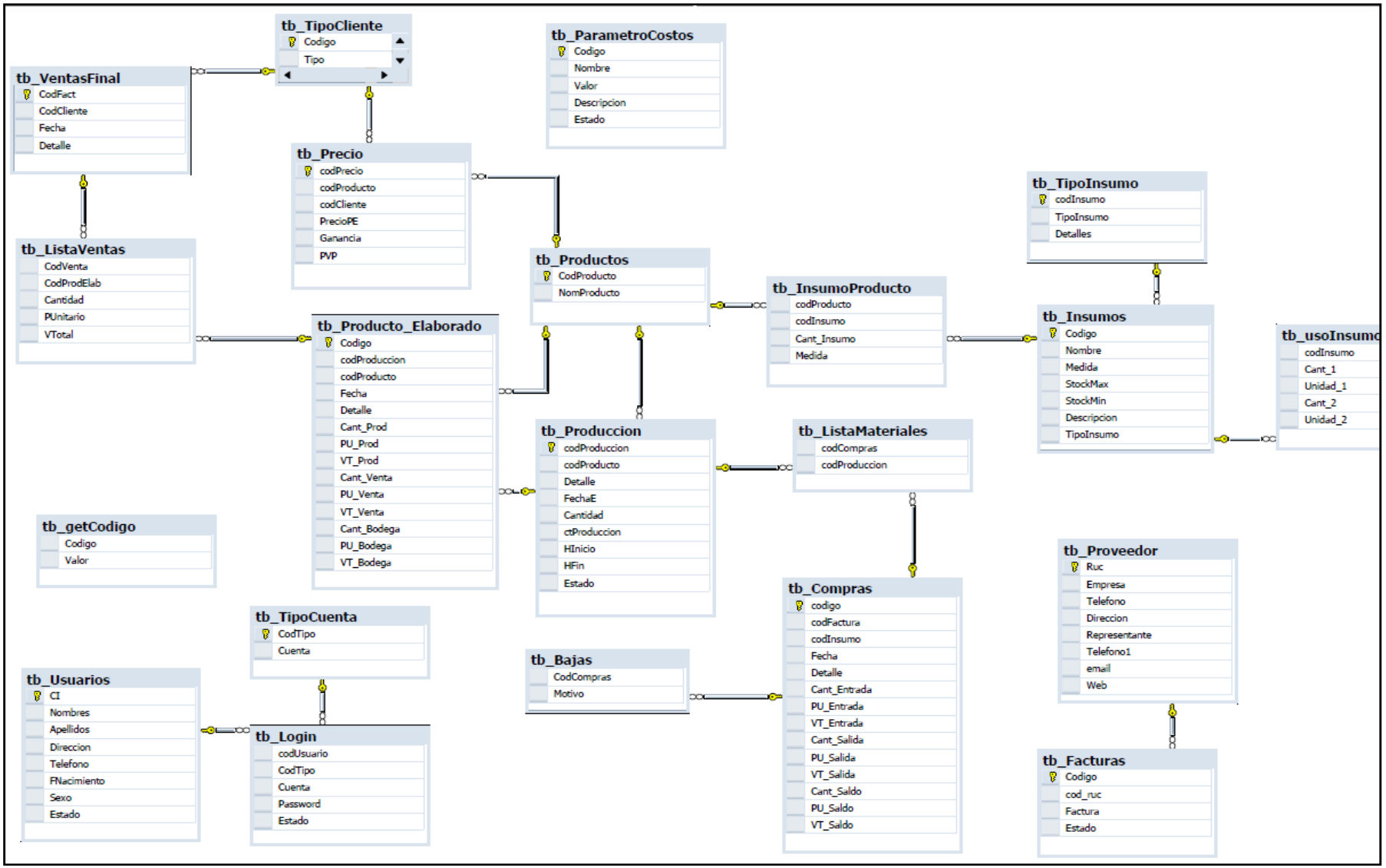

Figura IV.32 Diagrama de la Base de Datos de SystemLac

4.9.4. Diagrama de Componentes

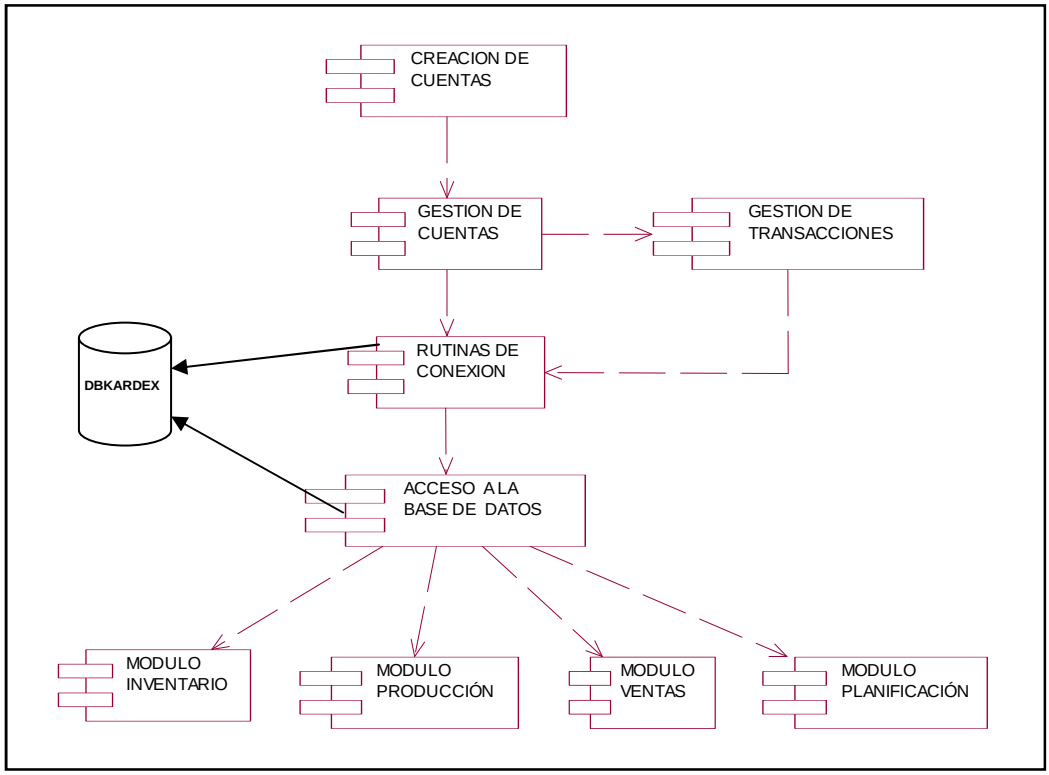

Figura IV.33 Diagrama de Componentes del sistema SystemLac

4.9.5. Diagrama de Despliegue

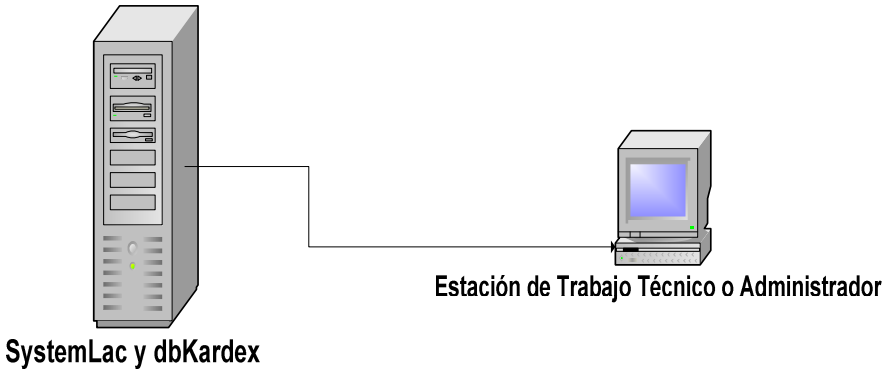

Figura IV.34 Diagrama de despliegue del sistema SystemLac

4.10. Fase de Implementación

4.10.1. Introducción

Esta fase involucra una serie de versiones internas del producto, desarrollados por partes para medir su progreso y para asegurarse que todos sus módulos o partes están sincronizados y pueden integrarse, además del desarrollo del código, el equipo implementa la solución.

Durante la fase de desarrollo, se crea la solución. Este proceso incluye la creación de código que implementa la solución y la documentación del mismo. Además de elaborar el código, también se desarrolla la infraestructura para la solución.

4.10.2. Organización de los elementos de implementación de SystemLac en Visual Studio.Net

La solución tiene como nombre SystemLac y su estructura en Visual Studio.Net podemos apreciar en la figura IV.35, utilizando C# para su implementación.

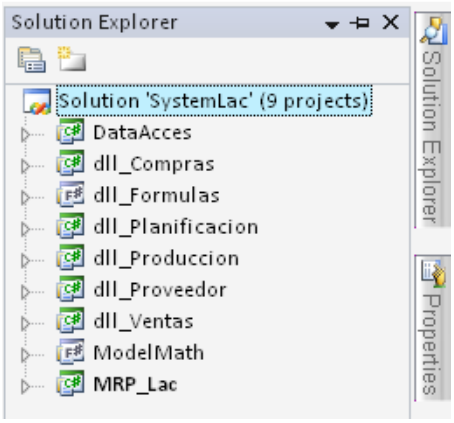

Figura IV.35 Organización de los elementos de implementación de SystemLac

La solución SystemLac, se encuentra distribuido en 9 proyectos:

DataAcces: es la capa de acceso a datos, está compuesta de los dataset dsPaSQL.xsd, dsSeguridad.xsd, dsTablas.xsd, los mismos que contienen las consultas necesarias para extraer los datos desde la Base de Datos hacia la aplicación.

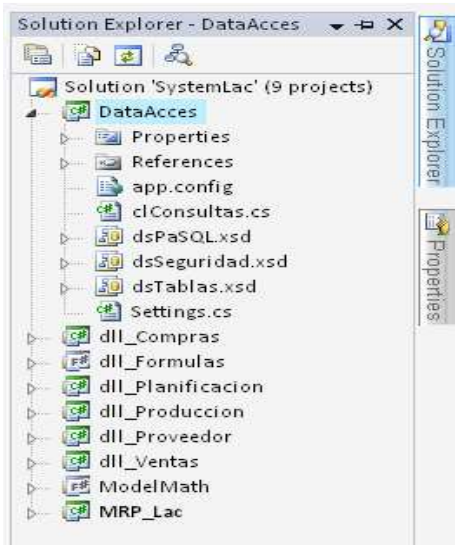

Figura IV.36 Organización y estructura de Data Acces

dll_Compras: corresponde al modulo de Compras, contiene las pantallas necesarias para satisfacer los requerimientos con respecto a ingreso de nuevos insumos, control de insumos, bajas de insumos, factura de compras, lista de insumos dados de baja, lista de facturas, lista de insumos, insumos que se encuentran con un nivel de stock y la cantidad de insumo que se necesitan para elaborar determinada cantidad de productos, con la materia prima que se tiene disponible.

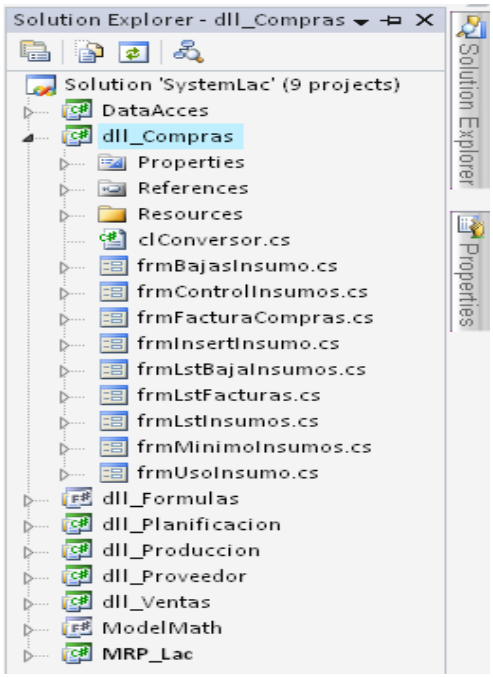

Figura IV.37 Organización y estructura de dll_Compras

dll_Planificacion: contiene la clase clSimplex.cs, en la que está definido todas las funciones para resolver el modelo matemático, y el formulario con la respectiva interfaz para determinar las cantidades de productos que se debe elaborar para alcanzar la máxima utilidad, los datos de entrada de las ecuaciones son extraídos del resto de módulos que conforman el sistema SystemLac.

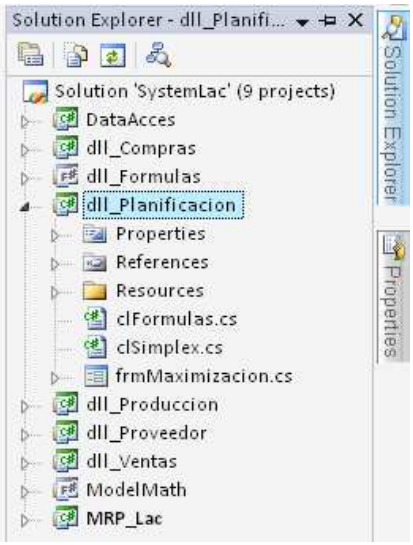

Figura IV.38 Organización y estructura de dll_Planificación

dll_Produccion: se refiere al modulo donde se encuentra todo los detalles referente a producción, como: Ingreso de los detalles de un producto, ingreso de los parámetros necesarios para determinar los costos, ingreso de una orden de producción, detalles de las cantidades necesarias para elaborar lista de productos elaborados, ingreso de los parámetros para calcular el costo de producción, fin de una producción, lista de productos elaborados, lista de productos en producción.

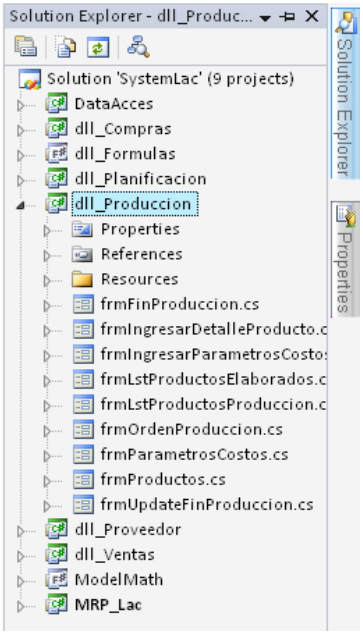

Figura IV.39 Organización y estructura de dll_Producción

dll_Proveedor: corresponde al módulo del Proveedor, que contiene las pantallas para ingreso de proveedor y visualización de la lista de proveedores.

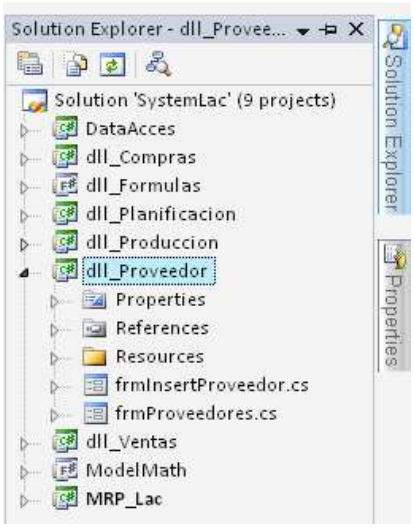

Figura IV.40 Organización y estructura de dll_Proveedor

dll_Ventas: este modulo contiene todas las tareas referentes a Ventas, estas son las siguientes:

- Ingreso de clientes.
- Ingreso de una venta.

- Ingreso del precio según el cliente en base al costo de producción y el tipo de producto.
- Lista de las ventas realizadas en un periodo.
- Reporte de ventas dado un periodo.

Figura IV.41 Organización y estructura de dll_Ventas

MRP_Lac: contiene el programa principal de la solución y el modulo de administración, el mismo que permite el acceso al sistema a través de cuentas de usuario, para de esta manera mantener la seguridad de la información.

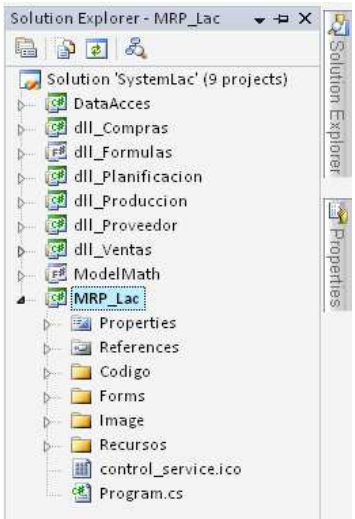

Figura IV.42 Organización y estructura de MRP_Lac

CONCLUSIONES

- Con el estudio de las características de los paradigmas de programación se determinaron las bondades que ofrecen cada uno de ellos y cuál ofrece con mayor eficacia la potencialidad necesaria para la construcción del modelo matemático planteado.

- Si bien es cierto que el paradigma ganador resulto ser el Orientado a Objetos con el L.P. C#, este lenguaje de programación es muy poderoso, que ofrece muchas ventajas al momento de implementar modelos matemáticos, pero cabe recalcar lo que le hace especial es el dominio que la mayoría de los programadores de la plataforma .net, tienen sobre él, permitiendo de esta manera que C# sea aprovechado al máximo. Y se pueda optimizar los recursos computacionales que este consume al momento de su ejecución.

- Con el modelo matemático planteado y el uso de datos históricos se obtuvieron resultados aproximados para generar la cantidad necesaria de productos a elaborar y la máxima utilidad que generaría la empresa ante esa producción.

- Aunque el modelo matemático resuelve con gran detalle la maximización de las utilidades, se puede indicar que no es el único modelo ya que existen otros que podrían hacerlo de la misma manera , además este modelo es una propuesta que se adapta al modelo de producción que actualmente maneja la planta de lácteos ESPOCH.

- Es necesario indicar que el modelo matemático desarrollado para la planta, presenta sus debilidades dado a que las utilidades que conforman la función objetivo son calculadas en base a datos históricos, las mismas que no pueden ser recalculadas a medida que se va variando la cantidad de producción.

- La construcción de un modelo matemático, implica un estudio minucioso de las características del lugar donde se va a modelar, en donde se debe tratar de que los resultados que arroja el modelo discrepen en lo mínimo con los datos del mundo real, dado que de esto depende la validez del modelo.

- La implementación de la aplicación System-Lact ayudara al técnico a determinar las cantidades necesarias de productos a elaborar, para generar la mejor utilidad en la planta de lácteos, además determinar la cantidad necesaria de insumos que se necesita para realizar un producto determinado.

RECOMENDACIONES

- Uno de los aspectos que se debe tomar en cuenta a la hora de escoger un paradigma de Programación, es fijarnos a que plataforma va enfocada, el nivel de conocimiento, la optimización de los recursos computacionales y el ámbito de la implementación del sistema.

- Es conveniente mejorar la línea de producción de la planta, para reducir al máximo la intervención humana y de este modo incrementar el número de unidades fabricadas

- Para el buen funcionamiento de la aplicación System-lact los datos a ser ingresados deben ser correctamente estudiados, con las unidades exactas a utilizar en la producción y su equivalencia por cantidad de producto requerido, caso contrario la aplicación emitirá errores en la producción como también incoherencias en los costos de producción y fabricación.

- Para un análisis de nuevas restricciones, sería conveniente hacer un estudio de la demanda del mercado, el cual nos permite conocer cuáles son las preferencias de los clientes.

- Para aprovechar al máximo la capacidad de la maquinaria con la que actualmente cuenta la planta de lacteos de la ESPOCH, sería conveniente incrementar la recepción de leche cruda.

- Es recomendable realizar una campaña de marketing para promover el consumo de los productos que ofrece la ESPOCH, para incrementar el tamaño de los clientes potenciales y los ingresos que generan las unidades de producción de la misma sean mayores.

RESUMEN

El estudio trata de las características de los paradigmas (modelo o patrón) de programación: estructurada, orientada a objetos y funcional, para determinar, utilizando las normas del estándar ISO 9126, cual optimiza recursos computacionales, para la implementación del modelo de planificación de producción que permita, a la Planta de Lácteos de la ESPOCH, la maximización de utilidades en sus procesos productivos.

Los parámetros utilizados en la evaluación de los paradigmas fueron: líneas de código, tiempo de ejecución, memoria utilizada y tiempo de procesador, adicionalmente se implementó el modelo matemático lineal con el método simplex en los lenguajes de programación pascal, C# y F#, se considero como restricciones del modelo: materia prima, capacidad de producción máxima, tiempo de producción de los productos elaborados, con lo cual se obtuvieron los siguientes resultados: El paradigma Orientado a Objetos 78 puntos, el paradigma Funcional 54 puntos y el Paradigma Estructurado 46 puntos, siendo el paradigma Orientado a Objetos el que ofrece muchas más ventajas al momento de optimizar los recursos computacionales.

Analizados los datos históricos recolectados se determinó que la planta actualmente no es rentable y que para maximizar su utilidad idealmente se debería elaborar la cantidad de productos que arroja el modelo propuesto con lo cual se obtendría un incremento del 429% en las utilidades

La aplicación desarrollada utilizando el paradigma ganador ayudará a mejorar la administración de insumos, producción y ventas, siendo una herramienta útil para la toma de decisiones.

SUMMARY

This study deals whit the characteristics of the structured, oriented to objects and functional programming paradigms (model or pattern) using the norms of the standard ISO 9126 to determine which one optimizes the computing resources for the implementation of the production model planning permitting maximization of profits in its productive processes to the Dairy Product Plant of the ESPOCH. The parameters used in the paradigm evaluation were: code lines, execution time, used memory and processor time. Additionally the lineal mathematical model was implemented with the simplex method in the pascal programming languages, C# and F#. The following were considered to be the model restrictions: raw material, maximum production capacity, production time of the elaborated products resulting in the following results: the paradigm oriented to objects, 78 points, the functional paradigm, 54 points and the structured paradigm, 46 points the paradigm oriented to objects being the one which offers many advantages at the moment of optimizing the computing resources. After having analyzed the collected historical data it was determined that the plant at the moment is nor profitable and that to optimize its use, ideally, the product quantity of the proposed model must be elaborated to obtain 429% increase in profits. The developed application using the winner paradigm will help improve the input management, production and sales, being a useful tool for decision making.

BIBLIOGRAFÍA

LIBROS

1. BIRD, RICHARD. Introducción a la Programación Funcional con Haskell. Madrid: Prentice Hall, 2000.

2. GARCÍA BERMEJO, J. Programación Estructurada En C. Madrid: Prentice Hall, 2008

3. GARCÍA MARTÍN, J. Técnicas de Planificación Agregada. España: McGraw-Hill, 2006.

4. JACOBSON, I., BOOCH, G. y RUMBAUGH, J. El Proceso Unificado de Desarrollo de Software. México: Pearson Addisson Wesley, 1999.

5. MEYER, B. Construcción de software orientado a objetos, 2. ed. Madrid: Prentice Hall, 1999.

6. PRESSMAN, R. S. Ingeniería del Software un Enfoque Práctico. Mexico: McGraw Hill, 1998.

7. RUMBAUGH, J., JACOBSON, I. y BOOCH, G. El Lenguaje Unificado de Modelado: España: Pearson Addisson Wesley, 2006.

8. VALLS FERRAN, J., CAMACHO, D. Programación Estructurada y Algoritmos En Pascal. Madrid: Pearson Educación, 2004.

Bibliografía Internet

9. BELLAS PERMUY, F. Departamento de Tecnologías de la Información y las Comunicaciones (TIC) Universidad de A Coruña. Introducción a la Orientación a Objetos.
   http://www.tic.udc.es
   2009-05-19

10. COROMOTO. La POO frente a la Programación Tradicional.
    http://nereida.deioc.ull.es/~cleon/doctorado/doc01/lang/cpp.html.
    2009-5-30

11. FRAGA, G. Departamento de Computación Cinvestav: Evolución de la Programación Orientada a Objetos. 7 de diciembre de 2006.
    http://www.cs.cinvestav.mx.
    2009-05-30

12. GAMMA, E., HELM, R. JOHNSON, R. Design Patterns: Elements of Reusable Object Oriented Software. Addisson Wesley, 1994.
    http://www.tic.udc.es/~fbellas.
    2009-05-30

13. GONZÁLEZ, J. Descripción del nuevo lenguaje de Microsoft C#, vinculado a la plataforma .NET.
    http://www.josanguapo.com/.
    2009-05-20

14. MICROSOFT.COM. Use técnicas de programación funcional en .NET Framework.

http://microsoft.com/spain/athome

2009-06-1

15. SALVADOR SÁNCHEZ A. Programación Avanzada: Introducción a la Programación Funcional.

http://www.daneprairie.com.

2009-05-21

16. UNIVERSIDAD DE HUELVA. Departamento de Electrónica, Sistemas Informáticos y Automática. Programación Declarativa.

http://www.universidaddehuelva.

2009-05-18