"NEWT, UNA HERRAMIENTA DE PROGRAMACION GRAFICA PARA LA ENSEÑANZA DEL PENSAMIENTO ALGORITMICO"

T E S I S

MAESTRIA EN CIENCIAS
ESPECIALIDAD CIENCIAS COMPUTACIONALES

INSTITUTO TECNOLOGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

POR

RAUL VALENTE RAMIREZ VELARDE

JULIO DE 1991

"Newt, una herramienta de programación gráfica para la enseñanza del pensamiento algorítmico"

TESIS

Maestría en Ciencias
Especialidad en Ciencias Computacionales

Instituto Tecnológico y de Estudios Superiores de Monterrey

Por

Raúl Valente Ramírez Velarde

Julio de 1991

"Newt, una herramienta de programación gráfica para la enseñanza del pensamiento algorítmico"

Por

Raúl Valente Ramírez Velarde

TESIS

Presentada a la División de Graduados e Investigación
Este Trabajo es Requisito Parcial
para Obtener el Título de
Maestro en Ciencias

Instituto Tecnológico y de Estudios Superiores de Monterrey

Julio de 1991

Dedicatoria

A mis padres, hermanos y amigos. A Cyndi.

Reconocimientos

Deseo expresar mi más sincero agradecimiento a todas las personas que de una forma u otra me ayudaron a llevar a feliz término esta tesis. Ellos son:

Mis asesores Ing. Norma Frida Roffe, Dr. Jorge A. Olvera e Ing. Santiago Rodríguez.

Mis amigos Martín González, Rafaél Cárdenas y Enrique García Moreno.

PhD. Bruce Sherwood de la Universidad de Carnegie Mellon, PhD. Judy Brown de la Universidad de Iowa, Keith A. Hall de la Universidad Estatal de Ohio, y PhD. Steve Cunningham de la Universidad Estatal de California en Stanislaus.

Y muy especialmente Cyntia de León.

Resumen

"Newt, una herramienta de programación gráfica para la enseñanza del pensamiento algorítmico"

Esta tesis plantea los problemas a los que se enfrentan los alumnos de cursos de lenguajes básicos a nivel universitario.

Se encontró que primero se tiene que partir de inculcar en los alumnos el pensamiento algórítmico para la solución de problemas y plantea la utilización de un programa educativo para ayudar a los alumnos de computación a desarrollar este tipo de pensamiento.

Para implementar este programa se realizó una investigación a fondo sobre los elementos que deben estar presentes en los programas educativos para que éstos puedan lograr las metas para los que fueron diseñados.

La investigación se realizó en términos del diseño de Instrucción Ayudada por Computadora y el diseño de Interfases Interactivas Gráficas.

Se expone una metodología de desarrollo de programas educativos consistente en tres fases: Análisis, Desarrollo y Evaluación.

Finalmente se diseñó e implementó el programa que ayuda a los alumnos a utilizar el pensamiento algorítmico independiente de cualquier lenguaje de programación por medio de diagramas de flujo.

Tabla de Contenido

CAPITULO I

INTRODUCCION

Es indudable que en el marco competitivo que se vive en la actualidad, se requiere que el profesionista tenga una adecuada formación que le permita utilizar los adelantos que ofrece la tecnología y poder encarar la competencia en el ejercicio de su profesión. Y dado que son las universidades las encargadas de proporcionar la mayor parte de dicha formación, son ellas las que tienen la mayor responsabilidad ante la sociedad de tomar la creación y asimilación de tecnología como un reto, y afrontarlo. Un ejemplo muy claro de la forma en que las universidades están respondiendo lo encontramos en la enseñanza de computación, específicamente la programación. Para los alumnos de cualquier carrera profesional, por ejemplo en las ingenierías, el saber programar por lo menos en un lenguaje proporciona una herramienta indispensable para la aplicación de las diversas técnicas de análisis de sistemas, y diseño de equipo. Es por esto que todas las ingenierías contienen en sus primeros semestres materias curriculares de programación básica. Sin embargo, muchos de los estudiantes que llegan a estos cursos no han tenido experiencia previa con las computadoras. Esto puede ser, para el alumno, una pastilla difícil de pasar.

El principal obstáculo que encuentran los estudiantes de programación es el énfasis que generalmente se da a la teoría, dejándose la práctica como una responsabilidad para el alumno. La resolución de problemas siempre es precedida por un proceso largo de instrucción formal en el salón donde al alumno se le enseña la sintaxis del lenguaje, la estructura de los programas y los comandos del compilador. Así, cuando llega el momento de poner a prueba sus conocimientos, el programador novato tiene que enfrentar tres problemas al mismo tiempo: encontrar una solución al ejercicio, dominar la sintaxis del lenguaje y utilizar eficientemente el compilador, intérprete o ambiente de programación. La idea principal de esta tesis es crear un medio de programación en el cual el alumno se pueda concentrar en la resolución de problemas, y quede protegido de las distracciones que resultan de utilizar un lenguaje con una sintaxis compleja y de operar un ambiente de programación o un compilador con muchos comandos y procedimientos.

Ciertamente, el quedar protegido del medio no es suficiente para encontrar soluciones a los diferentes problemas que debe afrontar el alumno. Es necesario hacer que el estudiante desarrolle las habilidades mentales adecuadas además de que adquiera los conocimientos. Esto no es exclusivo de la programación, por ejemplo en Matemáticas, el saber Algebra no garantiza que se pueda despejar una variable en alguna ecuación. En computación, una de las herramientas que se ha utilizado para el desarrollo de dichas habilidades es el diagrama de flujo. El **diagrama de flujo** es un tipo de

esquematización que permite visualizar las diferentes acciones que se están realizando y el flujo de la ejecución entre estas acciones. En este tipo diagrama se basará esta tesis para apoyar la capacidad de resolución de problemas en los alumnos.

Los diagramas de flujo se han estado utilizando insistentemente en los cursos introductorios de computación desde hace mucho tiempo. Sin embargo, su efectividad como auxiliares del pensamiento ha estado mermada porque generalmente no es posible utilizarlos extensivamente en la resolución de problemas reales. Es decir, que un alumno puede llegar a comprender la utilización de los diagramas de flujo pero nunca podrá estar seguro de sus diseños porque la única forma de verificación es por medio del maestro. Fuera de esto, la segunda forma de verificación es la implementación directa en un programa, pero generalmente el alumno estará demasiado ocupado luchando con el compilador o intérprete como para preocuparse por hacer un diagrama de flujo. Esto conduce a retomar la idea del ambiente de programación que ya se ha mencionado. El primer punto que trata esta tesis es pues, la utilización de diagramas de flujo en la enseñanza de leguajes de programación.

La utilización de computadores como auxiliares en la educación no es una idea nueva. Se ha intentado utilizar computadoras para la educación desde sus inicios en los años cuarentas. Y ahora en la época moderna, hay una enorme cantidad de proyectos en todo el mundo que lo siguen intentando, muchos con muy poco éxito. ¿Cuáles deben ser las características debe poseer un programa educativo para que éste pueda ser exitoso? Esto es el segundo punto que se discute en esta tesis. El tercer punto consiste en el diseño e implementación de un programa que ayude en la adquisición del pensamiento algorítmico por medio de un ambiente integrado y protegido de programación.

Este sistema computacional, el cual se introduce con el nombre de **Newt**, es un ambiente de programación en el cual el usuario está protegido de la sintaxis del lenguaje porque la forma de programar es gráfica y no escrita. El alumno crea un diagrama de flujo independiente de cualquier lenguage y podrá ejecutarlo para probar su diseño. Al mismo tiempo, el diagrama de flujo va generando un listado del programa equivalente al diagrama de flujo en algún lenguaje de alto nivel que permitirá al alumno crear una relación más directa entre un diagrama de flujo y la sintaxis de un lenguaje computacional. Además, se protege al alumno de las dificultades de saber manejar un compilador, intérprete o ambiente de programación utilizando la interfase gráfica de Newt, que hace que la manipulación del éste sea lo más intuitiva posible. Los comandos más útiles se presentan en forma de logotipos o íconos, tomando ventaja de la visualización y el resto de los comandos están a la disposición del usuario por medio de menús, ademas de muchas otra características, como interacción por medio de ratón, ventanas, animación, etc.

Entonces, esta tesis está realmente enfocando dos aspectos del mismo problema. El primero es el la creación de aplicaciones educacionales efectivas para el aprendizaje. Para esto se discutirá la problemática humana, computacional y pedagógica que debe

afrontar el diseñador de software educativo, llegando a una técnica que aproveche lo mejor posible las ventajas de la computación sin perder aquellas inherentes a la presencia del maestro. El segundo aspecto, es la utilización de estos programas en cursos de introducctorios de programación. Para explorar este tema se construyó el programa que ya se ha mencionado, Newt.

A grosso modo, la tesis está organizada de la siguiente manera:

El capítulo II contiene una panorámica general de los objetivos de esta tesis, la metodología que se utilizó para llevarla a cabo y los recursos que se necesitaron.

El capítulo III toma el tema de los diagramas de flujo, sus características y su utilidad como auxiliares en la enseñanza de lenguajes de programación.

El capítulo IV discute la justificación del software educativo como herramienta en la enseñanza. Se da también un marco histórico. Se mencionan diferentes aplicaciones que han dejado huella en la breve historia del software educativo y los elementos que deben caracterizar un buen ambiente de diseño de programas educativos.

El capítulo V habla sobre el diseño de la Instrucción Ayudada por Computadora que es la portadora de la enseñanza dentro de los programas educativos. Se discuten las teorías existentes acerca del aprendizaje y se deriva una metodología de diseño de instrucción basada en computadoras.

Los capítulos VI y VII contienen los principios que deben guiar el diseño de interfases gráficas e interactivas y los diversos elementos que componen una interfase gráfica. Las herramientas que componen dichas interfases y que deben estar presentes en la pantalla son menús, logotipos, botones y ventanas. Se discute la utilización del ratón y otros dispositivos de señalamiento y la utilización del color. Además se analizan técnicas para la codificación de la información en forma visual y la evaluación de la interfase.

El capítulo VIII hace una descripción funcional de Newt. Explica la utilización de la interfase para el desarrollo de diagramas de flujo detalla el Lenguaje Estructurado Para Principiantes Pico LEPP y su relación con los diagramas de flujo.

El capítulo IX es una descripción técnica de Newt donde se discute porqué se utilizó el lenguaje CT para su desarrollo y se explican algunas técnicas como animación y la evaluación de las instrucciones dadas por el usuario y las estructuras con las que se maneja el lenguaje Pico-Pascal.

Finalmente el capítulo X presenta una discusión de los resultados obtenidos y un resumen de las aportaciones hechas por la investigación.

CAPITULO II

EL PROBLEMA

2.1 Propósitos del Estudio

1. Hacer una profunda investigación sobre la teoría del desarrollo de programas educativos.

2. Diseñar e implementar un programa educativo que ayude a los alumnos de lenguajes de programación a lograr un dominio sobre el pensamiento algorítmico.

2.2 Definición del Problema

La problemática que se pretende resolver es la falta de estructuras y mecanismos mentales por parte de los alumnos de computación básica para poder resolver problemas computacionales y asimilar la tecnología. Se intenta también, aligerar el caudal de conocimientos que el alumno debe dominar al mismo tiempo para poder desarrollar programas que resuelvan correctamente los problemas que se plantean en clase.

2.3 Elementos del Problema

2.3.1 Los alumnos deben aprender el pensamiento algorítmico antes de poder diseñar programas en lenguajes computacionales, los diagramas de flujo son una de las mejores formas que se conocen para lograr ésto. Sin embargo, la única forma de evaluar un diagrama de flujo es através del maestro. Esto y el poco tiempo que generalmente se le puede dedicar a su diseño inhiben la asimilación del pensamiento algorítmico a través de la utilización de estos diagramas. Se desea incultar en los estudiantes la utilización y experimentación con los diagramas de flujo.

2.3.2 Otra forma de aprender el pensamiento algorítmico es a través de la implementación directa de programas. Para poder implementar programas que puedan resolver los problemas que se presentan a los alumnos en clase, estos deben aprender no sólo en pensamiento algorítmico, sino la sintaxis de un lenguaje, la estructura de un programa y los comandos del editor y del compilador. Estos conocimeintos son demasiados y evitan el correcto dominio del pensamiento algorítmico.

2.4 Investigación sobre el problema

Para poder desarrollar un programa que pudiera ser una herramienta efectiva para el desarrollo del pensamiento algorítmico se investigó en artículos, libros relacionados con el tema y se llegó a la conclusión de que el diseño de programas educativos se compone de dos partes. Estas son:

2.4.1 Instrucción Ayudada por Computadora

La instrucción o conocimientos que se desean transferir al estudiante o Instrucción ayudada por computadora. Esta instrucción contenida en el programa educativo es independiente de la codificación del programa en el sentido de que se deriva del material dado en clase. Dicha instrucción se corresponde con la instrucción dada normalmente por el maestro en el salón de clases. Es decir que se inicia introduciendo los conceptos básicos, luego las relaciones entre estos conceptos, pasando a la resolución de problemas.

Pero se encontró que para que la instrucción pueda ser efectiva, es necesario que dé un paso más. Se necesita que la instrucción lleve al alumno a desarrollar su creatividad y a ser capaz de hacer juicios dialécticos sobre los conceptos que se le han presentado. Aquí es donde Comienza la diferencia entre la instrucción normal y la instrucción ayudada por computadora. La adaptación de dicha instrucción para su utilización en computadoras permite que los estudiantes hagan simulaciones dinámicas con los objetos de estudio y sus relaciones, donde además la manipulación directa de los objetos y la experimentación con los mismos permiten desarrollar conceptualizaciones que van más allá de las que se pueden lograr en clase.

2.4.2 La Interfase

Adicionalente a al conjunto de conocimientos contenidos en el programa educativo, el programa debe estar diseñado de tal forma que la interfase con su usario, es decir la serie de comandos y acciones que el estudiante puede realizar con el programa, lleven un diálogo natural e intuitivo. Se encontró que la manipulación directa aunado con una representación visual de conceptos informativa, además de capturar la participación del estudiante, permite la desaparición mental de la computadora como mediador y logra la concentración en conceptos y objetos representados que son los que se desea el alumno llegue a dominar.

2.5 Definición del Sistema

2.5.1 Metas

2.5.1.1 Desarrollar una recopilación de las técnicas y teorías sobre diseño de programas educativos que den el marco teórico para extender la creación de programas educativos a cualquier área académica.

2.5.1.2 Crear una herramienta que permita a los estudiantes de programación básica desarrollar el pensamiento algorítmico como preparación previa al aprendizaje de programas educativos.

2.5.1.3 Crear una herramienta que permita a los alumnos que nunca han estado en contacto con la tecnología educativa una exposición a ésta sana y creativa.

2.5.1.4 Desarrollar una herramienta que permita a los estudiantes de computación, concetrarse en la resolución de problemas reduciendo la carga de cognocitiva que deben asimilar a sólamente el pensamiento algorítmico introduciéndolos a los lenguajes computacionales.

2.5.1.5 Ofrecer la especificación de un lenguaje estructurado de alto nivel que pueda ser utilizado como lenguaje introductorio a la programación y que facilite el desarrollo del enfoque estructurado de resolución de problemas al mismo tiempo que el pensamiento algorímico en las personas que lo utilicen.

2.5.2 Limitaciones y Delimitaciones.

2.5.2.1 El programa fué realizado en el lenguaje gráfico CT (Carnegie Tutorial) y necesita como mínimo una máquina IBM o compatible con monitor EGA.

2.5.2.2 Sólo se incluyó un conjunto reducido de instrucciones de 8 instrucciones en la especificación del lenguaje LEPP.

2.5.2.3 Sólo se implementó un conjunto de 8 instrucciones para la creación de diagramas de flujo y listados en los tres lenguajes utilizados (Pascal, C y LEPP).

2.5.2.4 Los programas generados sólo pueden ser interpretados por el sistema Newt y no pueden correr autónomante.

2.5.3. Producto Final.

2.5.3.1 Investigación bibliográfica sobre diseño de software educativo, principalmente programas interactivos con interfases gráficas.

2.5.3.2 Especificación parcial de un lenguaje de alto nivel e implementación de un conjunto reducido de instrucciones. El lenguaje será llamado LEPP (Lenguaje Estructurado Para Principiantes)

2.5.3.3 Implementación en un programa editor de diagramas de flujo que permita la creación, modificación y ejecución de éstos para su evaluación.

2.6 Estrategia de Solución

Para ayudar a los alumnos a desarrollar el pensamiento algorítmico, se propuso desarrollar un sistema que facilitara la resolución de problemas y experimentación al mismo tiempo que introdujera la alumno a la utilización de lenguajes de alto nivel.

Para lograr esto, se diseñó el programa en forma de simulación interactiva. Esto correspondiente a la parte instruccional del programa. Aunque en su fase inicial no contiene ningún tutorial, contendrá un compeljo sistema de manejo de error que proteje al usuario de cometer errores fatales al mismo tiempo que lo instruye en la forma de evitarlos.

Con respecto a la interfase, se diseñó Newt como un editor gráfico donde las opciones más comunes están al alcance de la mano constantemente y que alienta la experimentación. Los errores son detectados al instante y corregidos. Un programa realizado en Newt está siempre listo para correr, libre de errores.

La pantalla debe estar muy bien organizada para que el alumno pueda ver la relación entre el diagrama de flujo, el listado que genera y la forma en que se ejecuta.

Debe estar diseñado de tal forma que en futuro próximo se puedan añadir funciones de ayuda en línea y tutoriales.

Por último debe estar integrado con un rico menú de ejemplos que puedan ser cargados en línea y probados.

2.7 Recursos

2.7.1 Hardware: El desarrollo de Newt se llevó a cabo en una IBM ps 55sx con procesador 80386sx, coprocesador matemático, tarjeta Token Ring y ratón. En esta misma máquina se desarrolló la documentación de esta tesis

2.7.2 Software: Para la programación de Newt se utilizó el ambiente cT desarrollado por al Universidad de Carnegie Mellon y el DOS 5.0. Para la documentación se utilizó además del DOS 5.0, MicroSoft Windows, MS PaintBrush, MS Write, MS Word para Windows, Micrografx Designer, WinGif y Gcp. Los dos últimos procesadores de imágenes. Para la investigación se utilizó software de RED de FTP software; emuladores de terminal sobre todo tn3270, y WinQVTNet, emulador de terminal para Windows.

2.7.3 Documentos: The cT language, Manual de referencia del cT, Manual del DOS, Manual del usuario de BitNet, Manual de Word para Windows y Manual del Usuario de Windows.

2.8 Plan de Trabajo.

2.8.1 Descripción de actividades.

Actividad	Periodos año 1991
a) Análisis bibliográfico de la literatura relacionada.	Ene-Feb
a.1) Análisis bibliográfico de la literatura sobre diseño de software educativo.	Ene
a.2) Análisis bibliográfico sobre programación orientada a objetos.	Feb
b) Especificación del lenguaje.	Febrero
c) Diseño de las estructuras de datos para manejar las instrucciones.	Febrero
d) Diseño del ambiente de programación.	Mar-May
d.1) Diseño del editor de diagramas de flujo.	Mar-Abr

d.2) Diseño del codificador y del intérprete del programa.	Abr-May
e) Redacción del borrador de tesis.	Ene-Jun
f) Redacción formal de tesis.	Julio
g) Reuniones con el asesor de tesis	Ene-Jun

2.8.2 Gráfica de Gant

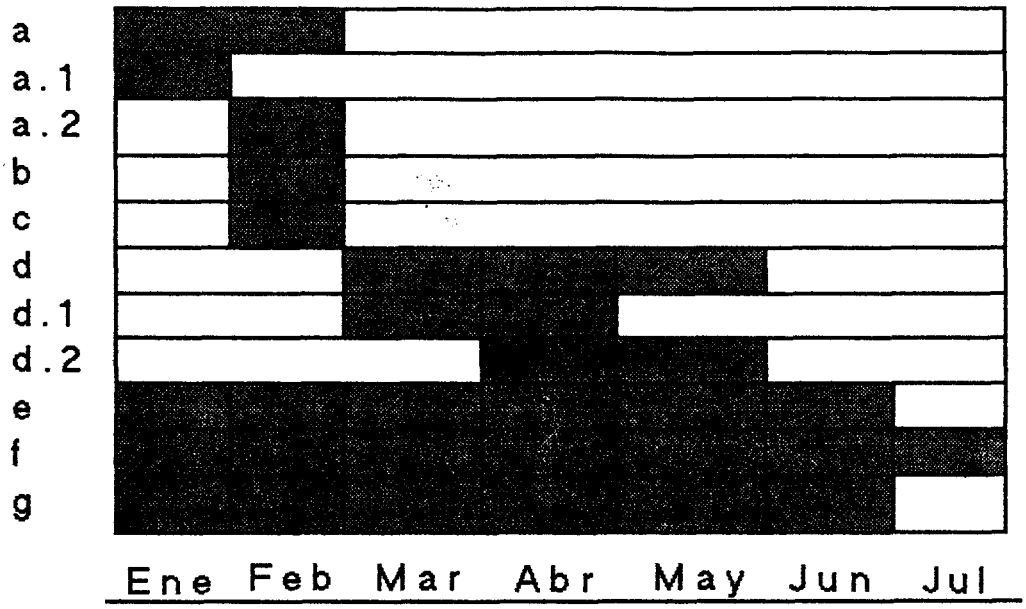


Fig. 1 Gráfica de Gant del Desarrollo de Tesis

CAPITULO III

LOS DIAGRAMAS DE FLUJO

3.1 Introducción

Las materias de programación básica generalmente forman parte del plan de estudios de muchas carreras en los primeros semestres y nadie puede negar su importancia en la formación del estudiante. El crear programas con los lenguajes que se enseñan en estas clases no es cosa fácil. La mayoría de las veces un trabajo arduo no es suficiente para realizar un buen diseño e implementación de un programa. Se requiere de un gran esfuerzo mental y una planeación cuidadosa. Adicionalmente, es necesario aprender una forma diferente de pensar. Esta nueva forma de pensar requiere del individuo que especifique paso a paso la resolución de un problema. Para las personas recién introducidas a la tecnología de la computación, esta forma de hacer las cosas puede no resultar sencilla sino además, muy difícil de llevar a cabo. La herramienta que tradicionalmente se ha utilizado para inculcar este tipo de pensamiento llamado algorítmico es el diagrama de flujo. Sus ventajas son muchas y a continuación se discuten

3.2 La Computación como Materia Curricular en La Universidad

La investigación y el desarrollo que llevó la tecnología de las computadoras a su estado actual fué hecho en gran medida en universidades. Es entonces natural que hayan sido éstas las primeras en hacer uso de la nueva tecnología. Las instituciones educativas desde fechas tempranas en la historia de la computación han incluído de una manera u otra la utilización de computadoras en sus cursos regulares. Programar, sea cual fuere el lenguaje computacional, nunca ha sido una tarea sencilla. Por ésto, al mismo tiempo que se utilizaba la computadora como herramienta en la ingeniería y las ciencias sociales (y se veía su potencial), la mayoría de las instituciones de educación media y susperior han incluído en sus curriculums académicos por lo menos una materia de computación básica en la que se enseña algún lenguaje[1].

¿Cuáles son las ventajas de utilizar computadoras? Primeramente son incansables al realizar cálculos repetitivos, lo cual las hace ideales para la ingeniería. El diseño de intercambiadores de calor, la simplificación de expresiones booleanas para diseño de circuitos, el diseño de sistemas hidráulicos y la simple utilización de métodos numéricos para la resolución de problemas generales; como lo son el encontrar raíces de funciones, la integración y derivación numérica, etc, todos estos diseños necesitan de tediosos

[1] Dado que la tesis no trata sobre linguísitca, cada vez que se hable de un lenguaje se estará refiriendo a un lenguaje computacional.

cálculos repetitivos que se tienen que llevar a cabo una y otra vez hasta alcanzar las especificaciones requeridas. Otra cualidad de las computadoras es su capacidad de manejar grandes cantidades de información indiferentemente. Esto las hace indispensables para la administración, las ciencias sociales y la estadística, donde con frecuencia se tiene que manejar información de toneladas de encuestas aplicadas o se tienen que computar estadísticas de miles de datos recabados en una población. Cabe señalar que se utilizan técnicas similares en el análisis y diseño de experimentos así como en muchas otras áreas del conocimiento.

En un principio, las computadoras consumían enormes cantidades de recursos energéticos y el costo era muy alto. Además eran muy lentas. Pero estas desventajas no impidieron que se extendiera su uso. Hoy en día, una microcomputadora consume menos energía que un foco de 100 watts, y algunos modelos han alcazado precios muy bajos. Pero la verdadera utilidad académica es su rapidez en el procesamiento de información. Antiguamente, la falta de velocidad se compensaba con jornadas nocturnas (las computadoras no necesitan descansar). Ahora, su velocidad y alta precisión permiten la ejecución de complicados programas, con miles de cálculos, en unos minutos. Esto ha dado lugar al desarrollo de la teoría de la Simulación como disciplina, siendo otro de los motivos por el que los alumnos de ingeniería deban aprender lenguajes de programación, y así programar simuladores.

Inicialmente, el lenguaje de programación en la ingeniería fué el Fortran. Posteriormente se desarrollaron un sin fin de lenguajes cada uno con una especialidad, PL/1 y Fortran IV para ingeniería, Pascal y Algol para usos generales, Cobol para la administración, C para el diseño de sistemas operativos y comunicaciones, Basic y Logo para niños etc.

Sin embargo, ningún lenguaje tiene una limitación de alcance para no poder realizar tareas típicamente asociadas con otros lenguajes. El C y el Pascal comparten muchas áreas de aplicaciones, el C se ha utilizado para simulación y diseño al igual que el Pascal, el Basic ha invadido el área del Cobol y se ha utilizado para programar muchos sistemas administrativos, y el Cobol se ha utilizado para desarrollar programas de ingeniería, y la lista puede continuar. Esto ha llevado a concluir que el lenguaje que se enseñe en un primer curso de programación no es lo importante sino el enfoque de resolución de problemas que se adquiere [FORS74], y de paso, ha terminado con la discusión de que cuál será el lenguaje ideal para la educación (Aunque la mayoría está de acuerdo en características que señalan al Pascal y al Módula 2 como favoritos).

Si el aprendizaje del lenguaje no es lo importante, ¿Exáctamente qué es lo que se debe enseñar en los cursos de computación introductorios? El lenguaje es el corazón del curso, sin embargo, sin las técnicas mentales adecuadas para la aplicación de los nuevos conocimientos, el saber uno o muchos lenguajes no sirve de nada. Estas técnicas mentales se deben basar en la naturaleza algorímica de las computadoras, y resultan ser totalmente adecuadas para la resolución de cualquier problema con o sin computadora.

3.3 El pensamiento algorítmico

La mayoría de los cursos de computación dedican una cierta cantidad de horas-clase a iniciar a los alumnos en el diseño de diagramas de flujo. El propósito de estos diagramas es el moldear en los alumnos la capacidad de definir una serie de pasos que lleven a la solución de un problema. Esto es lo que se conoce como Pensamiento Algorítmico. Sea pues, un **Diagrama de Flujo una representación gráfica de un Algoritmo.** Y un Algoritmo a su vez, **una sucesión de pasos *relevantes* que conducen a un fin.**

3.3.1 Algoritmos

El ejemplo más socorrido de un algoritmo es el se compone de las tareas necesarias para cambiar una llanta ponchada. Este ejemplo es muy adecuado porque muestra dos características muy importantes que pueden encontrarse en un algorítmo y que son la razón de su gran utilidad. La tabla 1 muestra la primera de ellas que es la toma de decisiones, con las cuales se pueden tomar diferentes rutas dentro del mismo algoritmo dependiendo de las características del medio. Las instrucciónes 2 y 8 pueden o no ejecutarse dependiendo de la existencia de una llanta de refacción.

La segunda característica es el nivel de especificación, es decir, el algoritmo se puede hacer tan general o tan específico como se desee. En el algorítmo de la tabla 1, no especifica como se levanta el auto utilizando un gato hidráulico (colocar gato en posición vertical bien apoyado, buscar endidura en defensa y mover la palanca). Las instrucciones 2, 3 y 4 pueden abreviarse indicando sólamente "sacar equipo necesario" pero esto sería vago. la instrucción 8.1 también se puede especificar enormemente indicando la manera de transportarse a la vulcanizadora (dado que el carro está descompuesto) y qué vulcanizadora escoger. El nivel de detalle en un algoritmo depende de los conocimientos que se pueden asumir en la persona encargada de interpretar el diagrama de flujo. Así, a una ama de casa no es necesario especificarle cómo debe batir un huevo, un ingeniero en electrónica sabe cómo calcular corrientes y resistencias y cualquier persona sabría cómo quitar un par de tuercas y cómo transportarse. Aunque el saber qué tanto debe detallarse un algoritmo aún puede resultar confuso, en programación la respuesta es sencilla. Los algoritmos deben detallarse hasta llegar a instrucciones elementales de lenguaje; tales como lecturas, escrituras, cálculos, etc.

1. Inicio (se detecta la ponchadura)
2. Si hay llanta de refacción, sacar llanta de refacción
3. Sacar gato hidráulico
4. Sacar llave de cruz
5. Levantar automóvil con gato hidráulico
6. Destornillar llanta
7. Retirar llanta ponchada
8. Si no hay llanta de refacción, llevar llanta ponchada a vulcanizar
9. Colocar en su lugar llanta vulcanizada
10. Atornillar llanta de refacción
11. Bajar automóvil
12. Guardar llanta ponchada
13. Guardar herramienta
14. Fin (Continuar camino)

Tabla 1 Algoritmo para sustituir una llanta ponchada

Es obvio que al hacer algoritmos para lenguajes de programación, el diseñador jamás deberá detallarlos más allá de las instrucciones del lenguaje (a menos que desee eficientar la ejecución misma del lenguaje). Sin embargo, existen lenguajes procedurales que permiten la conglomeracíon de varias instrucciones básicas en grupos orientados al cumplimiento de alguna tarea, permitiendo que cierta parte del algorímo sea una subdetallización, o si se prefiere, una generalización. A los lenguajes que permiten ésto se les denomina modulares o estructurales, y los grupos de instrucciones a los que nos referimos se les conoce como procedimientos o subrutinas.

3.3.2 Refinamiento por pasos

La utilización de generalizaciones ha dado lugar a una técnica de diseño de algoritmos que se denomina **Refinamiento por Pasos** o **Programación de Arriba hacia Abajo**. Esta técnica consiste en ir construyendo el algoritmo paso a paso identificando y etiquetando las tareas necesarias para la resolución de un problema utilizando generalizaciones. Ya que se hayan enlistado todas las tareas, se toma cada una de ellas, y se repite el procedimiento anterior pero tomando como problema meta el cumplimiento de la tarea que se acaba de tomar. Entonces se dice que se está **refinando** la tarea, o que se está diseñando de los niveles superiores de especificación hasta llegar a los niveles inferiores de detalle.

Tarea : Arreglar llanta ponchada

1. Preparar herramienta necesaria
2. Desmontar llanta inservible
3. Sustituir llanta inservible
4. Regresar todo a su estado normal

Tarea : Preparar herramienta necesaria

1. Sacar gato hidráulico
2. Sacar llave de cruz
3. Sacar varios (guantes, trapos, periódicos, etc)
4. Si hay, sacar llanta de refación

Tarea: Desmontar llanta inservible

1. Desatornillar llanta inservible
2. Colocar gato hidráulico
3. Levantar automóvil
4. sacar llanta inservible

Tarea: Sustituir llanta inservible

1. Si no hay llanta de refacción, llevar a arreglar llanta inservible
2. Colocar llanta funcional
3. Atornillar llanta

Tarea: Regresar todo a su estado normal

1. Bajar automóvil con gato hidráulico
2. Quitar gato hidráulico
3. Guardar herramientas
4. Si la hay, guardar llanta ponchada

Tabla 2 Diseño de un algoritmo para sustituír una llanta ponchada utilizando la técnica del refinamiento por pasos

En el ejemplo de la llanta ponchada (tabla 1), la primera tarea a refinar sería el resolver el problema de la llanta ponchada. el refinamiento podría ser algo como la tabla 2

3.3.3 Programación Estructurada

Uno de los primeros lenguajes que se utilizó para enseñar programación a novatos fué el BASIC. Cuando el BASIC se hizo muy popular, precisamente por su sencillez, se empezó a utilizar en proyectos de gran envergadura. Tarde o temprano los programas que resultaron de estos proyectos tuvieron que se revisados, modificados, mejorados o lo que sea. Se notó que el revisar un programa en BASIC era muy difícil por la utilización de brincos indiscriminados en el flujo del programa. "Si se cumple tal condición vaya a tal instrucción, si no, vaya a tal instrucción. Si se cumplen estas dos condiciones vaya al final del listado y se cumple sólo una vaya al medio, etc". El BASIC no ayudaba mucho a la utlización del refinamiento por pasos. Aunque se pudieran crear subrutinas, estos brincos indiferenciables unos de otros dentro de las mismas rutinas hacían muy difícil analizar estos programas. Entonces fué cuando se creó la programación estructurada.

Programación Estructurada significa llevar el refinanmiento por pasos hasta sus últimas consecuencias. En su forma más sencilla, se puede entender como una prohibición de la utilización de brincos[2] dentro y fuera de los procedimientos o bloques del programa y cambiarlo por una estructuración de los brincos según su categoría. Si el brinco es para realizar una subtarea del programa global, se lleva a cabo con una subrutina de la cual no se puede salir hasta que de alcance el fin de ella. Si el brinco es para crear un rizo (un ciclo) dentro del flujo del programa, entonces se hace con una instrucción especial. Si el brinco es al inicio del ciclo, es una instrucción (un WHILE). Si el brinco es al final, es otra instrucción (un REPEAT). Si el brinco es para crear un ciclo que se repite un número conocido de veces, es aún otra instrucción (un FOR). Y así, con la intención de acabar con el desorden y lo intratable de lenguajes antiguos, se crearon nuevos lenguajes con más instrucciones que estructuraban los cambios de flujo del programa. En el capítulo VII se puede ver con más detalle la utilización de programación estructurada.

3.4 Los Diagramas de Flujo

Un diagrama de flujo no es más que **una esquematización gráfica de un algoritmo**. Es un dibujo compuesto de íconos o imágenes que se corresponden uno a uno con las acciones que se llevan a cabo en el algorítmo. El algoritmo de la tabla 1 tendría un diagrama de flujo similar al de la figura 2.

---

[2] Ver los diagramas de flujo de la sección 3.4. En ellos un brinco se aprecia cuando la flecha que va de una instrucción a otra no va en dirección de arriba a abjo.

A partir de la figura 2 se pueden identificar las características que hacen de los diagramas de flujo herramientas tan útiles del aprendizaje de lenguajes de programación.

a. La utilización de flechas permite claramente ver las direcciones que puede tomar la ejecución del algoritmo según se evalúen las diferenctes condiciones.
b. Se pueden identificar las acciones que se están realizando por medio del dibujo que representa la acción. En la figura 2, el dibujo de una decisión como verificar la existencia de la llanta de refacción es muy diferente al dibujo una acción mecánica como el desatornillar la llanta ponchada.
c. La instrucciones que forman un conjunto (o una rutina) se pueden agrupar visualmente en el dibujo para poderlas identificar.

El diagrama de flujo que represente el algoritmo de un programa computacional estará compuesto por una rica variedad de figuras cada una representando una de las muchas instrucciones que componen un lenguaje computacional. Las más importantes de ellas y su posible equivalente a una instrucción se pueden apreciar en la figura 3. La figura 4 representa la solución clásica al problema de encontrar las raíces de una ecuación cuadrática. El diagrama de flujo que se genera es un ejemplo típico de un diagrama de flujo de un programa computacional.

La utilización de diagramas de flujo tiene una ventaja adicional a las ya mencionadas anteriormente. Los diagramas de flujo permiten desarrollar mecanismos de pensamiento lo suficientemente poderosos que permiten al alumno emigrar sin problemas de un lenguaje a otro. El hecho de utilizar figuras en lugar de instrucciones hace pensar en la solución de problemas en términos de acciones y no de instrucciones específicas. Esto a su vez lleva a que si el alumno puede desarrollar las estructuras y habilidades mentales suficientes, el enfoque algorítmico de resolución se puede aplicar a muchos otros problemas no necesariamente computacionales y ni siquiera académicos.

La desventaja primordial de los diagramas de flujo es que conforme las instrucciones del programa se van multiplicando, el diagrama de flujo se va haciendo más intrincado y difícil de interpretar. Además la ventaja que dan las flechas sobre la dirección de la ejecución del programa se pierde si las flechas tienen que ir de una página a otra o si se cruzan unas con otras. Esto significa que los diagramas de flujo son muy útiles en las primeras etapas del aprendizaje pero pierden su valor conforme el individuo va aumentando su habilidad para diseñar programas. La programación por refinamiento ayuda a resolver estos obstáculos agrupando las acciones que completan una tarea y representándolas por medio de un solo dibujo, reduciendo la cantidad de objetos en diagrama de flujo. El dibujo generado por la tarea, puede a su vez refinarse en algún lugar que no estorbe al resto del diagrama (otra hoja tal vez).

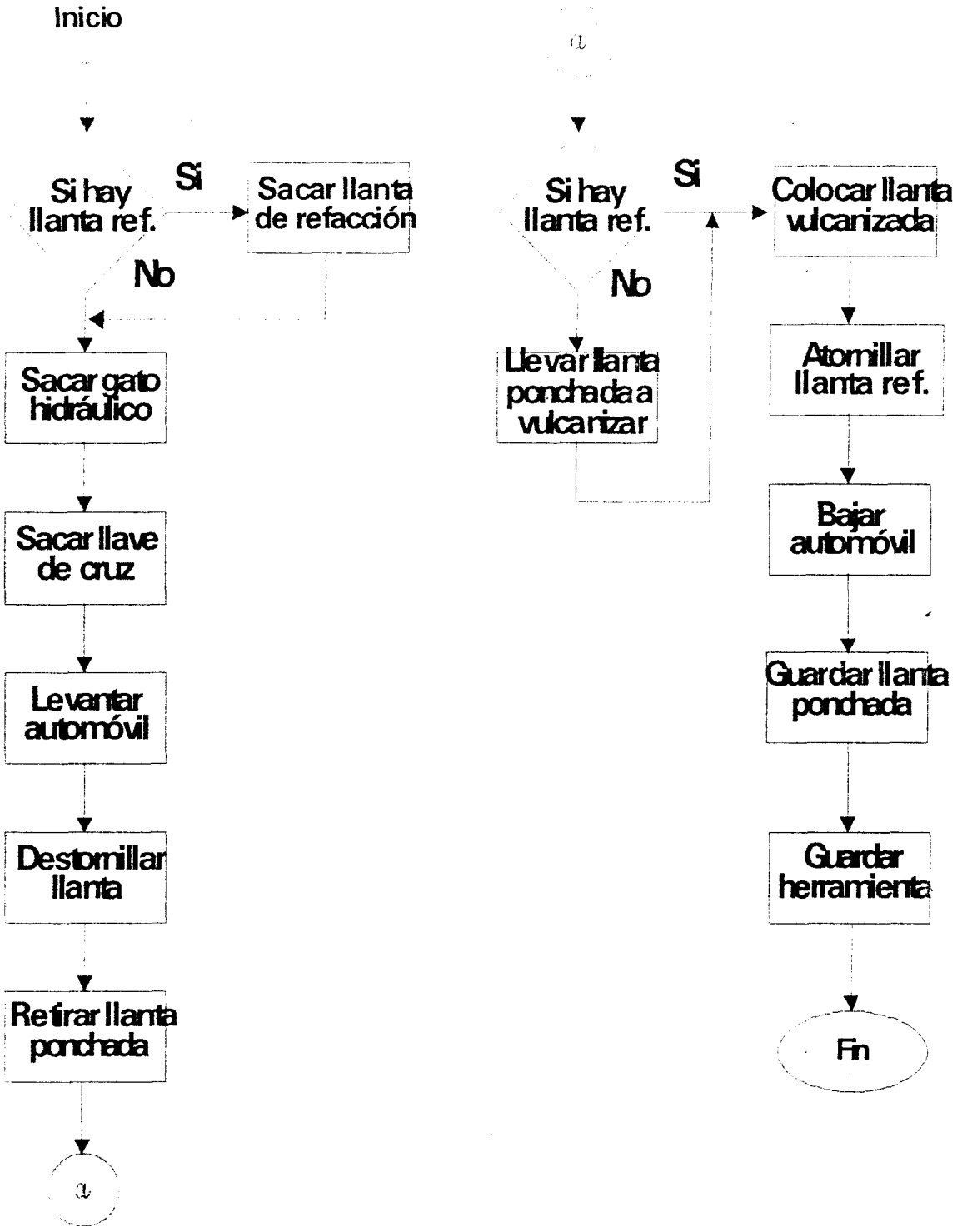


Figura 2 Diagrama de Flujo que representa el algoritmo para cambiar un llanta ponchada

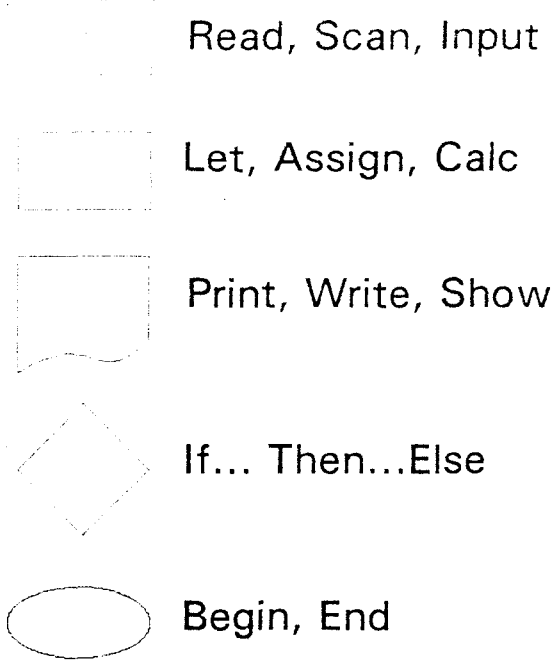


Fig. 3 Instrucciones de diagrama de flujo y su posible equivalente en lenguaje computacional.

3.5 Conclusiones

Los diagramas de flujo pueden llegar a ser una potente herramienta para la formación de las estructuras y precedimietos mentales en la mente de los estudiante de programación. Esto debido a la utilización de la representación gráfica de las acciones, clasificándolas con una imágen diferente según su categoría y por la visualización del flujo de las acciones que se puedan realizar según se vayan tomando decisiones.

Los diagramas de flujo pueden resultar ser difíciles de manejar cuando el pregrama que representan esa grando, de ahí que sean utilizados muy poco por las personas que ya saben programar. Por eso es que debe promoverse su uso en las primeras fases de la instrucción computacional, mientras tienen alguna utilidad y el alumno está dispuesto a usarlos.

Finalmente, cabe señalar que la adquisición de un pensamiento algorítmico y estructurado no sólo ayuda al estudiante a crear programas, sino a enfrentar una gama muy diversa de problemas como los que se va a enfrentar durante su vida académica y profesional.

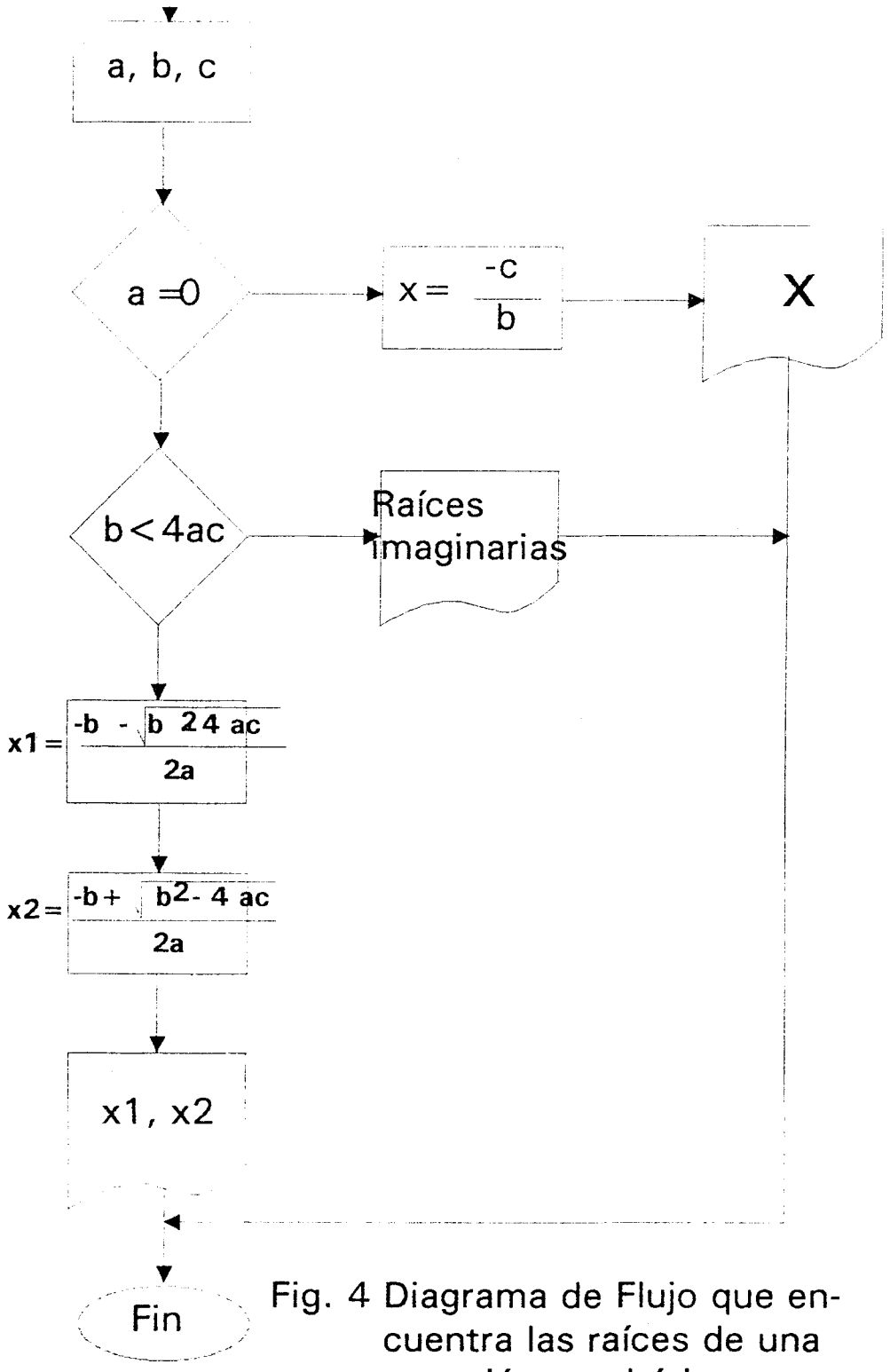


Fig. 4 Diagrama de Flujo que encuentra las raíces de una ecuación cuadrática

CAPITULO IV

LOS PROGRAMAS EDUCATIVOS

4.1 Introducción

La utilización de diagramas de flujo, en teoría, debería preparar al alumno para aprender el funcionamiento de cualquier lenguaje y de paso limar las diferencias entre los alumnos provenientes de distintas escuelas con distintos niveles de exposición previa a las computadoras. Pero no es así. La respuesta tal vez sea que la transición de los diagramas de flujo en el papel a la programación y compilación en un lenguaje de alto nivel es demasiado brusca. Es decir, antes de que el alumno tenga oportunidad de madurar las estructuras mentales especializadas en la resolución de problemas algorítmicos se ve obligado a aprender las peculiaridades de sintaxis de un lenguaje, a utilizar un compilador y a manejar un aparato completamente desconocido como lo sería la computadora y el sistema operativo. Toda esta marejada de nuevos conceptos echan por tierra el avance que el alumno ha obtenido hasta el momento en la estructuración de algoritmos y vuelve a empezar casi de cero.

Es evidente que el tiempo que generalmente tienen los alumnos a su disposición el maestro no es suficiente para dotarlos de las herramientas necesarias para continuar el camino. Y tal vez, ningún maestro jamás podrá tener todo el tiempo para dedicarse a cada alumno todo lo necesario. De ahí que que sea natural el pensar en el uso de las computadoras como instructores por su potencial para impartir instrucción personalizada. Esta no es una idea nueva. La utilización de estas máquinas en el aprendizaje nació casi al mismo tiempo que las primeras computadoras. A lo largo de los años, muchos han sido los proyectos que han utilizado computadoras en la enseñanaza y mucho es lo que se ha aprendido de ellos. Este capítulo, trata de dar una visión global sobre lo que son las computadoras en la educación y cuál es su importancia.

4.2 Historia de los Programas Educativos

Ya han pasado más de 30 años desde que se iniciaron los primeros proyectos que relacionaban las computadoras con la instrucción. Los primeros desarrollos se llevaron a cabo en empresas privadas [MITZ79], donde se intentaba desarrollar sistemas que aceleraran el entrenamiento de los empleados. A finales de 1959 un grupo de ingenieros, físicos, psicólogos y educadores bajo la batuta de Donald Bitzer inician los trabajos para dar vida al proyecto PLATO, en la Universidad de Illinois Urbana-Champaign. Este proyecto no sólo hizo historia por haber sido el primero, sino por mantenerse hasta la fecha en la vanguardia del desarrollo de Software Educativo. Aquellos programas se

desarrollaron con lenguaje maquinal muy difícil de asimilar, haciendo del desarrollo de programas educativos un área reservada a gurus de la informática. 1960 vió el nacimiento por parte de IBM (International Business Corporation) del Coursewriter [HUNK89], diseñado para permitir el desarrollo de módulos instruccionales sin la intervensión de profesionistas. En 1963 El Instituto de Estudios Matemáticos para las Ciencias Sociales (Institute of Mathematical Studies in the Social Science) en la universidad de Standford comienza estudios sobre la instrucción basada en computadoras. Y en 1964 La universidad estatal de Pensilvania se les une estableciendo el Laboratorio de Instrucción Ayudada por Computadora dirigido por Harold E Mitzel, quienes inmediatamente empezaron la investigación y el desarrollo de nuevas ideas y productos. Al cerrar la década, dentro del marco del proyecto PLATO entre 1967 y 1969 se desarrolla el lenguaje de desarrollo de instrucción ayudada por computadora TUTOR por Paul Tenczar.

Al iniciar los setentas (en 1972), la corporación MITRE en cooperación con la universidad Brigham Young empiezan un proyecto conocido como TICCIT, cuya cabeza estaba en la persona de Victor Bunderson. TICCIT quiere decir Televisión Interactiva de Información Controlada por Computadora de Tiempo Compartido (Time-shared Interactive Computer Controlled Information Televisión). La intención principal de este proyecto era la de conectar a los estudiantes utilizando sus recursos disponibles con sistemas remotos donde los estudiantes podían iniciar sesiones con sistemas tutoriales. De estos sistemas tutoriales emergió una idea que ha influenciado de manera importante la teoría de diseño de programas educativos, que es la **Instrucción Controlada por el Usuario** donde los programas son diseñados de tal forma que el usuario decide el flujo la instrucción de acuerdo a sus necesidades.

Por esas mismas fechas, la compañía Control Data Corporation comenzó a difundir el sistema PLATO comercializándolo entre las distintas intituciones de educación superior en los Estados Unidos. Para 1975, El Consejo Nacional Canadiense de Investigación (National Reseach Council) implementó un lenguaje de desarrollo de programas educativos llamado NATAL authoring Language cuya principal característica es ser estructurado como los modernos lenguajes profesionales.

En 1977, nuevamente dentro del proyecto PLATO, se desarrolla el lenguaje MicroTutor a partir del lenguaje TUTOR que permitió el desarrollo en Mini y Microcomputadoras de programas educativos. Este lenguaje fué desarrollado por David Andersen en el Laboratorio de Investigación de Educación Basada en Computadoras CBERL (Computer Based Education Research Laboratory) en la universidad de Illinois en Urbana-Champaign. Al mismo tiempo Alfred Bork, también dentro del proyecto PLATO, inicia importantes avances en la utilización de imágenes y color a través de terminales gráficas conectadas a sistemas de tiempo compartido.

Para terminar esa década, en 1978 nace la computadora Apple II cuyas principales características eran su bajo costo, habilidad para integrar texto y gráficas que provocó

una estampida de programas educacionales que la hicieron aún más popular. Pilot fué un lenguaje diseñado por la Apple computer con el que se hicieron gran cantidad de esos programas.

Al iniciar los ochentas, en 1981 IBM decide no quedarse atrás en el mercado de microcomputadoras y lanza su línea Personal Computer (PC) [ALES89]. Sin embargo, aunque tenía mayor capacidad de procesamiento, su costo la hizo permanecer como la máquina orientada a educación superior y la Apple para educación media y secundaria.

En 1984 nace la Macintosh que dejo estupefacto al mundo. Incluía una interfase gráfica que la hacía más fácil de manejar que cualquier otra computadora, inspirada en las investigaciones llevadas a cabo en Palo Alto California por la corporación Xerox en su laboratorio PARC. Incluía también una increíble capacidad para integrar voz, música, texto y gráficas. Desafortunadamente también incluía un costo elevado, una falta de color que la IBM pronto superó y una gran falta de programas educativos.

Ese mismo año se desarrolla el sistema Omnisim, que era un generador de aplicaciones educativas. En 1985, Bruce y Judith Sherwood inician el desarrollo del lenguage cT en la universidad Carnegie Mellon, basádose en el lenguaje MicroTutor de David Andersen [SHER90]. Este lenguaje estaba principalmente enfocado a desarrollar programas en minis y estaciones de trabajo como las Sun Stations. En 1988 sale la primera version de cT para IBM PC. Al mismo tiempo, la utilización de gráficas y color en los programas educativos llega al apogeo. En 1988 Young [BROW90] describe como los estudiantes de secundaria por medio de la utlización de gráficas podían disectar ranas sin matarlas.

En 1989 nace la computadora NeXT, con una poderosa interfase gráfica y totalmente manejada por medio ratón. La utilización de Interfases gráficas y dispositivos para apuntar como ratón o pantallas sensitivas sientan el precedente de un estándard en el desarrollo de aplicaciones educativas y ya nadie vuelve a pensar en programas basados en texto. En 1989, el proyecto Vesalious descrito por Stephen Roper, minimiza la utilización de cadáveres en clases de anatomía.

4.3 Justificación de los Programas Educativos

La cantidad de conocimientos que se deben aprender para realizar una tarea se conoce como **Carga Cognocitiva** [WHIT90]. Los conocimientos que deben adquirir a la vez los alumnos de computación para poder realizar un programa como los son, el diseño de un algoritmo, manejo de un editor para la codificación, compilación y depuración del programa, son simplemente excesivos. Podemos decir entonces, que los alumnos están cognocitivamente sobrecargados.

De aquí, que la primer forma en que se debe ayudar a los alumnos de computación es disminuyendo la carga cognocitiva que tienen que afrontar. Es decir, que mientras los alumnos aprenden el pensamiento algorítmico, deben ser aislados del manejo de un paquete y de la sintaxis de un lenguaje particular reduciendo así la carga cognocitiva. Si el Software Educativo es capaz de crear una herramienta que efectivamente apoye al alumno en la transición entre el diseño de algoritmos y la programación, éste puede ser la solución.

Se define como **Software Educativo** (SE) cualquier programa, escrito en cualquier lenguaje, en cualquier computadora, cuyo objetivo primordial sea el participar, como sustituto al maestro o como apoyo a aquél, en la tutoría de conocimientos y habilidades que se enseñen en alguna asignatura o disciplina del conocimiento.

Además, y fuera del ámbito que concierne estrictamente al diseño de algorítmos, los programas educativos deben encargarse de realizar los cálculos e iteraciones dejando al alumno las decisiones de enfoque y estratégicas para la solución de problemas. Idealmente, los objetivos de todo programa educativo deben ser los siguientes [WHIT90]:

1. Reducir la carga cognocitiva
2. Permitir la concentración en el aprendizaje de conceptos
3. Mostrar las relaciones entre estos conceptos
4. Presentar problemas realistas que sean retos intelectuales y ayudar en su solución
5. Propiciar la retención en el tiempo del conocimiento.

Durante la redacción de ésta tesis, se utilizarán indistintamente los términos **Programas Educativos** (PE) y **Software Educativo** (SE). En los programas educativos se identifican dos componentes independientes uno del otro:

a. La Instrucción Ayudada por Computadora.
b. La Interfase Computacional.

El concepto de **Instrucción Ayudada por Computadora** (IAC), denotará el diseño de un curso a impartir por computadora. Es decir, **IAC** es la ciencia que se basa en la Psicología, la Pedagogía, la Ingeniería de Software, la Inteligencia Artificial, y otras disciplinas para producir instrucción. La **Interfase** es el medio a través del cual se imparte tal instrucción. Al final, la Instrucción Ayudada por Computadora necesita de una Interfase correctamente diseñada para poder hacer llegar efectivamente el conocimiento a quien lo tenga que adquirir y asegurar una retención en el tiempo adecuada. Podemos decir que la calidad de la IAC hace referencia a la **importancia** educacional del programa, mientras que del diseño de la interfase afecta la **efectividad** del aprendizaje y la **satisfacción** del estudiante. Las técnicas utilizadas por la IAC, pueden aplicarse a muchos otros tipos de instrucción y lo mismo se puede decir de la técnicas utilizadas para el diseño de Interfases. Los capítulo IV y V tratan los

fundamentos de IAC, mientras que los capítulos VI y VII están enteramente dedicados a el desarrollo de Interfases.

4.4 Características de los Ambientes de Programación

El la sección 4.2 se habla en forma muy resumida de los logros con los que ha estado marcada la historia de los programas educativos. Muchos de los hechos que se mencionan como logros no es el desarrollo de aplicaciones en particular sino de ambientes de desarrollo de IAC. Algunos de los ya mecionados son, Coursewriter, TUTOR, DAL, y el mismo Carnegie Tutorial o cT.

Debe recordarse que los primeros programas educativos fueron programados en lenguaje maquinal o ensamblador, que hacían del diseño de estos programas algo extremadamente difícil y reservado a especialistas en computación. De ahí la tendencia a formar equipos de diseño donde se incluyen especialistas en la materia, educadores, diseñadores de gráficas y programadores. Sin embargo, en una gran cantidad de situaciones, por ejemplo en instituciones educativas, una sóla persona toma todas las funciones. Es por eso que la búsqueda de mejores lenguajes y ambientes de diseño que puedan manejar una gran variedad de tareas relacionadas con el diseño de IAC ha sido constante. Claramente, si no existieran adecuadas herramientas para el desarrollo, probablemente el software educativo estaría en una etapa muy primitiva, donde las personas expertas en la materia participarían como asesores y no como diseñadores directos y los objetivos mecionados en la sección anterior estarían muy lejos de ser alcanzados.

S. Hunka [HUNK89] menciona que las principales características de los ambientes para el diseño de programas educativos y son las siguientes:

4.4.1. **Flexibilidad.** Debe permitir la creación de IAC de una manera que se acomode a los estilos personales de cada individuo, por ejemplo algunas personas prefieren desarrollar de una manera lineal como en la secuencia natural de instrucción, mientras que otras prefieren crear modos separados e independientes que pueden ligarse.

4.4.2. **Minimización de la Memoria del Autor.** Muchos cursos computacionales requieren hasta de 50 horas para completarse, esto significa que el autor debe mantener en su mente una gran cantidad de conceptos,objetivos y objetos. Los ambientes pueden facilitar la tarea a través de varias ayudas, como cuadernos de notas electrónicas, documentación de secuencias, listas de objetos gráficos, variables y procedimientos.

4.4.3. **Adaptación a la Experiencia.** Esto se puede lograr jerarquizando las funciones a la disposición del autor de tal forma que los novatos tengan a su

disposición las más simples, y con forme vayan avanzando las funciones más complejas se van poniendo a su disposición. Una forma sencilla de hacer esto es por medio de un menú donde el autor se autocategoriza. Si se define como novato, los menús son cortos y los mensajes son más explícitos pero más distractivos.

4.4.4. **Utilización de Estrategias de Instrucción.** La experiencia sugiere que los buenos sistemas educacionales no siguen un sóla técnica de instrucción (ver capítulo V). Por ejemplo, algunas secuencias se hacen de forma lineal como tutoriales, otras con ejercicios y otras más con simulaciones interactivas.

4.4.5. **Interacción con Periféricos.** Esto es muy importante porque es muy raro que un sistema pueda dar todos los servicios computacionales por sí sólo. Es posible que se dese accesar archivos donde se guardan registros individuales sobre el desempeño de los alumnos, o se puede desear accesar el sistema operativo para identificar archivos importantes, se pueden imprimir reportes, etc. Además, las nuevas tecnologías están poniendo al alcance de los diseñadores de PE herramientas que abren nuevas fronteras a la instrucción computarizada, como son los video discos, video grabadoras, discos ópticos, etc.

4.4.6. **Minimización de Errores.** Esta es una de las funciones más importantes de un sistema de diseño de IAC. El sistema debe intentar capturar y evitar los errores que puedan ser críticos antes de que el sistema sea declarado como producto final. Un caso típico sería el asignar tipos incorrectos a variables, accesar partes a memoria incorrectas, hacer brincos a procedimientos inexistentes, etc.

4.4.7. **Depuración.** Este aspecto es de los que más se ignoran. Un sistema que contenga auxiliares para depurar podría incluir funciones como fijar puntos de paro, donde la ejecución se detiene para verificar el valor de las variables, opción de seguimiento paso a paso, para verificar el comportamiento del programa en forma pausada, monitoreo de registros y variables. También puede incluir facilidades para prefijar valores a variables, como si todas las respuestas sean correctas, etc. Es importante que el proceso de depuración pueda interactuar con el proceso de edición y ejecución para lograr resultados óptimos. En suma, el diseñador debe ser capaz de encontrar las secciones de código que llevan a errores de una forma fácil y rápida.

4.4.8. **Recuperación de Errores.** Siempre que se diseñe un programa se cometen errores; se borran secciones que después resultan haber tenido mucho valor, o se incluyen comandos erróneos. Un sistema debe incluir procedimientos para deshacer acciones incorrectas y formas de probar secuencias de instrucción antes de incluirlas definitivamente en el código.

4.4.9. **Ayuda en Línea**. Estos tipos de sistemas por su complejidad, contienen muchas funciones, algunas de ellas con muchos parámetros. Los sistemas que incluyen ayuda en línea ayudan a acelerar el tiempo de desarrollo y disminuir la frustración. La ayuda se puede implementar utilizando hypertexto, donde se posiciona el cursor en una palabra tópico y al llamarse el comando aparece la ayuda relacionada con ese comando. También es conveniente proveer acceso a la ayuda por índice y por tema, además de permitir aumentar la ayuda y guardar notas.

4.4.10. **Menús y Ventanas**. Las ventanas son excelentes para separar procesos diferentes en el desarrollo del software educativo. Edición, ejecución, ayuda, depuración, etc, pueden colocarse en ventanas diferentes para ayudar a organizar el ambiente de trabajo. Los menús ayudan a tener constantemente a la mano toda la gama de funciones que se pueden accesar.

4.4.11. **Atajos**. Siempre es deseable que la mayoría de las funciones incluídas en menús puedan ser accesadas por secuencias de teclas especiales, ctrl-tecla, alt-tecla y teclas de función, porque ayudan a acelerar el trabajo de los diseñadores expertos.

4.4.12. **Respaldo**. Este servicio muy recomendable y también contínuamente ignorado. En todo momento existe la posibilidad de que un error inconsiente elimine muchas horas de trabajo y esfuerzo. Un adecuado sistema de respaldo puede eliminar frustraciones y salvar del desastre proyectos muy avanzados.

4.4.13. **Documentación**. El sistema deberá intentar en todo momento obligar al diseñador de proveer el mínimo de documentación de los aspectos técnicos y académicos del proyecto que se esté desarrollando. La inclusión de comentarios puede ahorrar muchas horas de depuración especialmente si la persona que corrige los errores no es el diseñador original, además que ayudan a la integración de módulos.

4.4.14. **Ejecución**. Un proyecto que no puede ejecutarse independientemente del sistema que lo creó no sirve más que para fines demostrativos. Los programas deben generar código que puede puede ser ejecutado en ausencia del ambiente de diseño para poder gozar de las ventajas en velocidad y espacio que implican esta separación.

4.4.15. **Edición**. Esta es la parte más importante del sistema. El proceso de edición no debe sólo reducirse a la edición de texto como en un procesador de palabras porque esto sólo genera proyectos largos y tediosos. El editor debe ser capaz de editar toda una serie de objetos desplegables en pantalla, como texto, gráficas, menús, dibujos, botones, etc. También debe ser capaz de

interactuar directamente con la pantalla para que el autor pueda verificar instantáneamente los resultados de las modificaciones hechas al proyecto conforme se vayan haciendo. De ser posible, deben incluírse herramientas de dibujo en línea para el diseño de pantallas. Sin embargo, la pantalla no es más que otro de los objetos que conforman la interacción con el usuario y el proceso de edición debe ser capaz de mantener listas de objetos, incluyendo la pantalla y los demás que se han mencionado. El proceso de edición debe interactuar con el de depuración y ejecución de tal forma que el ambiente verdaderamente sea un sistema integral de desarrollo. Incluso existen muchos sistemas modernos que evitan al autor el escribir código de algún lenguaje utilizando generadores de aplicaciones interactivos, donde se van escogiendo de menús objetos, los cuales se colocan directamente en pantalla y se les asignan funciones también incluídas en menús a disposición del diseñador.

4.4.16. **Control de Flujo.** Un sistema que no puede alterar el flujo de la exposición es un sistema que sólo sirve para fines demostrativos y que tiene muy poca utilidad práctica. Un sistema de desarrollo de programas educativos debe incluir varias facilidades con las que el alumno puede afectar el orden de la presentación, como brincos programados con retorno o sin él, brincos por menús o por botón, etc.

4.4.17. **Análisis de Respuesta.** Esto es vital para que un sistema pueda ser verdaderamente interactivo. La experiencia a demostrado que un sistema de IAC que no demanda interacción por parte del alumno sólamente lo aburre. Por esto, deben proveerse facilidades para interpretar las respuestas del usuario de varias maneras posibles; ratón, teclado, pantalla sensitiva, etc. Deben proveerse mecanismos para indicar cuáles respuestas son correctas y que acciones generan, cuáles incorrectas y cuáles los mensajes de error para corregirlos. Auxiliares en la interpretación son deseables, cómo funciones de búsqueda dentro de strings, convertidores de mayúsculas a minúsculas, editores de línea de respuesta, etc.

4.5 Tendencias al Futuro

El área de diseño de programas educativos es una de las que más se ve influenciada por los adelantos tecnológicos. De hecho, los cambios producidos en la instrucción basada en computadoras por éstos cambios tecnológicos son tantos, que los mismos diseñadores encuentran muy difícil adaptarse a ellos. No obstante, eventualmente son asimilados, delineando de una manera más o menos clara la forma en que se van a utilizar nuevas técnicas y tecnologías en el futuro. A continuación se describen brevemente éstas tecnologías y su influencia en el desarrollo de programas educativos.

4.5.1 Hypercard

El ambiente Hypercard fue introducido primero en las máquinas Macintosh [BOWE90]. Su facilidad para crear aplicaciones le agenció una gran cantidad de entusiastas desde sus inicios. En el ambiente Hypercard, los programas creados se llaman stacks. Un stack es una colección de tarjetas. Una tarjeta a su vez, es la unidad básica de organización, y es equivalente a una pantalla de información. Por ejemplo, un stack puede estar compuesto por tarjetas que contengan información sobre libros. De esta forma, se crearía un stack equivalente al fichero de una biblioteca. Alternativamente, un stack puede contener tarjetas donde se incluyen unidades de alguna materia académica formando un curso completo.

Una tarjeta esta compuesta por dos niveles. El primero es el nivel de fondo. Los objetos colocados en este nivel son comunes a todas las tarjetas. El segundo nivel es el nivel de tarjeta que contiene elementos únicos para cada tarjeta. Los objetos que pueden estar presentes en una tarjeta son campos, botones e imágenes. Un campo es un contenedor de texto. Cuando se desea que un usuario cree, edite o almacene material textual se utiliza un campo. Por lo general el texto es visualizado a través de una ventana a una pantalla virtual que sólo muestra una parte de la pantalla a la vez. Se pueden importar imágenes a la tarjeta, por ejemplo la digitalización de la fotografía de una persona o un logotipo. Las tarjetas también pueden contener botones. Un botón es un iniciador de acciones. Los botones existen para mandar comandos al Hypercard como mover al usuario de una tarjeta a otra, moverse de una stack a otro, iniciar el manejo de alguna información, o mandar un mensaje o un dispositivo periférico. Los botones se pueden diseñar con varias presentaciones, una simple caja, un botón tridimensional, o un logotipo.

Las acciones se programan utilizando un lenguaje especial que se llama HyperTalk. HyperTalk es diferente a Pascal o Basic en el sentido de que la ejecución de las instrucciones no es estrictamente lineal, sino que las acciones son generadas por mensajes que maneja el ambiente Hypercard y son atrapados por algún objeto dentro de la tarjeta, por ejemplo presionar un botón del ratón mientras el cursor se encuentra sobre un botón de la tarjeta puede generar la aparición de ayuda en línea.

Finalmente, HyperTalk es un lenguaje interpretado que conjuntamente con la creación gráfica de los programas permite desarrollar de una manera rápida e intuitiva programas educativos.

4.5.2 Redes.

El uso de redes locales se ha ido generalizando tanto en las empresas privadas como en las instituciones educativas. La principal motivación para la utlización de

redes es la compartición de recursos, ya sean programas, espacio en disco, impresoras o incluso procesadores.

La utilización de redes también provee mayor control sobre estos recursos. Por ejemplo, la asignación de claves de identificación permite llevar un control sobre el desempeño de individuos y preferencias generales. Además, debido a que los recursos pueden ser accesados desde muchas estaciones pero se encuantran controlados por un sólo nodo, el dar de alta, cambiar o dar de baja paquetes individuales se hace más fácil. Al hacer un cambio a un paquete de software en un servidor en red automáticamente se ve reflejado en todas las estaciones. En cambio si las computadoras no están en red, los cambios se tienen que hacer en cada una de las máquinas.

Dentro de la interconectividad para la educación existen dos tendencias: interconectar en una red local libre o en un sistema integrado de aprendizaje (SIA) [BLAS90]. Un sistema integrado de aprendizaje es una red local, pero contiene un administrador de la instrucción que se encarga de manejar la relación entre los programas educativos y los estudiantes. además, más del 50% de los paquetes educativos son provistos por el vendedor del sistema integrado. Las ventajas de cada sistema son las siguientes :

Sistema Integrado de Aprendizaje:

- Permite centralizar la toma de decisiones porque todos los programas educativos y las computadoras se adquieren al mismo tiempo logrando una homogeneidad en el equipo.
- Permite un mayor control de calidad y control sobre el desempeño del estudiantado.
- Permite cubrir de una forma uniforme el material en los planes de estudio.

Red Local Libre

- Toma de decisiones descentralizado. Cada encargado de laboratorio o maestro busca el software y el equipo que más le satisfaga.
- Manejo descentralizado de laboratorios computacionales
- Promueve el profesionalismo de los maestros al delegarles a la toma de decisiones
- Permite mayor autonomía técnica puesto que el ensamblaje del equipo es por partes y no de una manera integral.

Las principales desventajas de ambos enfoque son:

- Costos muy altos de instalación y mantenimiento

- La instalación requiere de mucho tiempo y esfuerzo
- Existen pocos programas realmente diseñados para funcionar en red
- Falta de estandards en téminos de precios de licencias, arquitectura, organización, etc.

4.5.3 Video Interactivo

La utilización de gráficas de alta resolución y técnicas de visulización como codificación en geometría y color ayudan a aumentar el aprendizaje en las ciencias exactas. No obstante, estas técnicas son muy poco aplicables cuando se esta intentando enseñar materias que no están relacionadas con los números, por ejemplo la psicología y la ética. Es en estas áreas donde más popular se ha vuelto la utilización de videos interactivos. La instrucción ayudada por computadora auxiliada del video ayudan al estudiante ha desarrollar la habilidad para hacer decisiones racionales en un medio que parecerá casi auténtico, muy cercano al medio real donde se tomarán estas decisiones.

La presencia del video en los programas educativos lleva a la utilización de simulaciones interactivas a la frontera última de virtualidad. Es decir que se pueden presentar situaciones reales, como juicios, entrevistas, procedimientos, razonamientos, etc. que permiten al estudiante involucrarse con sus sentidos y razonamiento en las situaciones analizadas.

Adicionalmente la utilización de videos tiene las siguientes ventajas [HANS90]:

1. La tecnología de video interactivo facilita la reflexión colectiva. Esto se logra porque los programas educativos apoyados de video proveen de un medio de observación controlado totalmente por el usuario y más importante, puede proveer de un ambiente para la reflexión. Y la reflexión es un proceso inherentemente social. La reflección implica el debate de diferentes puntos de vista, retos a las creencias establecidas y un compromiso emocional al resultado de las conclusiones.

2. Las experiencias adquiridas durante la utilización del video interactivo deben ser retroalimentadas en el salón de clase. La educación superior ha establecido estandares delimitando lo que es relevante para el aprendizaje y lo que no es. Y el lugar para negociar tal relevancia es el salón de clases. La presencia del maestro en tales debates es requisito indispensable en este proceso debido a su poder de recompensar el aprendizaje.

3. Las herramientas proveen a los alumnos de auténticas experiencias visuales valiosas por sí mismas. Lo que la mayoría de los estudiantes más aprecian en la utilización de esta tecnología es la oportunidad de ver de cerca lenta y repetidamente acciones auténticas donde el protagonista principal es la habilidad

que están tratando de dominar. El mayor impacto se tiene cuando los objetos del aprendizaje son estudiantes novatos que todavía no han desarrollado criterios sobre sus propios patrones de comportamiento.

El video interactivo es sólo una parte de lo que ahora se denomina "Multimedios" que además incluye sonido, texto y gráficas.

4.5.4 Sistemas Basados en Conocimientos

En la última década la Inteligencia Artificial se ha desarrollado de prototipos de laboratorio ha convertirse en componentes principales en muchas áreas del desarrollo tecnológico y los PE también han dado cabida a la creación programas inteligentes. El crear dispositivos sofisticados con una inteligencia cercana a la del ser humano ha sido desde siempre un sueño y una fantasía del ser humano. Esta idea es especialmente tentadora en el área educativa donde crónicamente se nota una falta de tiempo y maestro, es decir, de personalización de la educación. Aunque las ambiciones de la inteligencia artificial han bajado bastante en la última década debido al reconocimiento de las limitaciones computacionales, el desarrollo de programas inteligentes ha sido intenso. En educación estos programas se conocen como Sistemas Tutoriales Inteligentes (STI), Tutores Basados en Conocimientos (TBC) o simplemente Sistemas Tutores Expertos (STE).

Edmund Hansen [HANS90] y muchos otros han encontrado que personalmente los estudiantes esperan que los programas educativos sean más herremientas que supersistemas cuasi inteligentes. Con todo, se ha encontrado que la utilización de Inteligencia Artificial acelera el proceso de aprendizaje hasta en un 30%, haciéndolo más efectivo en la relación aprendizaje-costo [PERE90]. Esto sin tomar en cuenta que en muy raras situaciones los sistemas expertos utilizan técnicas de visualización, manipulación directa o multimedios, pues generalmente están escritos en texto por medio de línea de comandos. Por lo que incluso puede ser que los alumnos los encuentren simplemente aburridos, por lo que su aplicación en la educación superior en ocaciones es pequeña reservándose a empresas privadas. Sin embargo, debe subrayarse que éstos sistemas son altamente interactivos de ahí su efectividad en la enseñanaza.

Los TBC pueden ayudar en varias formas en la instrucción. Primeramente porque el hecho de ser expertos los hace recipientes de una gran cantidad de conocimientos, la llamada Base de Conocimientos. Y en segundo lugar, los razonamientos (consejos expertos) derivados de estos sistemas llevan una lógica y clara fácilmente asimilable por el estudiante (reglas de inferencia).

Son principalmente tres las formas en que los TBC pueden participar en la enseñanza [THOR90]:

4.5.4.1. **Ejecutar Casos**. Aquí es cuando se utiliza al sistema experto como tal. Se alimenta cierta información concerniente a un caso, el cual es analizado independientemente por un humano y un sistema experto. Se compara el diagnóstico de la persona con los concejos dados por el experto y de ahí se llega a una decosión.

4.5.4.2. **Entrada de Información Seleccionada**. De esta forma se pone a prueba la solidez del sistema experto. Se presenta un caso cuyo análisis es conocido y este análisis es comparado con aquél derivado del sistema experto. Además se pueden probar individualemente la lógica del sistem experto, su manejo de conceptos y atributos.

4.5.4.3. **Tutorial Basado en Conocimientos**. En esta forma, el sistema basado en conocimientos, genera una cierta secuencia de escenarios donde el estudiante va constantemente comparando sus juicios con aquellos derivados por el sistema experto y aprende de las diferencias.

4.6 Conclusión y Discusión

El desarrollo de programas educativos es una vieja historia. Sin embargo, con tan' sólo hechar una pequeña mirada a esa historia, se puede notar que parece más bien una búsqueda desesperada. Y es que de la enorme cantidad de desarrollo que se ha hecho durante los pasados treinta años en el área de programas educativos, realmente muy pocos proyectos, privados, militares o universitarios han tenido algún éxito.

Pero los pocos proyectos que han logrado avances, han dejado una serie de experiencias que aunadas a los avances de la tecnología permiten afirmar que la presente década va a ser la que va a presenciar el apogeo de los programas educativos. las nuevas herramientas, sobre todo aquellas que ya no requieren que la persona codifique en algún lenguaje oscuro van a permitir el desarrollo de más y mejores programas.

Sin embargo, cabe hacer una pregunta, ¿Qué tan lejos se puede llegar? ¿Es posible que se llegue a alcanzar tal grado de tecnología que los académicos puedan desarrollar talentosas aplicaciones educativas sólo para ser sustituídos por ellas en el salón de clases? La respuesta parece ser por el momento un rotundo "No". Por dos razones.

La primera es porque los programas en sí constituyen un sitema cerrado. Es decir que están programados para resolver un número finito de situaciones, no importa que tan grande sea ese número. En algún momento, más temprano que tarde, el sistema llegará a confrontar una situación que no podrá resolver. La psicología necesaria para alentar a los alumnos al estudio y entender sus motivaciones simplemente no puede ser programadada. Y esto a su vez por la sencilla razón de que todavía no exite un ser humano, mucho

menos un ser humano programador, que sepa resolver de forma perfecta y científica, una sóla de las situaciones que debe confrontar el maestro en clase.

La segunda razón no es tanto cibernética como psicológica. De alguna manera, nada parece tener importancia para los alumnos si el maestro no está presente o tampoco le da importancia. El maestro es el ser que tiene el poder de decidir qué es importante, pero más que esto, es quien tiene el poder de gratificar el aprendizaje. Un autómata también puede calificar y gratificar, pero eso sólo convertiría de la educación en una fábrica fría de conocimientos, visión que es preferible dejar a un lado para las películas.

Entonces, se puede concluir que a mediano plazo, el papel de las computadoras en la educación se reducurá a herramientas y complementadoras de la instrucción impartida en el salón de clases.

CAPITULO V

LA INSTRUCCION AYUDADA POR COMPUTADORA

5.1 Introducción

Ya se ha mencionado que los programas educativos tienen treinta años de historia. Estos treinta años representan una gran cantidad de esfuerzos que por algún tiempo sólo llegaban a ser ideas, sugerencias y consejos. Pero ahora, los programas educativos han pasado por la fase infantil natural en su evolución y finalmente se cuenta con una base teórica sólida cimentada en trinta años de estudios para la cual han cooperado un sin número de ingenieros, psicólogos y educadores.

Parte muy importante de esa teoría lo constituye el conocimiento que se tiene sobre los procesos del aprendizaje humano. Sólo conociendo estos mecanismos es posible adaptarlos a la tecnología de la computación de tal forma que su impacto sea un incremento en la efectividad de la educación.

A partir de este fundamento, se puede llegar a especificar una metodología de desarrollo e implementación.

5.2 Reseña de Enfoques de Enseñanza

Es indudable que para que la IAC pueda llegar a tener algún efecto en el aprendizaje, es necesario que la estructura y las técnicas que se utilicen para su desarrollo trabajen armoniosamente a la par que los mecanismos que utiliza el ser humano para adquirir el conocimiento. Con el tiempo, se han desarrollado varias teorías que han aportado mucho a la ciencia de la educación [SMIT89]. A continuación se presenta un resumen de las principales:

5.2.1 Conductismo

Bajo este esquema, el individuo objeto del aprendizaje es conducido a adquirir conocimientos a través de modificaciones en su conducta. Estas modificaciones se logran utilizando una técnica llamada reforzamiento positivo. El reforzamiento positivo es una retroalimentación o estímulo cuyo objetivo es asegurarse que cierta conducta se repita, pero además, el reforzamiento debe hacerse tomando una actitud positiva, es decir, resaltándose los logros, alentando al individuo a que los repita y así alejarlo de las conductas que llevan a cometer errores. Bajo este esquema, la instrucción va llevando al individuo paso a paso a través de una secuencia de

ejercicios hasta el desarrollo de respuestas muy complejas, y en cada paso, el individuo recibe una gran cantidad de retroalimentación. En todo momento se intenta que el individuo cometa la menor cantidad de errores, además de proveer de una gran cantidad de práctica una vez que las respuestas finales deseadas han sido logradas

5.2.2 Neoconductismo

En esta teoría, se trata de inducir el aprendizaje a través de cambios en su conducta, pero utilizando reforzamiento positivo y negativo. Es decir, que el estudiante puede recibir premios o castigos. Sin embargo, los premios y castigos pueden no ser aplicados directamente al estudiante sino a los modelos que se están utilizando como respuesta del sistema. De esta forma, se trata de crear una relación directa entre el estudiante y un modelo que lo represente en el sistema, el cual recibirá los premios y castigos. Además, el material de instrución deberá proveer incentivos para continuar el aprendizaje, es decir, se puede crear un sistema de créditos donde la persona que es objeto de la instrucción se verá estimulada a continuar hasta acumular suficientes créditos para recibir una recompensa.

5.2.3 Procesamiento de la información

Se basa en proporcionar al estudiante con diversas formas de codificar la información que recibe de fuentes verbales. Además de hacerse un análisis de los conocimientos que se deben adquirir, debe hacerse un análisis de las tareas necesarias para adquirir el conocimiento, y las habilidades que se deben desarrollar. De este análisis de tareas, se deriva una lista de prerrequisistos que debe cumplir el estudiante para poder alcanzar los objetivos y considerarlos en el diseño de la instrucción y/o incluirlos. Además, el análisis de tareas debe dar un marco global de la secuencia de la instrucción. Debe proveerse al estudiante con muchas oportunidades para practicar y los ejercicios deben estar diseñados variando el contexto y las variables de manipulación. Para mejorar el aprendizaje, además de la práctica, debe hacerse énfasis en los principios generales que gobiernan el conocimiento que se transmite al educando.

5.2.4 Psicología cognocitiva

En este método, el alumno es llevado a través de una jerarquía de abstracciones. Primeramente el material debe presentarse en formas que sean consistentes con el nivel cognocitivo del estudiante, utilizando ejemplos concretos y abstractos, y formas lógicas para el pensamiento. Se debe hacer conciente al estudiante de las relaciones existentes entre los diversos principios y conceptos del conocimiento a adquirir. El material debe ser expuesto utilizando secuencias tanto inductivas como deductivas y debe permitir que el alumno se involucre en su

aprendizaje dirigiendo por sí mismo la secuencia y velocidad de la exposición dependiendo de sus necesidades y su capacidad. Finalmente, el material debe llegar a un conflicto dialéctico que lleve a desarrollar niveles más altos de pensamiento.

5.2.5 Estilos del aprendizaje

El material debe presentarse en varias formas de acuerdo con las preferencias sociales del los educandos, es decir, aprendizaje en grupo o individual. El material debe ser presentado de acuerdo a las diferentes formas de pensamiento de los educandos, como lo son, abstracto, global, analítico, verbal e icónico. El material debe ser presentado de acuerdo a las preferencias en la percepción de los educandos, esto es, auditivo, visual, tactil, kinestético, etc. El material, además, debe ser presentado en una secuencia que esté de acuerdo a las preferencias emocionales de los estudiantes, estructurado vs opcional. Por último, la instrucción debe incluir exámenes introductorios para certificar niveles adecuados de conocimiento previo, y proveer a aquellos que no califican fuentes alternativas de conocimiento para que desarrollen las habilidades necesarias.

5.3 Un nuevo paradigma del aprendizaje

Desde inicios de los setentas se ha venido haciendo mucha investigación sobre la psicología del aprendizaje. Esta investigación ha provocado cambios importantes en la teoría que han proporcionado bases firmes sobre las cuales se ha venido desarrollando la ciencia de la Educación Tecnológica[1] . Parte esencial de estos fundamentos, es la conceptualización de un paradigma o modelo del aprendizaje. Tradicionalemte, el paradigma sobre el cual se ha fundamentado la enseñanza es el Conductista, descrito en la sección anterior. Este modelo ofrece técnicas de enseñanza claramente definidas y probadas, que aunque hasta ahora han sido razonablemente efectivas, han mostrado graves deficiencias cuando se aplican a la enseñanza tecnológica. La incapacidad de adaptación del paradigma conductista al desarrollo de la tecnología hace evidente la necesidad del desarrollo de un nuevo modelo del aprendizaje.

Robert Tennyson [TENN90a] ha propuesto un modelo el cual simplemente ha llamado **Paradigma Cognocitivo**, que trata de explicar los mecanismos del aprendizaje basándose en una descripción de la estructura interna de la mente. Apartir de los elementos mentales que guardan el conocimiento (figura 5), Tennyson define los distintos tipos de conocimiento que pueden almacenarse y define las estructuras que lo almacenan y extraen. Por último, define los tipos de conocimientos que pueden existir en la mente de un individuo dependiendo de las circunstancias en que estos se utilicen. Las metas del paradigma cognocitivo son las siguientes :

---

[1] Entiéndase por Educación Tecnológica la ciencia que utiliza la tecnología para la educación, y no la educación para la tecnología.

1. Explicar los mecanismos de adquisición del conocimiento (aprendizaje) y su utilización (pensamiento).
2. Describir las estructuras de almacenamiento del conocimiento (memoria).
3. Fundamentar los procesos cognocitivos de más alto nivel (creatividad).

A primera vista es posible notar que el modelo de Tennyson va más allá de los modelos convencionales porque establece que el conocimiento puede provenir tanto de fuentes internas como externas, y establece además, una relación entre estímulos externos y procesos internos. En la figura se pueden identificar los componentes básicos del modelo :

5.3.1 Sensores Receptores. Incluye todas las formas en que la información penetra en el sistema cognocitivo. Básicamente éstas formas son auditiva, visual y táctil. Se entiende como estímulos externos todas aquello que resulta de la tecnología educativa.

5.3.2 Percepción. La información generada interna o externamente pasa a través del componente de la percepción cuya función es detectar la presencia de tal información y examinar su valor potencial. Otra función es la dirigir la atención y determinar la cantidad de esfuerzo necesario para resolver, situaciones.

5.3.3 Memoria Temporal y Memoria de Trabajo. Este componente contiene las memorias que manejan el conocimiento inmediato. La memoria temporal tiene una capacidad limitada y contiene la información por intervalos muy pequeños (unos segundos). La memoria de trabajo conlleva un esfuerzo consiente de la utilización de los mecanismos de codificación entre sí mismo y la memoria permanente.

5.3.4 Memoria Permanente. La adquisición del conocimiento y las acciones para explotarlo ocurren dentro del subsistema de almacenamiento y extracción de conocimiento de la memoria pemanente. Dentro del sistema de almacenamiento se encuentra codificada una base de conocimientos de acuerdo a varios formatos, mientras el subsistema de extracción utiliza habilidades cognocitivas para emplear conocimiento.

5.3.5 La base de conocimientos. Puede ser descrita como una red de conceptos asociados que varía en cada individuo de acuerdo a la cantidad, organización y métodos de acceso de la información. La cantidad implica el volumen de la memoria codificada, organización se refiere a las conexiones estructurales de ese conocimiento y los métodos de acceso son las estrategias de control utilizadas para las acciones pensantes como lo son, hacer memoria, resolver problemas y la creatividad. Las últimas dos actividades pensantes son las que separan a los expertos de los novatos. Es decir que una gran cantidad de

conocimiento no es la llave para un pensamiento experto. Dicha llave se encuentra en la capacidad de encontrar y utilizar el conocimiento apropiadamente.

5.3.6 Tipos de conocimiento. Declarativo: conocer y estar conciente del conocimiento. **Procedural**: el saber utilizar conceptos, reglas y pricipios. **Contextual**: el saber cuando seleccionar estos conceptos, reglas y principios. Los conocimientos declarativos y procedural forman el cuerpo de la base de conocimientos, el conocimiento contextual marcan su organización y capacidad de acceso.

5.3.7 Habilidades cognocitivas. La extracción del conocimiento utiliza habilidades cognocitivas al servicio de las acciones pensantes. Estas habilidades son **Diferenciación e Integración.** Diferenciación se refiere a la habilidad de entender una cierta situación y aplicar el criterio contextual apropiado para extraer selectivamente la información necesaria. Integración implica la habilidad de elaborar o reestructurar el conocimiento existente al servicio de las actividades pensantes. El término genérico para la extracción de conocimiento en forma diferencial o integral es Complejidad Cognocitiva.

5.3.8 Estrategias del pensamiento. Estas se dividen en tres categorias que se describen de la menos a la más compleja. **Hacer memoria**, utiliza sólamente la diferenciación mental del conocimiento y actúa sobre el conocimiento declarativo. **La resolución de problemas**, utiliza la diferenciación y la integración. Esta categoría se forma conforme se van resolviendo problemas, es un ejemplo de utilización de conocimiento procedural y contextual. La acumulación de estrategias de resolución de problemas es directamente proporcional a los problemas resueltos. Así, entre más problemas con diferentes contextos se hayan resuelto, será más sencillo encontrar soluciones para los problemas que han de venir aunque su planteamiento sea muy diferente al contexto del conocimiento inicialmente adquirido. **La creatividad** es la tercer categoría de estrategias del pensamiento. Además de la integración y diferenciación, utiliza la habilidad cognocitiva de crear conocimiento (aumentando la base de conocimientos) basándose en estímulos externos y el conocimiento ya alamacenado.

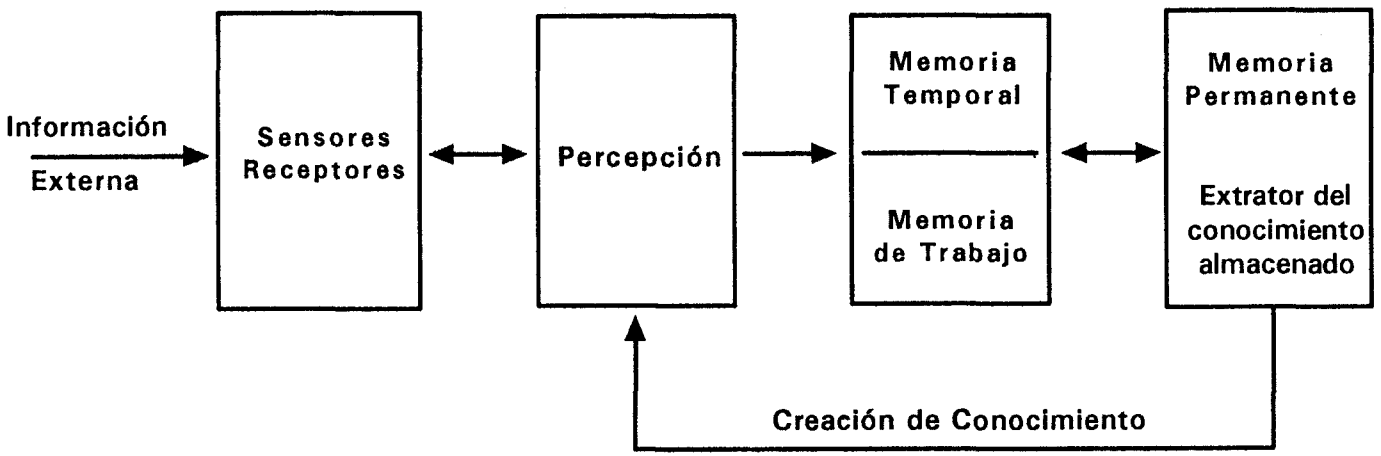


Fig 5 Paradigma Cognocitivo

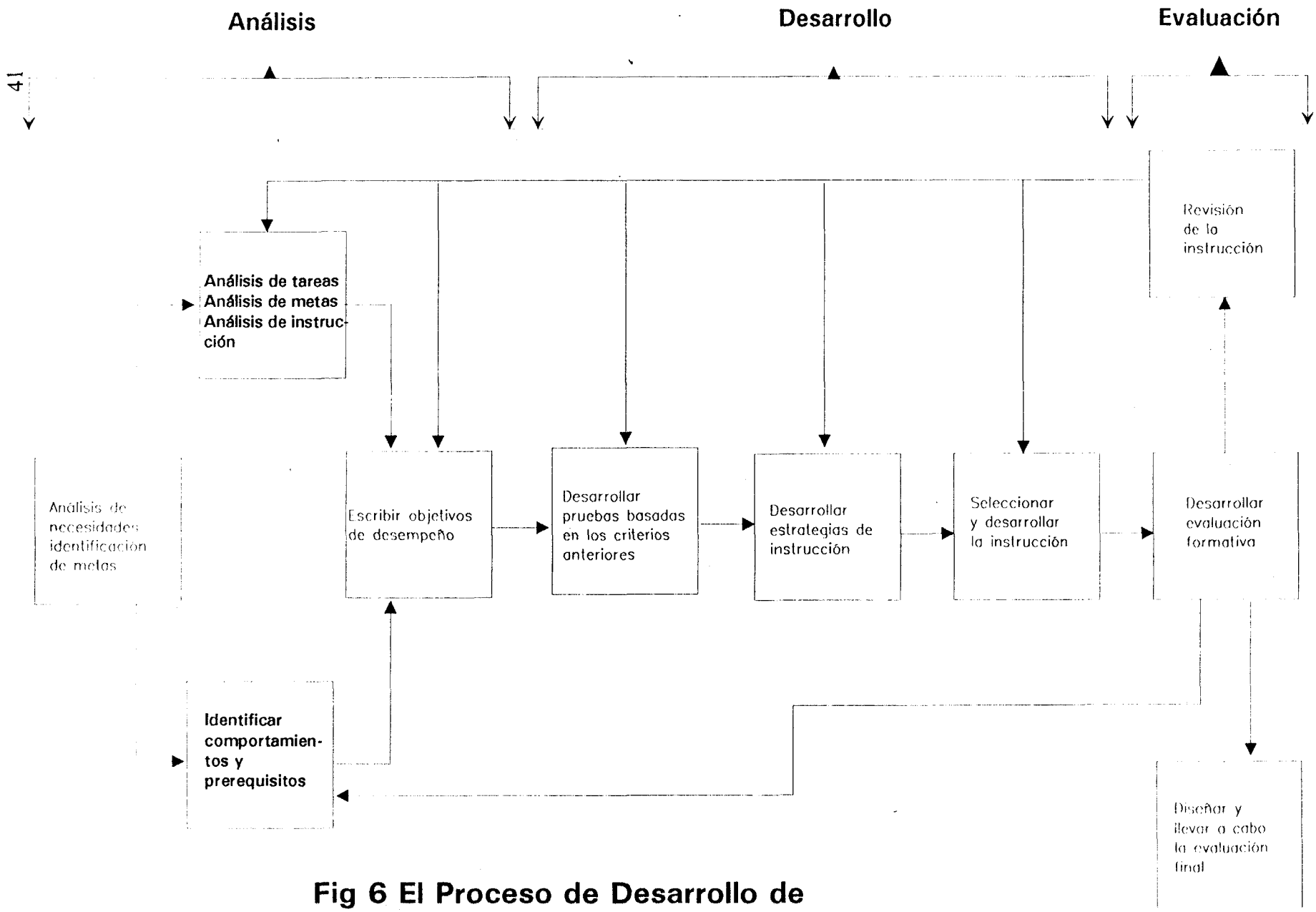


Fig 6 El Proceso de Desarrollo de Instrucción Ayudada por Computadora

5.4 El Desarrollo de Instrucción Ayudada por Computadora

Se ha discutido mucho acerca de los pasos necesarios para el Desarrollo de la Instrucción (DI). Conforme ha pasado el tiempo y la ténicas de desarrollo de SE han madurado, se han publicado muchos modelos. Estos modelos se han desarrollado para la instrucción en ambientes muy diferentes: académica, desarrollo de productos, aprendizaje de herramientas ingenieriles e industriales, educación de alto nivel, e incluso para fines militares. Sin embargo, sin importar su área de aplicación, todos los modelos (o metodologías) de DI tienen algo en común; se componen de tres fases primordiales, estas son:

a. **Análisis.** Donde se especifican los resultados deseados y los estados iniciales.
b. **Desarrollo.** Se planea y produce la instrucción.
c. **Evaluación.** Se verifica la efectividad y el impacto de la instrucción.

La fig 6 se basa en el modelo creado por Dick y Carey en 1985 [OKEY90]. Cada cuadro es una tarea que debe completarse en el proceso de diseño de IAC.

En la fase de análisis se intenta catalogar las deficiencias que tiene que cubrir la instrucción, delimitar sus alcances y listar objetivos. Además de esto, se especifican los requisitos de los sujetos de la instrucción. De esta forma, se sabe qué es lo que se tiene, qué es lo que se necesita, cuánto de lo que se necesita se puede obtener, y como se va a obtener.

La fase de desarrollo consiste en tomar las conclusiones del análisis de construir la instrucción tomando como derroteros las metas y las directrices derivadas de la fase de análisis. también se especifican las estrategias de instrucción, y se selecciona y estructura el contenido del curso.

En fase de evaluación se verifica si se han cumplido los objetivos y metas derivados de la sección de análisis por medio de exámenes sobre la conducta de los estudiantes. La evaluación se realiza sobre un conjunto de individuos, revisando aspectos globales como velocidad de desempeño, incidencia de errores de concepto y retención en el tiempo de los mismos.

5.4.1 Fase 1: El Análisis

En esta sección se identifican cuatro tipos de análisis muy importantes :

a. Análisis de necesidades
b. Análisis de metas
c. Análisis de tareas
d. Análisis de contenido y contexto

5.4.1.1 Análisis de Necesidades

Una **necesidad** es una diferencia entre el estado actual y el deseado. El análisis de necesidades es una forma de determinar faltantes en el desempeño o en la productividad. El procedimiento a seguir cuando se hace un análisis de necesidades es el siguiente :

1. Observar y medir el estado actual.
2. Denotar un estado deseado
3. Las necesidades serán las diferencias entre los dos estados
4. Dar una prioridad a cada necesidad
5. Establecer qué necesidades se pueden satisfacer con los recursos disponibles (presupuesto, personal, equipo, medio ambiente, etc) y con qué prioridad.

5.4.1.2 Análisis de Metas

Una **meta** es un desempeño o comportamiento específico. El análisis de metas es importante porque las metas de cualquier empresa son en ocasiones vagas o demasiado generales. Para poder alcanzar una meta y asegurar, haberla logrado es necesario especificar el valor que deben alcanzar las variables que identifican una meta, por ejemplo, velocidad de ejecución, porcentaje de aciertos, tiempo de retención, tiempo de aprendizaje, etc). El procedimiento a seguir para realizar un análisis de metas es el siguiente:

1. Identificar y enunciar comportamientos que se relacionan con una meta.
2. Eliminar comportamientos que sean redundantes, demasiado generales y reenunciar los que no estén claros.
3. Convertir los enunciados en objetivos.
4. Verificar si la lista es suficiente para alcanzar las metas.
5. Producir un plan de aceptación, es decir, una descripción detallada de cuándo se ha logrado una meta.
6. Comunicar estos objetivos a las personas que los tienen que cumplir, en este caso, los estudiantes.

Sin embargo, antes de poder enlistar las metas de la IAC, es muy importante delimitarlas. Es decir, se debe decidir el papel de la IAC dentro del curso para poder delimitar sus metas. Shanna Schaefermeyer [SCHA90] menciona que la IAC puede tomar tres tipos de papeles que hacen variar su alcance e impacto en la educación, estos son:

1. **Adjunto**. Esta catagoría de IAC tiene un enfoque reducido y generalmente se limita a ensayar metodologías y presentar problemas de

limitada dificultad con muy poca supervisión. Este tipo de instrucción tiene un efecto realmente limitado en la enseñanza.

2. **Administración**. Los programas desarrollados que pertenecen a esta categoría, aunque sean parte importante del proceso enseñanza y aprendizaje, no pueden considerarse IAC propiamente dicha. Estos programas generalmente se dedican a mantener registros del aprendizaje individual, hacen análisis de resultados y evalúan la efectividad de la enseñanza.

3. **Curso Base**. Este tipo de programas tienen como objetivo el cubrir la mayoría de los temas de un curso buscándose que puedan, en lo posible, no sólo sustituir las funciones del maestro, sino mejorarlo. Estos programas son complejos y por lo general son tutores con cierto grado de inteligencia o simuladores con interfases interactivas y tal vez gráficas.

5.4.1.3 Análisis de Tareas

Determina la secuencia de comportamientos que llevan a la terminación de una tarea. **Sea una Tarea cualquier evento provocado por el usuario que sea significativo y que lleve a alcanzar alguna meta.**

En este momento es importante diferenciar entre las tareas de alto nivel que se tienen que realizar para alcanzar alguna de las metas mencionadas en el análisis correspondiente, como lo pueden ser los pasos para derivar un fórmula, la construcción de una molécula a partir de átomos elementales o la construcción de un algoritmo por medio de actividades elementales, etc. y las tareas computacionales de más bajo nivel que también son necesarias para lograr los objetivos. Estas últimas tareas pueden ser: accesar algún menú, alimentar algún comando, guardar y cargar archivos, etc.

Aunque los términos que a continuación se mencionan se discuten con mayor profundidad en el capítulo VI, se adelanta su presentación para mantener la claridad de la exposición y enfatizar la diferencia entre la interfase y la instrucción contenidos dentro del prográma.

En general el conocimiento que el usuario tiene de un sistema (y las acciones que puede realizar sobre éste) se divide en dos categorías; **Semántico y Sintáctico**. El conocimiento **semántico** es aquel que se relaciona con las tareas que tiene que realizar el individuo aún si no tuviera un programa computacional como herramienta de trabajo, es decir, ¿Qué deseo hacer?. El conocimiento **sintáctico** es aquél que depende del sistema computacional (programa y computadora). Se refiere principalmente a los comandos y su forma llamada sintaxis, por ejemplo: mnemónicos, menús y botones; es decir, ¿Cómo lo voy a hacer?. Las acciones que se realizan en el

sistema estarán divididas de acuerdo al nivel de conocimiento que se esté utilizando. Así, las tareas relacionadas con la instrucción pertenecen al nivel semántico y son a las que se refiere la discusión que sigue sobre análisis de tareas. Las tareas relacionadas con los comandos del programa pertenecen al nivel sintáctico y necesitan de un análisis independiente el cual se discute en el capítulo VI.

Para llevar a cabo un análisis de tareas se deben seguir los siguientes pasos:

1. Producir una organización jerárquica de objetivos.
2. Determinar qué es lo que se tiene que saber para alcanzar los objetivos.
3. Enlistar prerequisitos en forma jerárquica.
4. Establecer una secuencia para instrucción.

El fín último de la instrucción es ayudar al estudiante a migrar de lo que ya conoce (y sabe hacer) hacia lo que no conoce (y sabrá hacer cuando haya terminado la instrucción). Después de haber llevado a cabo el punto 2, el diseñador de IAC debe determinar las habilidades de más bajo nivel subordinadas a las encontradas en el paso 2, y verificar si éstas a su vez tienen prerequisitos, es decir, si el estudiante necesita poseer algunas otras habilidades. Si se siguen estos pasos, se llega a la identificación de habilidades con niveles de sofisticación descendentes. El análisis termina cuando se puede asumir que los estudiantes poseen las habilidades identificadas de más bajo nivel. El análisis produce un cuadro con forma de árbol de tareas que sirve como mapa a seguir por los diseñadores de IAC en la producción de instrucción. Primero se enseñan las habilidades de más bajo nivel, luego las siguientes y así hasta llegar a enseñar las tareas de más alto nivel semántico.

5.4.1.4 Análisis de Contenido y Contexto

Otro par de análisis muy importantes para el diseño de IAC son los de Contenido y Contexto. Estos son necesarios para mantener la enseñanza en comunión con los mecanismos internos del aprendizaje. De la sección 5.3 se aprendió que la información que almacena el cerebro se divide en conocimiento declarativo, procedural y contextual. De tal forma, que el identificar correctamente los conceptos y su clasificación ayuda a localizarlos dentro del esquema de la enseñanza y a estructurar correctamente la secuencia desarrollada en el análisis de tareas. A continuación se explican los pasos necesarios para llevar a cabo estos análisis:

Análisis de Contenido

- Definir conceptos críticos de la información
- Definir las relaciones entre éstos y su organización jerárquica.
- El contenido y secuencia de la información debe basarse en asociaciones internas del cerebro tanto como en las estructuras externas que las contienen.

Análisis de Contexto

- Definir el contexto para la utilización del conocimiento
- Definir problemas complejos asociados con el concepto
- Analizar la solución de estos problemas para identificar conceptos utilizados
- Agrupar conceptos de acuerdo a su utilización
- Organizar conceptos en una red asociativa

El analizar los problemas dentro del contexto, identificar aquellos utilizados y la organización de su utilización permite secuenciar la instrucción para mejorar tanto la adquisición de conocimiento como su utilización.

5.4.2 Fase 2: El Desarrollo

5.4.2.1 Modelo de diseño integral de la instrucción

Robert D. Tennyson, la misma persona que propone el Paradigma Cognocitivo como base para la enseñanza tecnológica propone un modelo de desarrollo de IAC [TENN91b] que se relaciona directamente con las estructuras mentales involucradas en el aprendizaje descritas en su paradigma cognocitivo. La tabla 3 muestra ésta relación.

5.4.2.1.1. Conocimiento

El paradigma cognocitivo (sección 5.3) describe cuatro formas en que el conocimiento puede ser almacenado en la memoria:

a. Declarativo
b. Procedural
c. Contextual
d. Complejidad cognocitiva

Componentes del modelo de desarrollo de IAC	Metas Educativas				
	Adquisición del conocimienyo		Utilización del conocimiento		
Conocimiento	Declarativo	Procedural	Contextual	Complejidad Cognocitiva	Sistema Total
Objetivos del Aprendizaje	Información Verbal	Habilidad Intelectual	Habilidad Contextual	Estrategia Cognocitiva	Proceso Creativo
Tiempo de Instrucción	10%	20%	25%	30%	15%
Estrategia de Enseñanza	Exposición	Práctica	Resol. de Problemas	Complejidad y Dinámica	Autodirec-cionamiento

Tabla 3 Relación entre las estructuras mentales que componen el aprendizaje, el Modelo de Desarollo de IAC y las estrategias de enseñanza.

Cada tipo de conocimiento requiere de un sistema de almacenamiento especializado. Así, Tennyson habla del Sistema Declarativo como un subsistema de memoria encargado de almacenar el conocimiento declarativo. Además de los cuatro ya mencianados, Tennyson agrega un quinto sistema que es cuando se fuerza a que todo el susbsistema de almacenamiento entre en acción, el Sistema Total.

5.4.2.1.2. Objetivos del aprendizaje

Los objetivos en la enseñanza deben estructurarse en una jerarquía con cinco componentes de acuerdo a las estructuras mentales que los soportan:

1. Información Verbal
2. Habilidades Intelectuales
3. Habilidades Contextuales
4. Estrategias Cognocitivas
5. Proceso Creativo

Información Verbal. Es el primer contacto que tiene el intelecto del estudiante con la enseñanza. Estos objetivos pretenden que compresión y apreciación de todos los conceptos y sus relaciones, reglas y principios por los que se rigen los objetos de estudios y demás hechos que sean necesarios para la enseñanza.

Habilidades Intelectuales. Estos objetivos son los que pretenden que el alumno domine las habilidades necesarias para la utilización de los conceptos, relaciones, reglas, principios y hechos que se presentaron de forma verbal.

Habilidades Contextuales. Se pretende la adquisición y formación de una base de conocimientos y sus métodos de acceso. La base de conocimientos es la estructura interna tomada por los esquemas (representaciones mentales) del conocimiento en su almacenamiento. Los métodos de acceso representan la parte contextual y son las estrategias de control que utilizan la base de conocimientos en actividades mentales como recordar, resolver problemas, utilizar el criterio, etc.

Estrategias Cognocitivas. Igualmente, se pretende un doble propósito. Primero, elaborar las estrategias del pensamiento que armarán al estudiante con más y mejor conocimiento contextual, es decir, conocimiento de complejidad cognocitiva. Y segundo, desarrollar habilidades cognocitivas de diferenciación e integración. Estas habilidades proveen de herramientas para utilizar y mejorar la base de conocimientos.

Procesos Creativos. Primero, desarrollar la habilidad de derivar conocimiento a partir del ya almacenado y del medio ambiente externo utilizando criterios uniformes ya establecidos. Segundo, desarrollar criterio. Es decir, que se debe no sólo encontrar la solución a los problemas con la poca información disponible, sino que se debe ser capaz de desarrollar el propio problema.

5.4.2.1.3. Tiempo de Instrucción

En la mayoría de los métodos de instrucción, la mayor parte del tiempo se gasta en la adquisición de conocimiento. Sin embargo, en el modelo integral, la mayor parte de tiempo es utilizada en la utilización del conocimiento. Esto permite que la instrucción no sólo conste de adquisición de conocimiento, sino también, de un mejoramiento constante en su utilización.

5.4.2.1.4 Estrategias de Enseñanza

El propósito del modelo es el ligar las estrategias de enseñanza directamente con los componentes mentales que almacenan el conocimiento. Así, en lugar se proponer estrategias específicas para todos los tipos de aprendizaje, se identifican catagorías de estrategias generales. Cada una de esas estrategias está compuesta por variables que pueden ser manipuladas de acuerdo a la situación. Estas estrategias son:

1. Estrategias de exposición
2. Estrategias de práctica
3. Estrategias orientadas a la solución de problemas
4. Estrategias de complejidad dinámica
5. Experiencias autodirigidas

Estrategias de Exposición. Representan las acciones encaminadas a enseñar conocimiento declarativo. Presenta el contexto de la información además de un marco mental para las abstracciones de la instrucción. Presenta la organización de la información, pero además intruduce el 'por qué' de la naturaleza teórica y 'cuándo' son apropiados los criterios formados por esa naturaleza. La exposición debe basarse en los conocimientos y esquemas ya existentes para la presentación de los nuevos conceptos. Después de haber introducido el concepto, se deben presentar las ideas, principios, reglas y hechos de tal forma que extiendan el conocimiento ya existente y ayuden a establecer nuevo conocimiento. Existen varias formas de Exposición, éstas son:

Etiquetamiento. Se introducen los conceptos discutiendo su origen, de esta menera el alumno evita la memorización de la terminología.

Definición. Ligar la nueva información con conocimientos ya existentes en la memoria permanente y presentar lo atributos críticos del concepto. La nueva información debe presentarse de tal forma que se relacione con el medio ambiente del individuo, es decir, deben utilizarse como base al nuevo conocimiento los esquemas mentales del estudiante de todo tipo, social, académicos, familiares, comunitarios, etc.

Ejemplo perfecto. Permite una comprensión sencilla del concepto y la formación de una representación mental (esquema) apropiada.

Ejemplos de exposición. Presentan aplicaciones variadas y diferentes de la información, mostrando también algunas variaciones en el contexto.

Ejemplos de trabajo. Estos ejemplos típicamente se desarrollan paso a paso con explicaciones. Es decir, en cada paso se debe mostrar que información se está utilizando, cómo se está utilizando y por qué se está utilizando. El objetivo principal de estos ejemplos es permitir al estudiante a percibir la aplicación de conceptos dentro de un cierto contexto.

Estrategias de práctica. El objetivo es que el alumno aprenda a utilizar el conocimiento correctamente en situaciones nuevas. Podemos compararlo con el típico caso: "Bueno, eso es con manzanas. Ahora hagámoslo con peras". Es muy importante que el sistema de instrucción supervise el

desempeño del estudiante muy de cerca para poder prevenir y corregir errores de concepto y procedurales.

Estrategias orientadas a la resolución de problemas. este tipo de estrategias ayuda al estudiante a obtener un conocimiento contextual completo. Se propone que un 25% de la instrucción sea dedicado a éste tipo de estrategias, como la utilización de simuladores orientados a solución de problemas. El propósito de la simulación es mejorar la organización y acceso de la información dentro de la base de conocimientos mental presentando problemas que requieran del estudiante el localizar y extraer el conocimiento adecuado para proponer una solución. Al utilizar una simulación para resolver problemas el estudiante pasa por los siguientes procesos:

a. Analiza el problema
b. Conceptualiza el problema
c. Define metas esécíficas para reolver el problema
d. Propone una o varias soluciones

La diferencia entre práctica y resolución de problemas por medio de simulaciones es que la práctica ayuda al estudiante a adquirir conocimiento procedural y la simulación le permite utilizarlo.

Estrategias de complejidad dinámica. Estas estrategias alientan al estudiante a desarrollar sus procesos de pensamiento mientras utiliza el conocimiento. Una de estas estrategias es la simulación de complejidad dinámica que extiende el formato de la simulación de resolución de problemas utilizando una técnica iterativa que muestre las consecuencias de las decisiones tomadas y que actualice la situación haciendo la siguiente iteración más compleja. El procedimiento a seguir sería el siguiente:

1. Presentar variables y condiciones iniciales
2. Capturar la solución del estudiante y verificarla
3. Establecer la siguiente iteración aumentando, borrando o cambiando variables y condiciones basándose en el esfuerzo acumulativo del alumno.
4. Las características de cada iteración deben variar de acuerdo a al aprendizaje individual de cada estudiante.

Experiencias Autodirigidas. La creatividad puede mejorarse utilizando métodos de instrucción que requieran de acciones innovativas. Los PE pueden proveer de un ambiente adecuado para la manipulación de nueva información obtenida del medio y para que permita al estudiante poder

dirigir su propio aprendizaje y así crear nuevo conocimiento. Nuevamente, los simuladores proveen tal ambiente cuando los estudiantes pueden visualizar los resultados de sus decisiones, pueden manipular directamente las variables de las que dependen esos resultados y pueden aprender a predecir el resultado de sus decisiones. Es aquí donde la visualización toma un papel importante dentro de la instrucción porque, conjuntamente con la manipulación directa y la animación propias de los simuladores interactivos, permite la acción de los procesos del pensamiento de más alto nivel y aumentar la creatividad del individuo.

5.4.3 Fase 3: La Evaluación

Por lo general, en el desarrollo de IAC se pone mucho énfasis el desarrollo de la instrucción, dejándose a un lado toda la secuencia de análisis previos o haciéndose de una forma superficial mientras se desarrolla la instrucción. Y lo mismo puede decirse de la evaluación. Cuando se ha terminado el desarrollo de la instrucción, en muchas ocasiones no se lleva una evaluación formal de la interfase y de la instrucción con la cual se pueden identificar y corrigir errores, que aún siendo muy sencillos pueden hacer que todo el sistema tenga una utilidad mínima.

La evaluación consiste en el simple hecho de verificar qué se han logrado las metas perseguidas. En la evaluación de programas educativos, se evalúa conjuntamente la IAC y la Interfase. Esto primeramente, porque el diseño de cada uno siempre debe tener en mente el diseño del otro. Y además, la evaluación se hace en base al cumplimiento de metas específicas escritas en la sección de análisis. Claramente, no es posible probar la eficiencia de la IAC sin la presencia de una Interfase porque ésta es la encargada de llevar la instrucción al estudiante. ¿Cómo es posible, por ejemplo, probar la efectividad de los procesos que alientan la creatividad del individuo sin calificar la efectividad de los simuladores integrados? Por ésto, la evaluación se discutirá a detalle en el capítulo VI, después de haber cubierto los temas relacionados con el diseño de interfases.

Al realizar una evaluación de la instrucción, aún a través de la interfase, debe identificarse cuales características se están evaluando a la instrucción y cuales a la interfase.

La figura 6 muestra que la evaluación es la última fase del desarrollo global de la instrucción. La evaluación consta de tres partes:

1. **Desarrollar evaluación formativa.** Esta evaluación se realiza partiendo de las metas y objetivos planteados en la sección de análisis.

2. **Revisión de la Instrucción.** Se enumeran las fallas, y se localizan puntos débiles de la instrucción. Y se fortalecen. De ser necesario, regresar a la sección de análisis.

3. **Diseñar y llevar a cabo la evaluación final.** Cuando se hayan reparado las fallas y faltas en la instrucción, se diseña la evaluación final basándose en las prioridades de cada uno de los objetivos de la sección de análisis, encontrando la eficiencia final de la instrucción.

5.5 Conclusión

Se ha tratado de demostrar que en el caso de los alumnos de programación básica, se ha encontrado que el problema es la sobrecarga cognocitiva. El objetivo de esta tesis es probar que un ambiente integrado, que aisle al estudiante de las tareas sintácticas de la computación como sintaxis de lenguajes, compilación de programa, edición de código, manejo de espacio en disco, etc., lo introduzca suavemente a los conceptos semánticos de la computación, como teclado, disco, monitor, ratón, archivos, programas, etc y le permita concentrarse en los conceptos semánticos del pensamiento algorítmico para la resolución de problemas. Además, para que este ambiente pueda tener un efecto en el aprendizaje, la instrucción global (incluyendo al maestro humano) debe segir una secuencia lógica de exposición de conocimientos, llevando al alumno de sus esquemas mentales, hasta niveles de pensamiento más elevados. Es decir, que la instrucción global debe iniciarse con exposiciones simples e introductorias, pasar por la elaboración de ejemplos y la resolución de problemas. Finalmente, debe contener simulaciones de las acciones donde el alumno pueda manipular las variables, tomar decisiones sobre estas variables, visualizar los resultados de sus decisiones, y D.M., con la práctica predecir los resultados de sus decisiones. Sería muy conveniente que el individuo creara conocimiento experto al enfrentar condiciones variantes en el medio integrado y los problemas a resolver.

En el desarrollo de herramientas para el aprendizaje es importante utilizar la mayor cantidad posible de técnicas desarrolladas por la teoría del conocimiento. No se pueden desarrollar programas educativos sin entender los procesos elementales del entendimiento. Esto se aplica inclusive a la instrucción no ayudada por computadora. En pocas palabras, no es posible corregir una situación incorrecta si no se sabe qué es lo que está mal, y no es posible resolver un problema si no se sabe de qué depende la solución. De igual manera, es importante saber cuándo se ha encontrado una solución al problema, de ahí la importancia de enlistar objetivos y establecer niveles de aceptación. Es por ésto que realizar los análisis que se han mencionado es escencial antes de avocarse al desarrollo de soluciones. Sin éstos análisis, lo único que resulta de la IAC es una gran pérdida de tiempo y dinero que sólamente puede perjudicar a los estudiantes que se supone se debe ayudar.

CAPITULO VI

DISEÑO DE INTERFASES GRAFICAS INTERACTIVAS

6.1 Introducción

Un programa educativo construído sobre un firme análisis de instrucción pero con una interfases mediocre e insensitiva al estudiante, jamás logrará los resultados que se esperan de él debido a que el usuario sentirá un rechazo natural hacia lo cotidiano y aburrido, o puede sentirse intimidado por la difilcutades que tiene que salvar para hacer que el sistema haga lo que él desea. En cambio, una instrucción simple con un análisis práctico, aunque no muy profundo, soportado con una interfase intuitiva y efectiva puede lograr resultados sorprendentes.

Una interfase debe en todo momento tomar en cuenta las necesidades y motivaciones del las personas que están haciendo uso del sistema. Esto en ocasiones puede resultar ser una tarea muy difícil de lograr. ¿Cómo se puede acoplar la necesidad que tienen los usuarios principiantes de constante retroalimentación sobre las posibles acciones a llevar a cabo y sus consecuencias o sobre los errores que han cometido y la manera de arreglarlos? ¿Cómo se balancea la claridad y brevedad de los títulos de menús y comandos con la correcta utilización del castellano o la sensibilidad del los individuos a quienes están dirigidos estos mensajes? ¿Cómo evitar al mismo tiempo que el usuario se aburra sin que se sienta incómodo por un sistema demasiado demandante? ¿Cómo balancear la automatización de tareas con el control maestro que debe tener el usuario sobre el sistema y el flujo de la instrucción? ¿Cómo construir un sistema poderoso y flexible pero al mismo tiempo fácil de utilizar con comandos sencillos?

Todas estas desiciones deben ser tomandas aún antes de comenzar a diseñar la interfase. La mejor forma de lidiar con el problema es partiendo de un análisis concienzudo que sistemáticamente tome en cuenta la presencia de un ser humano de un lado de la pantalla, y un sistema que tiene la obligación de ser funcional, flexible y confiable del otro lado. Finalmente, los errores derivados de la primeras etapas del diseño deben ser rápidamente identificados y corregidos, entre más rápido se logre esto más confiable, más barato y más consistente resultará el sistema.

6.2 La Creación de la Interfase

El desarrollo de la interfase es crucial para el éxito de un programa educativo. Si la interfase está diseñada pobremente puede provocar sentimientos de frustación, confusión y aburrimiento que anulan la importancia de la instrucción contenida en el programa. En

cambio una interfase bien diseñada, da al estudiante una sensación de éxito, avance, mejora y competencia, en general una satisfacción subjetiva conjunta al aprendizaje de conceptos. El éxito de un programa educativo no se fundamenta en promesas o en promociones rimbombantes, sino en una planificación muy cuidadosa, una gran sensitividad a las necesidades de los estudiantes, una cuidadosa atención a los detalles en la planeación y el desarrollo, muchas pruebas y mucho entusiasmo. Todo esto suena como mucho trabajo, ¿Para qué? ¿Qué es lo que persigue el diseñador de una interfase interactiva? Aunque cada quien, por causa de la individualidad humana persigue sus propias metas, es posible delinear en forma global los objetivos que se deben buscar al diseñar una interfase. Aquí se mencionan los más importantes:

1. **Funcionabilidad Correcta**. Es decir, que el programa lleve felizmente a cabo las tareas para las cuales fué diseñado. Un programa educativo puede tener como objetivo primordial el aprendizaje, pero además un programa educativo para el aprendizaje del Cálculo puede incluir dentro de sus comandos el encontrar el área bajo alguna curva, demostrar tal y cual concepto, mantener registros de cada alumno y asesorarlo en su aprendizaje, etc. Pero igualmente puede no incluir alguna de estas acciones por tal o cual razón. De ahí que un análisis de tareas sea primordial para delinear la funcionabilidad de un programa. En pocas palabras, ¿Qué va a hacer, y qué no va a hacer el software educativo?

2. **Confiabilidad y Disponibilidad**. Un sistema que pierde información, se inhibe con ciertos comandos, da resultados erróneos o no está disponible por largas temporadas jamás podrá ser exitoso no importa que tan bien esté diseñada la interfase. Para cubrir este aspecto es necesario hacer numerosas pruebas que ayuden a encontrar y corregir errores y asegurar la confiabilidad del programa. La disponibilidad de un programa depende del lugar donde vaya a ser utilizado y de las políticas de ese lugar, pero si depende del diseñador, debe asegurarse que el sistema siempre esté disponible para quien tenga derecho a usarlo.

3. **Satisfacción Subjetiva**. Aunque lograr este objetivo no es necesario para el éxito de un programa, como lo demuestran los primeros programas educativos, es muy deseable. La satisfacción subjetiva se logra cuando se toma en cuenta al alumno en todo momento; lo comandos son claros y sencillos de utilizar, los comandos son fáciles de aprender y memorizar, la información en pantalla es clara y las respuestas se despliegan rápidamente, los errores son poco comunes y su corrección es sencilla, el programa toma decisiones por el alumno cuando es pertinente y en general, qué tanto le gustó al estudiante utilizar el programa.

Para lograr estos objetivos, las interfase debe desarrollarse de una manera metódica, apoyándose en los resultados que se han obtenido en años de investigación. No

es sorprendente que los pasos para el desarrollo de una interfase, en su forma más general, sean los mismos que para el desarrollo de la instrucción, como seguramente también los son para el desarrollo muchas otras cosas importantes. Estos son:

1. Análisis
2. Desarrollo
3. Evaluación

6.2.1 Análisis

Hay dos elementos en el desarrollo de una interfase que deben ser analizados:

a. Las Tareas
b. El usuario

El Análisis de Tareas es muy importante debido a que ayuda a delimitar la funcionabilidad del programa. Es muy común que el análisis de tareas se haga ya que el desarrollo del programa va muy avanzado sólamente como complemento al diseño original o como requisito de la institución o departamento. En estas circunstancias, es posible que la funcionabilidad del programa no haya quedado bien delimitada provocando un tiempo de desarrollo mucho más largo y por lo tanto costoso; si un individuo no sabe a dónde va, seguramente tardará más tiempo en llegar. También es probable que algunas tareas indispensables no sean incluídas desde un principio y tengan que ser posteriormente agregadas, lo que comúnmente se llama un "parche" en el programa. Lo que es más, un análisis de tareas que no se hace previo a la implementación puede terminar con una definición de funcionabilidad muy alejada de la intención original porque las conveniencias de la implementación pueden influir en la objetividad del analista.

El Análisis del Usuario es importante porque permite definir las características del usuario medio, y a partir de esta definición, construir una imágen del sistema para el usuario. Es decir, que sabiendo cuales son las características del estudiante que va a utilizar el sistema, se puede tener una idea de la imágen mental que éste tendrá del programa, y estructurar los comandos y acciones del programa de acuerdo a lo que se espera del sistema. Se pueden identificar comandos que pueden confundir al usuario, conceptos que pueden no ser entendidos, acciones que lo intimiden, y se pueden encontrar elementos que motiven al usuario a utilizar el programa y sacarle el mejor provecho.

Ya se ha mencionado anteriormente que los connocimientos que el alumno tiene del sistema y las distintas tareas que puede realizar están clasificados en distintos niveles semánticos. Esto es importante para el análisis de tareas porque permite hacer una clasificación jerárquica y funcional de las tareas que es

indispensable para su caracterización. Para clasificar estos niveles se ha introducido una teoría de conceptualización de niveles sintácticos y semánticos.

6.2.1.1 Teoría de Clasificación del Conocimiento Sintáctica y Semántica

Conocimiento Sintáctico

El conocimiento sintáctico es aquel que tiene que ver con los detalles que dependen de los dispositivos que utiliza para comunicarse con el sistema. Ejemplos clásicos son la sintaxis de comandos como read, write, output, etc, que tienen el mismo propósito pero diferente sintaxis. También puede ser que el comando se proporcione por medio del teclado, pantalla sensitiva, ratón etc. Uno de los principales problemas que debe sobrellevarse es la arbitrariedad que se presenta en estos detalles de diseño que generalmente se consideran de menor importancia. Algunas acciones o tareas pueden ser difíciles de recordar porque los comandos que las invocan son arbitrarios. Ejemplo; una cadena larga de teclas o un submenú escondido bajo el nombre de "Misceláneos". Otro problema lo representa la dificultad de proveer de una estructura jerárquica o modular. Es decir, es posible que las tareas no estén agrupadas en su forma de invocación de acuerdo su función. Ejemplo típico es que para terminar una sesión por teminal se debe teclear "logout", mientras que para terminar una sesión con editor es posible que la secuencia sea "Ctrl-k". Finalmente otra dificultad se encuentra entre las diferentes plataformas computacionales. Por ejemplo, un usuario de un programa disponible en IBM puede estar acostumbrado a utilizar cierto conjunto de comandos (como las teclas de función o flechas de cursor), sin embargo, si esta persona intenta utilizar el programa en una máquina Machintosh, éstas teclas pueden no estar disponibles, y ni siquiera aparecer en el teclado. Un buen diseñador sabrá comprender éstas dificultades y facilitar lo más posible la sintaxis de las tareas, tal vez con ayudas en línea o con dispositivos innovadores como menús jerárquicos o botones con logotipos.

Conocimiento Semántico

El conocimiento semántico es aquél que está relacionado con los conceptos. Estos conceptos se encuentran almacenados en memoria permanente formando una estructura jerárquica que va desde los conceptos (y sus acciones relacionadas) más simples, pasando por estrategias de mediano nivel, hasta objetivos de alto nivel. La forma más común en que la gente trata de resolver problemas complejos es descomponerlos en subproblemas siguiendo una estructura jerárquica (como el refinamiento por pasos en programación), hasta que un subproblema sea manejable con los recursos que se tienen. El diseñador de interfases debe seguir el mismo procedimiento, dejando muy en claro la semántica de un objeto y las posibles acciones deben

ser explícitas. De tal forma, que los objetos y las acciones pueden ser identificados, dejando la sintaxis para el final.

Ya se mencionó que los objetivos de la instrucción están planteados en términos de conceptos semánticos. Sin embargo, los conceptos de la instrucción no son los únicos conceptos de la aplicación que pertenecen a la categoría semántica. Por ejemplo conceptos como copiar, borrar o mover, limpiar pantalla, cargar archivos, cambiar características de algún objeto, etc, son tareas claramente semánticas que están presentes en una gran variedad de aplicaciones y que no están relacionados con la instrucción.

6.2.1.2 Caracterización del usuario

Para poder tener una idea clara de las tareas que se llevarán a cabo en el sistema es muy importante que el diseñador entienda el usuario (en este caso, el estudiante) para quien se está diseñando la interfase. Debe entenderse que diferentes personas aprenden, piensan y resuelven problemas de diferentes maneras. Algunas personas prefieren tablas que grafos, otras prefieren palabras en lugar de números, otras imágenes en vez de números y palabras, etc. Es difícil saber qué fundamentos teóricos tienen las personas que van a utilizar el sistema y de ahí partir a definir nuevos conocimientos o alentarlos a aplicarlos en el programa.

Todos los diseños deben empezar con un entendimiento de los usuarios que son el objetivo del diseño: su edad, su sexo, su preparación académica, su cultura, su conocimiento de la tecnología, sus motivaciones, sus metas y su personalidad.

Todo esto puede afectar el diseño de una manera u otra. Por ejemplo, una mujer puede encontrar repulsión inconciente a la utilización de palabras agresivas en el lenguaje del programa como **Matar** (un proceso), **Cortar** (algún objeto) o **Abortar** (el programa). Un alumno que se sienta obligado a utilizar un programa, estará falto de motivación y su apreovechamiento será mucho menor. Una persona con muy escasa cultura puede sentirse cohibido con el solo hecho de acercarse a una computadora, el programa debe ser muy "amable", muy lento en su progreso y debe inspirar confianza.

En general, una forma adecuada de diferenciar los usuarios de cualquier sistema es separarlos en tres categorías:Novato, Intermedio y Avanzado.

Novato. Estos usuarios tienen muy poco conocimiento sintáctico de cómo usuar el sistema y probablemente muy poco conocimiento semántico sobre conceptos computacionales. Puede ser que tengan muy poco conocimiento de las tareas que deben llevar a cabo y peor aún, pueden

sentirse intimidados por la computadora. Para éstos usuarios, los programas se diseñan con un número limitado de tareas, con una sintaxis muy sencilla (probablemente a través de menús). Las posibilidades de error deben mantenerse mínimas para eliminar al fustración, la ansiedad y el miedo. Debe preveerse retroalimentación constante sobre tareas completamente terminadas, y en aquellas donde hayan cometido errores deben proporcionarse mensajes claros e instructivos. Los manuales deben estar cuidadosamente diseñados y de ser posible, debe existir la posibilidad de accesar ayuda y tutoriales en línea.

Intermedio. Estos usuarios tienen conocimiento semántico de la computadora y las tareas que debe llevar a cabo, pero puede tener poco conocimiento sintáctico. Para éstos usuarios, el programa debe proveer una manera de disminuir la carga sobre la memoria con una estructura de comandos, terminología y menús simple y consistente. Es decir, el usuario no recuerda la sintaxis de un comando, pero sabe donde encontrarla. Puede existir un menú que se la recuerde o incluso que invoque el comando por él. También es deseable que la cantidad de errores posibles sea pequeña de tal forma que aliente la exploración o que se intente invocar comandos cuando la sintaxis ha sido parcialemente olvidada.

Avanzado. Estos usuarios tienen total dominio de la sintaxis y la semántica del sistema y demandan más de él. Los tiempo de respuesta deben ser cortos, los mensajes de error deben ser breves para evitar la distracción, y se espera que la mayoría de los comandos puedan ser accesibles con combinaciones rápidas del teclado.

6.2.1.3 Caracterización de Tareas

Después de haber tipificado cuidadosamente el perfil del usuario, deben identificarse las tareas que se van a llevar a cabo. Parte de éste procedimiento en su más alto nivel fué discutido en el capítulo anterior, ahora se discutirá todo el proceso completo de análisis de tareas.

Es muy raro que se diseñe e implemente un sistema cuyo objetivo sea hacer cosas que nunca se habían hecho. La mayoría de los programas computacionales son creados con la intención de realizar tareas que ya se hacen, implementando una tecnología que dé alguna ventaja sobre la tecnología anterior. Tal vez el nuevo sistema sea más rápido, más barato, más exacto, maneja mayores volúmenes. etc. De esta manera, se puede estar seguros que un análisis de tareas debe empezar análizando las que ya se realizan en el sistema que se piensa sustituir. Aunque también es común que se desee utilizar la nueva tecnología para hacer cosas adicionales, en

ocasiones muy diferentes a la intención original del diseñador, por lo que éste debe intentar predecir estos usos alternativos.

Cualquier diseñador estará de acuerdo que un conjunto de tareas debe ser siempre delineado antes de comenzar la implementación o continuar el diseño. Pero la mayoría de las veces la caraterización de tales tareas se hace de una manera informal o implícitamente. El diseñador tiene una idea mental de los comandos que se van a implementar, en ocasiones inclompleta o incluyendo comandos adicionales sólo porque a lo mejor el usuario los puede encontrar útilies, y con ella procede en la implementación del sistema, esto es a todas luces incorrecto. Las conveniencias de la implementación no deben influenciar la funcionabilidad de un sistema ni sus comandos.

En general, un buen enfoque es de tomar las tareas del más alto nivel semántico (calcular área bajo la curva, derivar matemáticamente la ecuación de algún método numérico, crear un diagrama de flujo, etc) y descomponerlas en tareas de mediano nivel, refinar o descomponer éstas en tareas de nivel inferior y así hasta llegar a las tareas atómicas. las tareas **Atómicas** son aquellas que el usuario puede realizar con un sólo comando, selección de menú, boton, etc.

Escoger el conjunto de tareas atómicas más apropiado puede ser muy difícil. Si las acciones atómicas son demasiado básicas, el usuario puede sentirse frustrado por la cantidad de ellas que tiene que llevar a cabo para completar una tarea de alto nivel. Si las tareas atómicas son demasiado elaboradas y complejas, el usuario tendrá que memorizar una serie de opciones en cada comando, que son fáciles de olvidar, y pueden llevar a que contínuamente no se obtenga lo que se quiere.

Pero, ¿Qué es una tarea? **Una tarea no es más que una acción sobre algún objeto, provocada por el usuario que lleve a algún fin.** Esta definición que parece trivial es muy importante, porque de ésta se pueden identificar al menos dos elementos que componen una tarea: una acción y un objeto sobre el cual se realiza la acción. El objeto indica el nivel semántico de la tarea, mientras que la sintaxis depende totalmente de la acción y del programador. Además de esto, la acción será la llave para clasificar la tarea y de ésta forma el conjunto de comandos de que dispone el sistema podrá tener una estructura y hasta una jerarquía.

Otra característica importante es la frecuencia del comando. Comandos que invoquen tareas atómicas frecuentemente utilizadas deberán ser simples, su acceso debe ser inmediato y su ejecución rápida, aún aunque esto signifique que tareas mucho menos frecuentes tengan un acceso menos inmediato y una ejecución mucho más lenta.

Adicionalmente a éstas características Donald Norman [RUBI88] propone un modelo de tarea con siete aspectos:

1. Meta. Una meta significa lo que el usuario quiere.
2. Mapeo de la meta a una Secuencias de Acciones. El usuario debe traducir sus intenciones al estado deseado del sistema y de ahí determinar la secuencia de acciones físicas a realizar.
3. Especificación de la secuencia de acciones. Aquí el usuario construye una representación mental de dicha secuencia y la ensaya tratando de determinar el resultado.
4. El Estado del Sistema. Indica que valores contienen las variables en ese instante.
5. Mecanismos de Control. Los dispositivos a disposición del usuario que le permiten alterar el estado del sistema.
6. Mapeo entre mecanismos y el estado del sistema. Cómo es que una manipulación en un dispositivo altera el estado del sistema.
7. Interpretación del estado del sistema. Aquí percibe el sistema y lo interpreta en téminos de variables de interés.
8. Evaluación del Resultado. Se realiza una comparación entre el estado del sistema que se percibe y el estado que se desea. Si ambos estados no son idénticos, entonces se genera un nuevo conjunto de metas y el proceso comienza otra vez.

De esta definición existen dos conceptos que son muy relevantes: el estado de un sistema, y la secuencia de acciones que lo alteran. En estos elementos se basa un modelo gráfico de tareas denominado **Diagramas de Transiciones de Estado**. Estos diagramas no son más que un autómata (ver fig 7) donde cada nodo representa un estado del sistema. Los vectores dirigidos representan las posibles acciones que se originan en un estado y llevan a otro. Estos diagramas son una gran ayuda durante el desarrollo porque permiten encontrar callejones sin salida dentro del sistema y rutas mínimas. Cuando presentados al usuario, son excelentes en la capacitación para la utilización del programa y ayudan a predecir tiempos de aprendizaje, tiempos de desempeño de tareas y posibilidad de errores. Además, los autómatas son representaciones matemáticas que abren todo un campo de investigación en las interfases interactivas que bien podría denominarse **Formalización Matemática de Intefases Interactivas**.

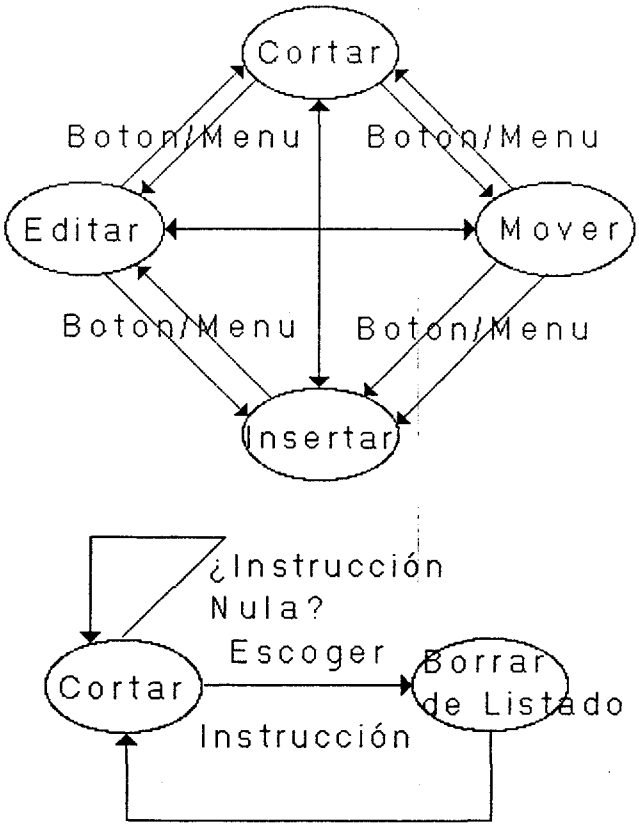


Fig. 7 Diagrama de Estados
de Modo Editar

6.2.2 Desarrollo de La Interfase: Diseño e Implementación

Existe una tendencia generalizada a tomar la etapa de diseño e implementación como la única etapa en la creación de software. Generalmente las etapas de Análisis y Evaluación son hechas a un lado por la gente encargada del desarrollo. Por esto, es sorprendente que sea tan poco común la utilización de técnicas adecuadas de desarrollo. Es decir, no sólamente se minimiza el papel del análisis y la evaluación, sino que se tiene la idea entre los programadores no expertos (como seguido lo son las presonas que desarrollan programas educativos), de que el desarrollo de software consiste en sentarse a programar. Esta es una idea a todas luces equivocadas. Sin embargo, debe admitirse que el diseño es un proceso de naturaleza creativo e impredecible. Los diseñadores de sistemas educacionales interactivos deben mezclar un profundo conocimiento del tema, una experiencia sobre la factibilidad de implementar adecuadamente las técnicas computacionales, una sensibilidad hacia las necesidades de los futuros usuarios y una estética intuititva. El diseño fué caracterizado de la siguiente manera por Carroll & Rosson [SHNE87]:

1. Es un proceso; no es un estado que puede ser representado estáticamente
2. No es jerárquico; no se puede describir como un proceso de arriba a abajo ni como abajo hacia arriba.
3. Sufre cambios radicales; involucra el desarrollo de soluciones parciales que pueden no tener ningún papel en el producto final.
4. El desarrollo constantemente lleva al descubrimiento de nuevas metas.

No obstante esto, existen varias técnicas que pueden ayudar a ahorrar tiempo y esfuerzo en la implementación de cualquier programa [KEAR85]. Estas son:

1. Especificación Funcional y de Programación
2. Prototipos
3. Pruebas Piloto

6.2.2.1 Especificación Funcional y de Programación

Los dos tipos de especificación que se describen a continuación, deben contar con el apoyo de análisis de tareas y de una caraterización del usuario cuidadosos. Su principal función es crear directrices de diseño que permitan al programador concentrarse en la codificación de las características del programa sabiendo de antemano en qué consisten estas características. Al terminar estas especificaciones el programador sabe exáctamente qué es lo que tiene que hacer, evitándose la distracción de reconstruir la imágen mental que tiene del programa cada vez que se le ocurre una nueva idea.

Especificación Funcional. Explica la estructura global del programa y su operación en téminos entradas, procesos y los elementos de las salidas. Describe lo que se asume originalmente del usuario a quien está dirigido el programa. También deben especificarse las características del hardware y medidas de desempeño aceptables. Una especificación funcional puede incluso incluir un diagrama donde se muestra las interrelaciones entre las funciones mínimas a implementar y los datos con los que trabajan. La especificación funcional está dirigida al diseñador y muestra cómo va a funcionar el programa independientemente de su implementación. Ayuda también a revisar el programa y hacerle cambios antes de que sea codificado.

Especificación de Programación. Este tipo de especificación es mucho más detallada porque incluye descripciones de los formatos de pantalla, secuencias de entradas y salidas, archivos de datos e incluso la lógica para alterar el flujo de la ejecución del programa. Esta lógica puede ser representada en diagramas de flujo o de manera algorítmica. Este tipo de especificación está dirigida al programador y le ayuda a implementar el programa.

6.2.2.2 Prototipos

El objetivo de los prototipos es el proveer de una descripción detallada de los aspectos dinámicos e interactivos del programa. Estos aspectos son muy difíciles de visualizar sólamente de las especificaciones. Un prototipo es una parte del programa final que tiene implementadas, con el mínimo esfuerzo, las secuencias de pantallas y de interacción más importantes del programa. En el prototipo sólo funciona un conjunto muy reducido de funciones, las cuales incluso, pueden no funcionar de la manera que lo harán cuando el modelo esté terminado. Por ejemplo, algunas pantallas pueden desplegar cálculos no calculados sino leídos de un archivo, las animaciones pueden ser burdas y en pasos discretos, muchas opciones de menú pueden estar bloqueadas o puede ser que sólo se desplieguen algunos mensajes de error, etc.

Además de mostrar cómo se verá y sentirá el programa una vez que esté terminado, otra función de los prototipos será verificar qué tan difícil será el programar algunas secuencias, sobre todo si las acciones dentro de estas secuencias no se han hecho anteriormente. También ayudan a verificar la estructura y accesibilidad de los comandos. En general, las funciones que se deben incluir en un prototipo son aquellas que son o muy comunes o muy poco comunes. Además, el prototipo debe incluir a lo más el 10% del total del programa.

6.2.2.3 Pruebas Piloto

Las pruebas piloto son pruebas de la vida real en las que todas o algunas funciones completas del programa se prueban con futuros usuarios del sistema. La intención es encontrar errores innevitables en todos los sistemas como respuestas inesperadas, instrucciones ambiguas, explicaciones y mensajes de error que no son claros, etc. Aunque las pruebas piloto son más de la fase de la evaluación final, es recomendable que también se hagan en secciones que tengan cierta independencia dentro del mismo programa, para detectar la presencencia de estos errores y corregirlos desde las primeras fases del desarrollo cuando el código del programa es todavía pequeño y el encontrar y corregir esos errores es más fácil.

Las formas de llevar a cabo las pruebas son muy variadas. Pueden hacerse por observación directa donde el evaluador se sienta al lado del usuario y observa los problemas que van ocurriendo. Puede utilizarse una video grabadora para analizar más cuidadosamente las acciones que llevaron al error. Puede ser que se deje trabajar a un grupo de usuarios por un rato y después que éstos llenen hojas de cuestionario que incluyan una sección de comentarios.

El realizar pruebas pilotos de una manera u otra siempre provee de información muy valiosa. Entre más pruebas se hagan mucho mejor. Así, las pruebas piloto deben hacerse en todo el tiempo del desarrollo y evaluación hasta que se acabe el tiempo límite o el presupuesto.

En cuanto a qué aspectos cambiar y cuándo hacerlo, existe una regla de dedo: Si una persona sugiere un cambio, hacerlo si a todas vistas parece una mejora mayor; considerese muy seriamente cambios sugeridos por dos personas diferentes; y siempre lleve a cabo cambios sugeridos por más de dos personas independientemente de lo que ud. piense.

6.2.2.4 Diseño Participativo

Esto se refiere a permitir la participación del usuario final del sistema en el diseño e implementación de éste. Los argumentos a favor de tal participación son los siguientes:

1. Mayor precisión en la información sobre tareas y su desempeño
2. Oportunidad para discutir sobre desiciones de diseño
3. Sensación de participación que permite el desarrollo de orgullo por parte del usuario
4. Potencial para una mayor aceptación del sistema por parte del usuario

En cambio, se han señalado algunas desventajas:

1. La participación extensiva del usuario hace el desarrollo más lento y costoso
2. Pueden desarrollarse resentimientos entre aquellos usuarios que no fueron incluídos
3. Pueden desarrollarse antagonismos con aquellas personas cuyas sugerencias fueron rechazadas
4. Se puede forzar a los diseñadores a comprometer su diseño en aras de satisfacer a algún participante incompetente o influyente
5. Algunas personas que se van afectadas por la implemetación del sistema pueden intentar boicotear la funcionalidad de éste

Al final de cuentas, el encargado del proyecto debe tener la suficiente sensibilidad para sopesar el ambiente político y social que rodea el proyecto y determinar el nivel de participación en el diseño.

6.2.3 Evaluación

La evaluación de un sistema no es algo que necesariamente se debe hacer al final de la implementación del sistema. En términos de tiempo, esfuerzo y dinero, entre más tiempo se demore un cambio necesario, más cuesta el llevarlo a cabo. Los productores teatrales han aprendido que se requiere una gran cantidad de ensayos para asegurar un estreno exitoso. Los primeros ensayos serán con dos o tres actores con ropa normal, pero finalmente se llevarán a cabo con toda la vestimenta de la obra y todos los actores. La correspondiente analogía entre la implementeación de una obra teatral y una interfase ya se ha mencionado: La utilización de prototipos y pruebas pilotos. Esto implica que los procesos de desarrollo y evaluación deben alternarse y complementarse mutuamente. Si además se toma en cuenta que el diseño y la implementaciόm implican descubrimiento de nuevas metas y tal vez la modificación de las ya existentes, se puede ver que la creación de un sistema interactivo no es un proceso totalmente lineal, sino más bien un estira y afloja entre análisis, diseño, implementación, evaluación, análisis de los resultados parciales, rediseño, reimplementación, reevaluación, etc.

Fuera del parecido que a través de los ensayos se puede encontrar entre la implementación de una interfase y una obra teatral, ambos tienen que enfrentarse a la evaluación última; la obra teatral a la revisión de los críticos, la interfase al la evaluación subjetiva y funcional del usuario. Las obras teatrales no se reimplementan cuando una crítica ha sido negativa, simplemente fracasan. Los sitemas interactivos se rediseñan.

Existen pricipalmente tres formas en que se puede evaluar un sistema interactivo, educacional o no:

1. Pruebas de aceptación mínimas
2. Encuestas
3. Entrevistas y Discusiones

6.2.3.1 Pruebas de Aceptación Mínimas

La primera prueba a la que se somete generalmente un sistema es la verificación de cumplimiento mínimo de especificaciones en la sección de análisis. Para que estas pruebas puedan llevarse a cabo o tengan alguna validez, los criterios mínimos deben haberse hecho por escrito antes de comenzar con el desarrollo. Tales criterios son los siguientes:

1. Tiempo de aprendizaje por función
2. Velocidad de desempeño por tarea
3. Porcentaje de errores cometidos por usuario
4. Satisfacción subjetiva del usuario

5. Retención de formato de comandos en el tiempo

Un texto que describa un exámen de acepatación mínima podría leerse de la siguiente manera:

30 usuarios típicos serán entrenados para utilizar el sistema 45 minutos. A estos usuarios se les darán 15 minutos para terminar las pruebas que se describen a continuación. El 80% de las personas debe finalizar sus tareas dentro del tiempo de terminación establecido y el número de errores debe ser menor que 3.

6.2.3.2 Encuestas

Las encuestas son una forma barata de determinar la productividad de un usuario con el sistema. La encuesta debe ser preparada, y revisada entre colegas. Antes de realizar una encuesta a gran escala, debe primero probarse con una pequeña muestra de usuarios.

Las encuestas pueden diseñarse en línea, ahorran costos de impresión y el esfuerzo de distribuir y recolectar. La mayoría de las personas prefieren, contestar una serie de preguntas breves en pantalla que llenar un forma impresa y entregarla. Si la población de usuarios es grande, se puede aplicar la encuesta a sólo un fracción de los usuarios.

Por lo general, las encuestas se construyen en forma de frases que el usuario tiene que calificar de una forma gradual, por ejemplo:

1. Los comandos del sistema son fáciles de usar.
2. Cuando se construye un nuevo conjunto de comandos para una nueva aplicación del sistema, estoy seguro que se desempeñarán correctamente a la primera corrida.
3. Cuando recibo un mensaje de error, encuentro que es útil para identificar el problema.
4. Siento que los comandos pueden simplificarse substancialemente.

Las respuestas podrían ser del siguiente tipo:

De acuerdo	1 2 3 4 5	En desacuerdo
Hostil	1 2 3 4 5	Amigable
Vago	1 2 3 4 5	Específico
Dealentador	1 2 3 4 5	Alentador

Cuando se incluyen preguntas precisas en la encuesta la posibilidad de obtener información valiosa de los resultados de la encuesta es mayor.

6.2.3.3 Entrevistas y Discusiones

Entrevistarse con los usuarios es muy productivo porque se pueden tratar directamente detalles particulares sobre el desempeño de la interfase. El entrevistar usuarios puede resultar caro y llevarse mucho tiempo, por eso generalmente sólo se lleva a cabo con una fracción de los usuarios. Sin embargo el esfuerzo vale la pena porque el contacto directo con los usuarios produce sugerencias muy constructivas.

Después de recopilar los resultados de una serie de entrevistas, una discusión en grupo pone en claro que comentarios forman parte del sentir general y por lo tanto demandan mayor atención.

6.3 Factores Humanos en el Diseño de Interfases

6.3.1 Diferencias entre usuarios

Es indudable que el crear un concepto de usuario promedio es de gran ayuda en los análisis previos a la codificación del programa, porque permite delinear la funcionabilidad del programa y crear un retrato aproximado del usuario hipotético. Sin embargo, el crear la interfase teniendo sólamente en cuenta a este usuario puede hacer un gran porcentaje de los usuarios reales se sientan incómodos por ciertas características que se ven obligados a enfrentar. Estas características pueden ser tan sutiles como la excesiva conglomeración de imágenes en la pantalla, el rápido despliegue de información en ésta, la organización poco intuitiva de la información en pantalla, etc. El problema con este tipo de situaciones es que dependen de una apreciación totalmente subjetiva de un solo individuo, que a su vez, depende de su perfil psicológico, su educación, sus costumbres, etc. Por esto, el diseñar una interfase teniendo en cuenta la diversidad de usuarios siempre es un reto.

6.3.1.1 Diferencias Hombre Mujer

Una de la principales diferencias que se pueden encontrar es aquella entre usuarios hombres y usuarios mujeres. Se ha mencionado muchas veces que la mayoría de los jóvenes que asisten a los salones de juegos de video son hombres, además de que los diseñadores también lo son. Las pocas mujeres que asisten a tales salones prefieren juegos como PacMan, Donkey Kong y otros que en total forman un conjunto reducido. Esto se debe a que muchas de ellas consideran otros juegos simplemente agresivos, oral y visualmente. En los juegos que sí disfrutan, se pueden identificar ciertas características, algunas no muy sorprendentes. Además de poco violentos, son juegos en los que el área de juego está completamente visible, los personajes tienen cierta

personalidad, los colores son más suaves (aunque no necesariamente rosa), y siempre existe la sensación de haber terminado el juego en alguna etapa. Una dato curioso de algunas mujeres que juegan PacMan, es que encuentran motivante el limpiar todos y cada uno de los puntos que aparecen en pantalla. En otras cuestiones no vinculadas con los juegos de video, Una mujer puede sentirse incómoda cuando tenga la necesidad de "Matar"[1] un proceso, "Abortar" un programa, "Cortar" la mitad del trabajo realizado en toda una tarde, etc.

6.3.1.2 Tipos de Personalidad

Este tipo de detalles que alejan al usuario de un programa pueden evitarse si se pusiera un poco más de atención a las diferencias individuales entre usuarios. Sin embargo, encontrar esas diferencias implica definir todo un perfil psicomotriz de los posibles usuarios, lo cual es difícil. Una ténica muy común para realizar esta caracterización es utlizar el modelo de Indicador de Tipos de Myers y Briggs [SHNE87] que se basa en las teorías de tipos de personalidad de Carl Jung. Este modelo, enumera cuatro tipos básicos de diferencias en personalidad, llamadas dicotomías:

Introvertido vs Extrovertido. Las presonas extrovertidas gustan de estímulos externos y disfrutan la variedad. Las personas introvertidas prefieren patrones familiares, confían en sus ideas internas y les gusta trabajar solos.

Sensitivo vs Intuitivo. Las personas sensitivas prefieren rutinas preestablecidas, aplicar habilidades bien dominadas y son buenos para el trabajo que requiere de presición. Las personas intuitivas disfrutan de resolver problemas nuevos encontrando nuevas relaciones y características del ambiente de trabajo, pero no gusta detenerse a hacer las cosas con presición.

Perceptivo vs Juzgativo. A las personas perceptivas les gusta aprender situaciones nuevas pero pueden tener problemas para tomar decisiones. Las personas juzgativas prefieren hacer planes detallados y siempre buscan ajustarse a ese plan incluso si nuevas situaciones pueden cambiar la meta original.

Emocional vs Pensante. Las personas emocionales toman en cuenta los sentimientos de otras personas y se llevan bien con la mayoría de quien los rodea. Las personas pensantes pueden demostrar pocas emociones, tratan a la gente de forma impersonal, les gusta arreglar todo de maneras lógicas.

---

[1] Cabe notar que el comando "KILL", que se utiliza mucho en programas escritos por gente de lengua inglesa, jamás se usa en lengua castellana, donde su contraparte sería "MATAR"

Además de estas dicotomías básicas, se han desarrollado varios cientos de escalas psicológicas, pero una persona que no sea psicólogo difícilmente puede diseñar una interfase tomando encuenta tanta variedad de usuarios. Otra perspectiva de diferencias entre usuarios tiene que ver con su cultura, grupo étnico, lengua, etc.

6.3.2 Bloqueos Psicológicos

Otro aspecto que debe cuidarse al diseñar una interfase, es que el sistema debe evitar que el usuario caiga en un bloqueo psicológico que puede causar que se rompa la comunicación hombre máquina y el sistema no sea explotado en su máxima capacidad. Estos bloqueos son típicamente cuatro [FOLE74]:

6.3.2.1 Aburrimiento

El aburrimiento es el resultado de la mala utilización de los tiempos de presentación. El usuario siente que el avance del programa es muy lento. Se puede pensar en el aburrimiento como una consecuencia normal de programas de presentación largos, pero la evidencia muestra que no necesariamente tiene que ser así. El aburrimiento sólo se presenta cuando el usuario no interactúa con el sistema, es decir, que el sistema toma su propio camino y no demanda la atención del usuario por medio de constantes peticiones de datos o preguntas dirigidas.

En ocasiones, tiempos de respuesta demasiados prolongados llevan al usuario al aburrimiento e incluso a la frustración. Debido a que el tiempo de respuesta depende de la máquina en que se esté corriendo y los algoritmos que se utilicen, es de vital importancia que el diseñador siempre optimice el código y los algoritmos cuando éstos influyan en los tiempos de respuesta, para mantenerlos dentro de los límites tolerables. Sin embargo, no todas las personas y no todas las acciones de estas personas requieren los mismos tiempos de respuesta. R. Miller [BEAT82] sugiere la existencia de una jerarquía de tiempos de respuesta de eventos de acuerdo al nivel mental al que pertenezca la acción generadora. Entre más bajo sea el nivel de la acción, el tiempo de respuesta que se espera es menor porque la acción generadora necesitó de un menor esfuerzo para realizarse, y por lo tanto, la sensación de "culminación" (haber terminado una tarea importante) es menor. En general, el usuario espera un tiempo de respuesta proporcional al esfuerzo que él haría para realizar la acción similar.

Estos niveles mentales son tres: Léxico, Sintáctico y Semántico. Anteriormente ya se había mencionado un modelo de acciones y conceptos de usuario denominado Sintáctico y Semántico. Aunque la idea es la misma, las

definiciones entre este modelo y el de Miller pueden no coincidir. Mientras que el modelo Sintáctico y Semántico se refiere a la representación mental de conceptos y sus acciones correspondiantes por parte del usuario, el modelo de Miller es una jeraquización del esfuerzo mental y el nivel de conciencia que se alcanza cuando se realiza una tarea.

El nivel léxico corresponde a aquellas cosas que son hechas por reflejo, ya sea natural o derivado del entrenamiento. Una persona normal tarda al rededor de 50ms para llevar a cabo estas acciones; más o menos lo que tarda un mecanógrafo en oprimir dos teclas. El tiempo de respuesta debe ser aproximadamente el mismo porque la psicología humana no tolera interrupciones de acciones reflejas.

En nivel sintáctico se encuentran aquellas acciones seminconcientes con las cuales se construyen pensamientos y frases completas. Por ejemplo, "coloca un transistor entre el primer y el segundo nodo" sería una construcción mucho más conciente que cualquier acción léxica porque el usuario construyó una idea. Los tiempos de respuesta son más variables en estos casos. Medio segundo es lo óptimo, pero retrasos de dos a cuatro segundos son tolerados.

Las acciones semánticas son aquellas en las que se esperan respuestas profundas. Por ejemplo, se puede pedir a un sistema que encuentre la ruta mínima entre una serie de nodos de un grafo, encontrar el enlace con la menor energía en una molécula compleja, o tal vez se pida una simulación dinámica de una planta. Los tiempos de respuesta en estos casos son muy variables. Pueden pasar hasta 10 s sin que el usuario pierda su concentración, e incluso, puede utilizar el tiempo para planear su próxima acción.

6.3.2.2 Pánico

El pánico es la consecuencia de permitir que los tiempos de respuesta se alarguen demasiado. El usuario empieza a preguntarse, ¿Existirá algún problema?, ¿se habrá inhibido la máquina?, ¿Estará el programa defectuoso?, ¿Estaré perdiendo mi información? La solución, si no se puede reducir el retraso de ninguna menera, es introducir un pequeño desplegado donde se de confianza al usuario o información. Lo óptimo es que se vaya indicando concurrentemente el porcentaje de avance de la acción, pero cualquier tipo de acción sirve. Otro ejemplo puede ser la aparición de puntos que indican el avance, así si el usuario ve muchos puntos, sabe que la tarea es muy larga. Si se trata de una búsqueda o un cálculo, se pueden ir desplegando resultados parciales conforme se vayan obteniendo. El mínimo aceptable es una indicación de que el comando fué aceptado, una indicación de que la acción se

ha iniciado, y finalmente algún mensaje que indique que la tarea se ha completado.

6.3.2.3 Frustración

El usuario es víctima de la frustración cuando no puede llevar a cabo sus intenciones exactamente. Es decir, que el usuario quiere hacer algo, se lo indica al sistema, pero éste proporciona un resultado diferente. La frustración se declara francamente cuando el usuario de ninguna manera puede comunicar sus intenciones al sistema, o constantemente se están cometiendo errores. Por último, la frustración se maximiza cuando es difícil deshacer estas acciones.

Se puede ver que la frustración es causada por dos tipos de sistemas:

1. Sistemas que no proporcionan la suficiente cantidad de opciones al usuario
2. Sistemas que no perdonan errores del usuario

El primer caso es un sistema cuya funcionabilidad no está correctamente especificada y necesita de nuevas funciones. Por ejemplo, el usuario desea insertar una bomba después del equipo C pero la acción de insertado sólo se lo permite hacerlo antes del equipo.

El segundo caso es un sistema cuya recuperación de errores está pobremente instalada. El usuario no sólo encuentra que no puede cancelar una acción antes de que se lleve a cabo, sino que para regresar al estado anterior tiene que seguir por un procedimiento tortuoso. Si los errores son frecuentes, aparte del mecanismo para corregir errores rápidamente (como un comando "Deshacer"), es conveniente incluir un paso adicional antes de llevar a cabo la acción en el que el usuario debe pensar doblemente si realmente desea hacer lo que está indicando.

6.3.2.4 Confusión

La confusión aparece cuando al usuario le es difícil identificar diferencias entre comandos o entre elementos de la pantalla. La solución consiste en organizar los comandos en una estructura jerárquica, presentando los comandos y sus opciones en forma secuencial conforme se vaya recorriendo la estructura. Se puede eliminar confusión causada por exceso de detalle en la pantalla agrupando los objetos visibles de acuerdo a su función y eliminando información innecesaria. Y finalmente se puede recurrir a la utilización del cursor físico y virtual, iluminar teclas de función representadas en pantalla, representando acciones de oprimido en botones, cambiar el cursor de acuerdo a la utilización y tipo de información que se está pidiendo, tijeras

para cortar, cruz para posiciones en pantalla, línea parpadeante para entrada de texto, etc.

6.4 Conclusión

El diseño de una interfase computacional, requiere de tomar en cuenta una gran cantidad de consideraciones. Las que se han planteado hasta aquí, son sólo una pequeña parte del caudal de investigación que se ha hecho en esta área. A todas luces, cada uno de los temas recién tratados tiene cabida y relación con la construcción de programas educativos. Un alumno, como cualquier otro usuario, está propenso an sentirse incómodo con comandos ofensivos, a confundirse con comandos demasiados misteriosos o a aburrirse con exposiciones largas y monótonas. Aún más, en lo referente a alumnos de programción básica, debe hacerse un hincapié adicional en la construcción de la interfase debido a que esta será la encargada de introducir a gente inexperta a la tecnología de las computadoras y por lo tanto será parte de esa primera impresión tan permanente en la memoria de las personas.

Los programas educacionales dirigidos a gente que apenas se está adiestrando en los temas presentados, requerirán de mensajes de error significativos y útiles, de una adecuada planeación de tareas que no le permita cometer errores que le hagan perder horas de trabajo, que si comete errores, que éstos puedan ser recuperados fácilmente, si no es que automáticamente, de tal forma que pueda aprender de sus errores y no exasperarse de ellos. Las interfases deberán proveer de mecanismos que las hagan al mismo tiempo intuitivas y que consistentes, deberán ser innovadoras pero nunca realizar acciones sorprendentes que confundan al alumno o lo frustren. En definitiva, de la calidad y claridad del diálogo que se derive de la interfase computacional de un programa educativo, puede depender la diferencia entre un sistema muy bueno pero fracasado y un sistema práctico y muy exitoso.

CAPITULO VII

CARACTERÍSTICAS DE LAS INTERFASES DE USUARIO

7.1 Introducción

La Interfase de un programa es la encargada de llevar el diálogo entre el usuario[1] (en este caso, el estudiante) y el sistema de una manera natural que permita extraer la máxima productividad en la relación entre ambos. Principalmente muestra las posibles acciones que se pueden llevar a cabo y los resultados de estas acciónes. Se ha llegado a un consenso entre los diseñadores de programas educativos sobre el tipo de interfase de usuario que debe acompañar a las aplicaciones. Se llegó a la conclusión de que la principal característica de dicha interfase es que es gráfica. La razón de ésto es el uso constante que los PE hacen de las técnicas de visualización de datos, como la codificación en color y en forma, de la manipulación directa de objetos en pantalla y de la animación (que son imposibles de programar en terminales y computadoras que sólo soportan texto).

La segunda característica importante es la estandarización de elemetos de la interfase como menus, botones, imágenes (íconos o logotipos), ventanas, formas de captura de datos (o ventanas de diálogo), etc. Otra característica importante es capacidad de interactuar. Es decir, se espera que las interfases de los PE sean interactivas en el sentido de que los estudiantes estén constantemente dando y recibiendo información de y hacia el programa. En este aspecto, el paradigma más aceptado es el llamado "Programación LLevada por Eventos", en el cual se intenta integrar todas las posibles formas de interacción, como ratón, teclado, pantalla sensitiva, y a veces comados por voz; de tal forma que todos estos eventos sean los que afecten el flujo del programa, asociándose un procedimiento computacional a cada posible evento y adicionalmente a través de la manipulación de estos eventos se logra que el usuario lleve el control maestro del programa.

Además debe mencionarse como no menos importante que los programas deben ser totalmente orientados al usuarios. Es decir, que el usuario debe ser retroalimentado con información útil cada vez que cometa un error o no sepa que hacer al llegar a cierto punto, que el programa esté protegido (y por lo tanto el usuario) contra todos los posibles errores, desde los más obvios y veniales hasta los más improbables y complejos, y además, que siempre habrá una forma sencilla de corregir estos errores. También, el usuario no tendrá que esperar largos intervalos de tiempo para obtener algún resultado y

---

[1] La discusión concerniente al diseño de interfases es relevante para cualquier sistema, ya sea educativo o no. Por la tanto, en este capítulo cada vez que se mencione al usuario, debe asumirse que la discusión se está refiriendo al usuario de un programa educativo, o sea, el alumno.

que no tendrá que pasar por sesiones largas y aburridas de entrada de información. El usuario siempre tendrá la opción de adelantar o repetir las lecciones y secciones del programa que desee. Y finalmente, que los comandos sean fáciles de aprender y recordar y que su uso sea intuitivo.

Otra característica deseable pero no fundamental, es la capacidad de adaptación de la interfase, en la cual los alumnos de mayor aprovechamiento podrán avanzar más rapidamente, e incluso, saltar algunas secciones del programa que los que demuestren tener desventajas en el aprendizaje. En pocas palabras, el utilizar un programa educativo no sólo debe ser instructivo sino agradable.

7.2 Dispositivos Físicos

Aunque una discusión extensiva sobre dispositivos físicos de interacción como el ratón, el teclado, pantallas sensitivas, pantallas CRT, reconocedores de voz, digitalizadores, etc, se puede considerar como pertinente, en este trabajo no se hará por la sencilla razón de que la disponibilidad de hardware es en raras ocasiones una opción para el programador. Además, la descripción de estos dispositivos puede extenderse demasiado si también se incluyen las características físicas y partes mecánicas. Sin embargo, sí se incluye una breve descripción funcional de los dos dispositivos físicos de entrada de datos más comunes, el ratón y el teclado.

7.2.1 El Teclado

El teclado como se conoce ahora fué diseñado hace más de cien años por dos norteamericanos, Sholes y Glidden [RUBI88]. Su objetivo original fué el diseñar un aparato mecánico para numerar boletos de tren, cheques de banco y hojas de libros en forma consecutiva. La extraña disposición que tienen ahora los teclados viene de aquellos tiempos, el llamado teclado QWERTY, por el arreglo de las primeras seis teclas (en la parte superior izquierda). Este teclado, sorprendentemente no fué diseñado para maximizar la velocidad de tecleado, sino al contrario para disminuirla. Las primeras máquinas de imprimir tendían a atascarce seguido, debido a que aún entoces, las personas aprendían a mecanografíar rápidamente. Se han hecho varios estudios respecto al arreglo de las teclas, y se ha llegado a la conclusión que el arreglo que se utiliza normalmente está muy lejos de ser óptimo. Sin embargo, todos los intentos para cambiar a teclados más modernos, como el alfabético o el DVORAK han fracasado debido a la enorme inversión en horas de aprendizaje y dinero que con el tiempo se ha acumulado con los teclado QWERTY.

La mayoría de las personas utilizan el teclado observando las teclas que se van oprimiendo en lugar del recomendado procedimiento de observar el resultado en la pantalla y no las teclas. Debido a eso, es recomendable que el teclado esté situado siempre en el mismo lugar, que sea lo suficientemente compacto para disminuir el

área donde se buscan las teclas y que las etiquetas de las teclas sean grandes y claras. Las dimensiones de las teclas deben ser de alredor de 12-15mm^2 con un espaciamiento de 19mm+/-1mm entre el centro de cada tecla. Es también recomendable utilizar diferentes tonos en el color de las teclas para diferenciar grupos con diferentes funciones, como teclas de cursor, de función, numéricas o alfanuméricas. Lo mismo se puede hacer con el tamaño y forma de las teclas.

Existen tres grupos diferentes de teclas que se incluyen en los teclados computacionales, aparte de las teclas alfanuméricas heredadas de la máquina de escribir, cuya función es importante.

7.2.1.1 Teclas de Comados

La intensión de estas teclas es la de proveer de comandos completos sin necesidad de tener que teclear todo un comando o accesar un menú. Ejemplos clásicos son "Inserción", "Supresión", "Borrar", "Avance de Página", etc. En general son muy buena idea, sin embargo, si se utilizaran las suficientes teclas como para crear un vocabulario de comunicación con la máquina lo suficientemente flexible el tamaño del teclado crecería demasiado volviéndolo de dimensiones colosales.

7.2.1.2 Teclas de Función Programables

Estas teclas tiene la misma función que las teclas de comandos, excepto que no tienen ninguna función permanente aplicada. Las funciones varían de aplicación en aplicación. Por lo general las funciones asignadas a estas teclas son muy inconsistentes lo que las hace muy difíciles de aprender y retener en el tiempo, volviendo su utilidad nula.

La localización de éstas teclas es importante. Deben estar lo suficientemente separadas de las teclas alfanuméricas como para que pueda exitir una distinción real, pero al mismo tiempo deben estar al alcance de los dedos del usuario sin mucho esfuerzo.

7.2.1.3 Teclas de Control de Cursor

Estas teclas sirven para controlar la posición del cursor en la pantalla. Son revolucionarias porque hacen que la pantalla pase de un simple intérprete de comandos que trabaja de línea en línea, a una completa aplicación de pantalla completa. El movimiento del cursor permite seleccionar comando en pantalla y nombres de objetos. Cuando éstas teclas no existen tiene que ser simuladas con teclas alfanuméricas como los arreglos awsz, esdx ó ikmj, limitando la utilización de las mismas para la entradada de texto.

El arreglo de las teclas de función es importante. Se ha encontrado que las formas más útiles son en forma de cruz y de "T" invertida, pero arreglos en forma de "L" acostada y lineales son comunes.

A pesar de todas sus deficiencias, el teclado sigue siendo la forma más genérica y flexible de interacción con la máquina. Además, el teclado es un dispositivo presente en todos los sistemas computacionales de cualquier plataforma, cosa que no se puede decir de todas los demás dispositivos. Así, hasta que se perfeccione y se abarate la tecnología de los reconocedores de voz, los tecleados llegaron para quedarse.

7.2.2 El Ratón

El ratón es un pequeño dispositivo del tamaño de un jabón con uno, dos o tres botones en la parte superior. Está conectado al computador a través de un cable lo que le dá la apariencia de un ratón. Su función principal es la de seleccionar objetos en pantalla. Esto se logra moviendo el ratón en una superficie plana. El ratón está equipado con un dispositivo para detectar ese movimiento y medir su dirección, por ejemplo un bola de goma conectada a potenciómetros o detectores ópticos. El movimiento del ratón es mapeado a movimientos relativos en pantalla, donde un cursor, típicamente una flecha, cambia su posición en pantalla correspondiendo a los movimientos del ratón. Cuando el cursor se haya colocado sobre el objeto deseado, como un logotipo o un menu, se presiona uno de los botones (si son más de uno, el botón más usado es el izquierdo). Al accionarse el botón, el programa registra la posición del cursor en pantalla he identifica el objeto que se esté señalando e inicia la acción correspondiente.

La principal ventaja del ratón es que son muy fáciles de usar. Una persona que nunca ha utilizado un ratón puede aprender a dominarlo en 15 minutos (dominar el mapeo del ratón al cursor puede costar trabajo), mientras hacer las mismas tareas con el teclado puede necesitar de muchas horas de entrenamiento. Además, debido a que el ratón es ligero y fácil de manipular, permite al usuario trabajar en una posición cómoda, o cambiar de posición sin perder el hilo de la acción.

Para poder programar utilizando un ratón, es necesario conocer tres datos; la posición, si se ha oprimido algún botón y cuál botón se ha oprimido. La posición indica sobre qué objeto se desea realizar la acción. El poder preguntar si se ha oprimido algún botón ayuda a identificar en qué momento se desea llevar a cabo la acción y el botón que se ha oprimido ayuda a identificar entre posibles opciones.

Otra característica de las acciones generadas por medio del ratón es que el usuario espera que la respuesta sea inmediata. Las acciones de ratón son del más bajo nivel mental (léxico), por lo que cualquier retraso en la respuesta, ya sea el

posicionamiento del cursor, o en la acción que debe generar el oprimir un botón, como la aparición de un menú o el inicio de un cálculo puede resultar desesperante.

Así, la combinación del ratón con el software apropiado pueden hacer de esta herramienta una de las más útiles en la interacción con el sistema porque es fácil de utilizar, fácil de programar y es muy versátil.

7.3 Dispositivos Virtuales

Dentro de los dispositivos virtuales de la interfase, se pueden reconocer aquellos que sirven para la recolección de información como los lenguajes de comandos, los menús, las formas y los botones, y aquellos que sirven para proporcionar información al usuario, como las cajas de diálogo y los logotipos.

7.3.1 Ventanas

Las ventanas son dispositivos virtuales que en su más sencilla forma separan y claramente definen una sección de la pantalla para indicar un cambio de contexto en información. Es decir, en su forma más simple se puede dividir la pantalla en varias secciones en las que el usuario puede buscar la información de acuerdo a su categoría. Por ejemplo, un área puede estar designada a instrucciones del sistema, otra área puede estar designada para capturar información sobre opciones que desee el usuario, en otra se leen los mensajes de error y en otra mucho mayor se pueden ver los resultados de todas estas acciones.

¿Cuál es el máximo de ventanas que se pueden desplegar sin perder la utilidad de éstas? Esta respuesta depende de las características y complejidad de las ventanas. Además de ésto, existen limitaciones físicas como la resolución del monitor y la capacidad de memoria de la computadora. No obstante, la cantidad de ventanas que se pueden tener en pantalla depende de la forma en que se acomoden en pantalla.

7.3.1.1 Superimposición, Tejado y Movimiento

La siguiente característica que puede aparecer en las ventanas es la capacidad de superimponerse. Esto quiere decir que el contenido de un ventana puede dibujarse arriba de otra. Cuando la ventana que está arriba de la otra desaparece, el contenido de la ventana ocultada se vuelve a desplegar intacto. Cuando las ventanas no están superimpuestas se dice que están tejadas[2]. Es decir, que ninguna ventana está arriba de otra, y por lo general, los bordes de todas las ventanas coinciden. En general, si se utilizan ventanas

[2] Haciendo analogía al poner tejas en los techos de las casas, o azulejo en los baños, donde cada unidad está colocada a un lado de la otra.

que se superimponen, la cantidad que se puede manejar es mucho mayor que si se utilizan tejadas, pero la probabilidad de tener una pantalla congestionada es mayor. Es conveniente señalar que la utilización de ventanas que se ocultan unas de otras generalmente implica la implementación de algún tipo de mecanismo para poder mover dichas ventanas y así poder arreglar la pantalla como mejor convenga.

7.3.1.2 Pantalla Virtual

Si se decide utilizar pantallas tejadas, es muy probable que la cantidad de información que se desea desplegar no quepa en las ventanas. En tales circunstancias, la ventana puede pasar de ser una simple área de la pantalla normal a una verdadera ventana a un área con mucha mayor capacidad para contener información que sería una pantalla virtual. La pantalla virtual es un dispositivo cuya información sólo puede ser accesada a través de la ventana. Si la ventana fuera lo suficientemente grande, podría mostrar toda la información contenida en la pantalla virtual. Sin embargo, como generalmente no lo es, la ventana debe proveer algún tipo de dispositivo para cambiar la vista que se obtiene de la pantalla virtual. Esto generalmente se logra utilizando unos pequeños elevadores verticales y horizontales que cambian el área de la ventana virtual que se está observando. Esto también se puede hacer utilizando botones con logotipos indicando la dirección hacia donde se quiere mover la vista de la pantalla virtual[3].

7.3.1.3 Tamaño y Escalamiento

Otra característica deseable en una ventana es poder cambiar el tamaño de ésta de una manera sencilla. Esta característica es indispensable cuando se tienen pantallas virtuales porque permite aumentar la vista de la información hasta el tamaño completo de la pantalla si así se desea para poder utilizarla con facilidad, o disminuirla tanto como sea posible para que no estorbe la vista de otra pantalla o poder utilizar la ventana junto con otras al mismo tiempo.

Esto se puede lograr utilizando botones rápidos que aumenten o reduzcan el tamaño de la ventana en tamaños y proporciones fijas. También se pueden utilizar "agarraderas" que el usuario puede señalar con el cursor del ratón, cambiándose el tamaño de la ventana conforme se mueve el cursor hasta que el ratón libera la agarradera.

Cuando se hace un cambio de tamaño se puede decidir escalar la información contenida de tal forma que se ajuste al nuevo tamaño. Si se hace

---

[3] Esto es lo que en lengua inglesa se describe por el simple término de "scrolling"

así, se tiene toda la información a la mano y se evita la utilización de elevadores para mover la vista de la ventana. Sin embargo, esto puede ocasionar que dicha información se distorcione.

Otras características deseables en las ventanas son:

1. La capacidad de incluír títulos para poder identificar fácilmente el contenido de las pantallas cuando estas se ocultan mutuamente.

2. Un dispositivo para poder cerrar rápidamente la ventana. Esto generalmente se implementa con un botón especial colocado en una parte visible de la pantalla.

En resumen, las ventanas son ideales para la rápida organización de la información desplegada cuando se pueden implementar. Aún cuando no es posible implementar un mecanismo completo de ventaneo, con el sólo hecho de delimitar alguna o algunas áreas en pantalla y desplegar información lógicamente agrupada según su función en esas áreas es suficiente. La fig. 8 muestra un ejemplo de una ventana.

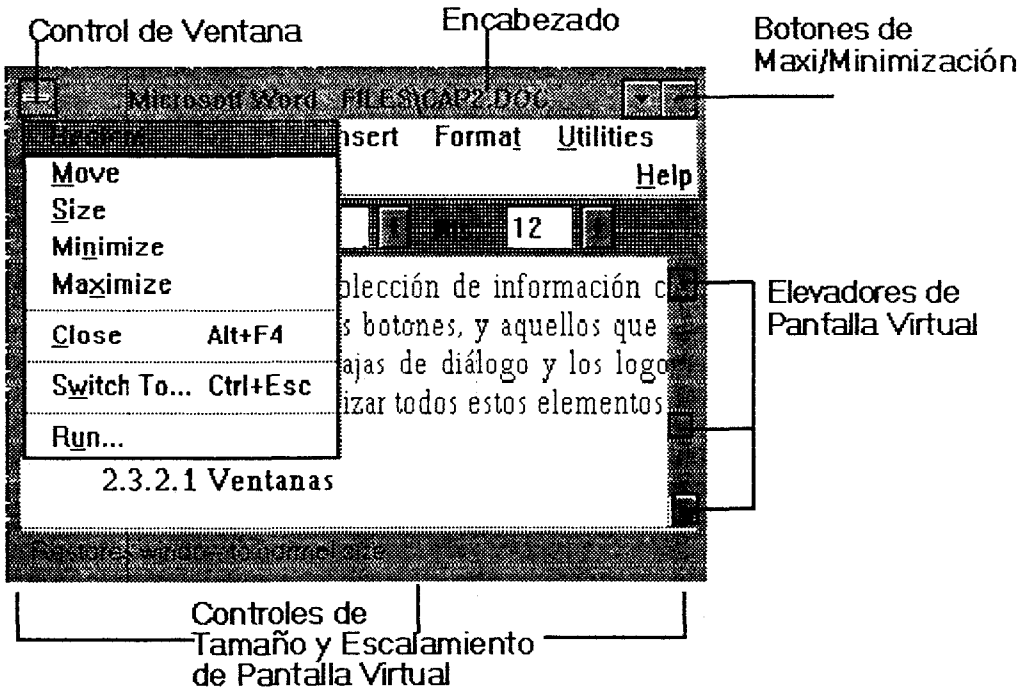


Fig 8 Elementos de una Ventana

7.3.2 Menús

Originalmente los menús se diferenciaban muy poco de los comandos comunes y corrientes de un sistema. Generalmente se presentaba una lista de opciones, el usuario la leía y escogía su opción, y tecleaba los caracteres necesarios para identificar la opción. Entonces los menús no eran diferentes a la utilización de lenguajes de comandos con una pequeña ayuda. Pero ahora, los menús han evolucionado hasta convertirse en los dispositivos más utilizados en el diálogo entre el hombre y la máquina.

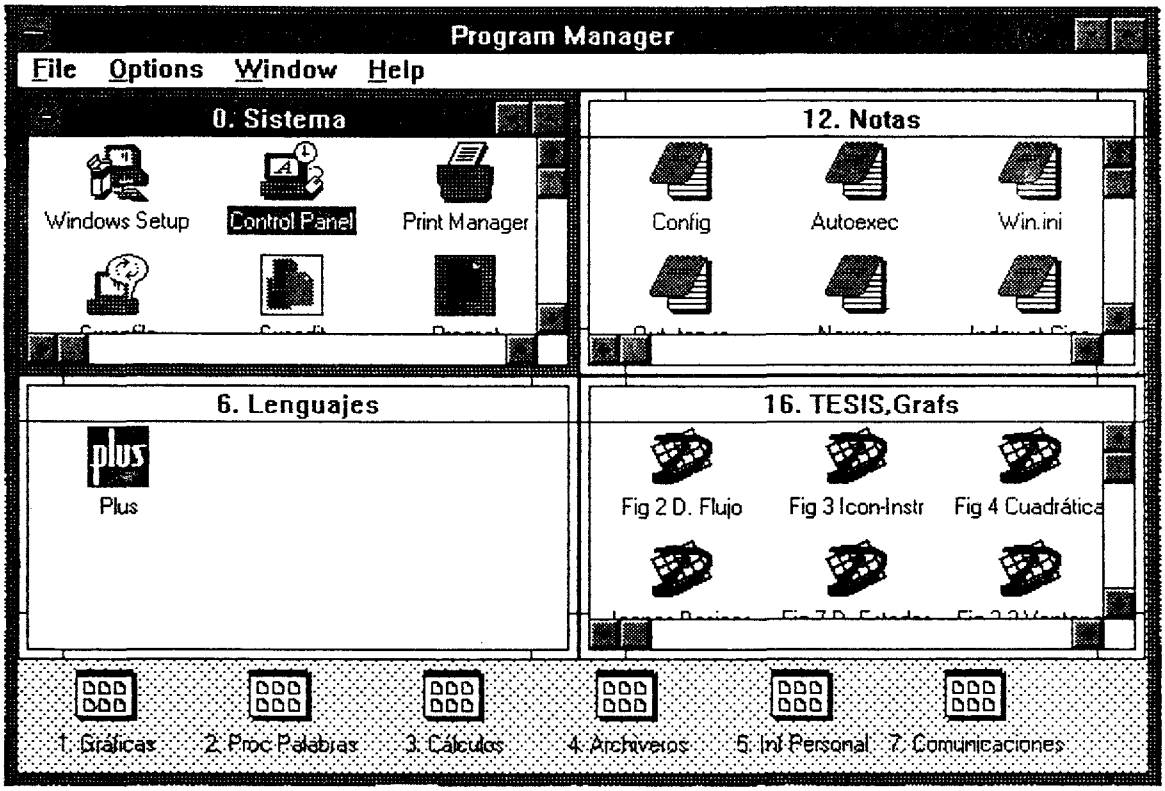


Fig. 9 Ventanas Tejadas

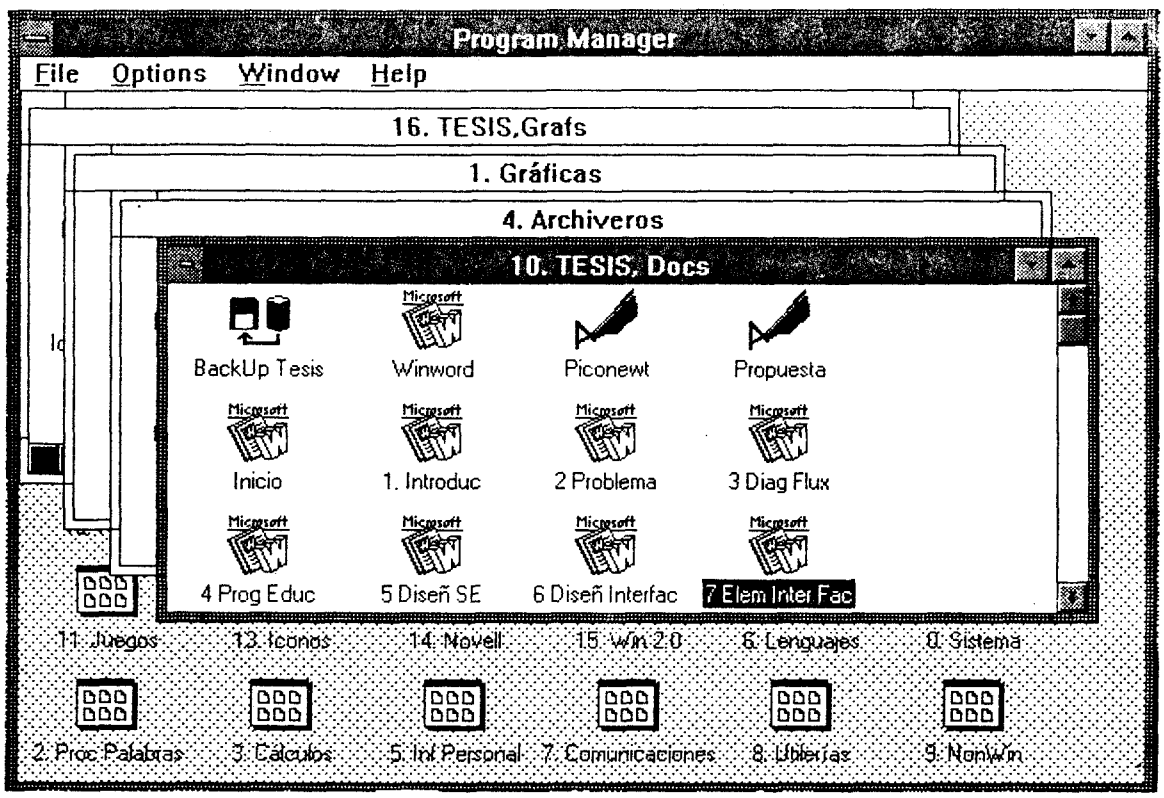


Fig. 10 Ventanas Superpuestas en Cascada

7.3.2.1 Tipos de Menús

Ahora los menús se presentan en muchas formas. Por ejemplo, todos los menús pueden estar agrupados en la parte superior de la pantalla denominada barra de menús. Un menú puede ser invocado utilizando una tecla o combinación de teclas como "Ctrl-M", o utilizando el cursor del ratón y accionándolo en la barra de menú sobre el menú deseado. Al invocarse el menú, su lista de artículos se "desenrrolla" sobre la pantalla y el usuario puede escoger su opción, esto es lo que se llama menú "drop down", o **menú desenrrollado**. También puede ser que el menú se llame accionando el ratón en alguna región de la pantalla, haciendo que el menú aparezca en cualquier lugar de la pantalla aún cuando nunca estuvo a la vista. Esto es lo que se llama un menú "pop up" o **menú oculto**. También es posible que todos los artículos del menú siempre estén a la vista dibujados en la pantalla en forma de botones con logotipos representativos de las acciones que generan en lugar de etiquetas. El usuario únicamente tiene que accionar el ratón sobre el botón para que se inicie una tarea deseada. Finalmente es posible que el menú contenga artículos que no son mútuamente excluyentes o que pueden o no especificarse. Generalmente consisten de opciones que se pueden autorizar

marcando una pequeña cruz o paloma en un área adyacente a la etiqueta del menú dedicada a ésto[4], estos son los **menús de control**.

Cualquiera que sea su forma, el menú moderno consta de una característica muy especial; la selección de la opción requiere de una indicación directa por medio del ratón o del teclado sobre la opción deseada.

Debido a que las posibilidades de acción siempre están a la vista del usuario, los menús son especialmente efectivos cuando los usuarios tienen poco entrenamiento, no utilizan el sistema muy seguido o no están familiarizados con la terminología, y necesitan ayuda para estructurar sus decisiones.

7.3.2.2 Organización Semántica

Es importante que el diseñador que implemente un sistema de menús haga un análisis de tareas detallado para asegurarse que todas las funciones estén convenientemente soportadas y que la terminología utilizada es adecuada. Es decir, que el nombre del artículo en el menú (o en su defecto la imagen que la representa) sea suficiente para identificar la acción asociada.

Todos los menús deben estar organizados en forma semántica, es decir, que un menú debe contener sólo artículos relacionados entre sí, y el título del menú debe indicar esto claramente. Las categorías deben ser claras y distintivas, y en todo momento el usuario debe tener una idea de lo que va a pasar. Además, de ser necesario debe crearse una estructura jerárquica en la que los comandos pueden accesarse a través de varios niveles de menús donde un artículo de un menú genere un submenú, y así, formar un árbol de decisiones.

Otro aspecto importante en el diseño de menús es que la cancelación de una acción generada por menú no sólo siempre debe ser posible, sino que siempre debe ser un procedimiento sencillo e intuitivo. La cancelación debe dejar al sistema en estado idéntico a como estaba antes de que se accionara el menú.

7.3.2.3 Diseño de Menús

Siguen algunas consideraciones a tomar en cuenta cuando se diñan menús:

---

[4] La terminología en inglés para estas áreas es "Check Box"

1. Los Títulos deben ser descriptivos, dando una idea general de la clase de artículos que se está agrupando en un menú. También deben ser breves y consistentes con el estilo gramático.

2. No construir sistemas de menú con más de cuatro niveles.

3. Los artículos deben agruparse dentro del mismo menú. Deben ponerse entre cuatro y ocho artículos, incluyéndose todas las posibilidades. En general, los artículos deben ser mútuanente excluyetes en sus funciones.

4. La terminología de las etiquetas debe ser familiar, aún a expensas de sacrificar un poco la lengua. Evite etiquetas chistosas. La primer palabra o letra de las etiquetas debe ayuda a discriminar.

5. Cuando sea posible ofrecer ayuda. Esto se puede hacer incluyendo la ayuda como un artículo dentro del menú, o desplegando mensajes significativos en algún área de la pantalla.

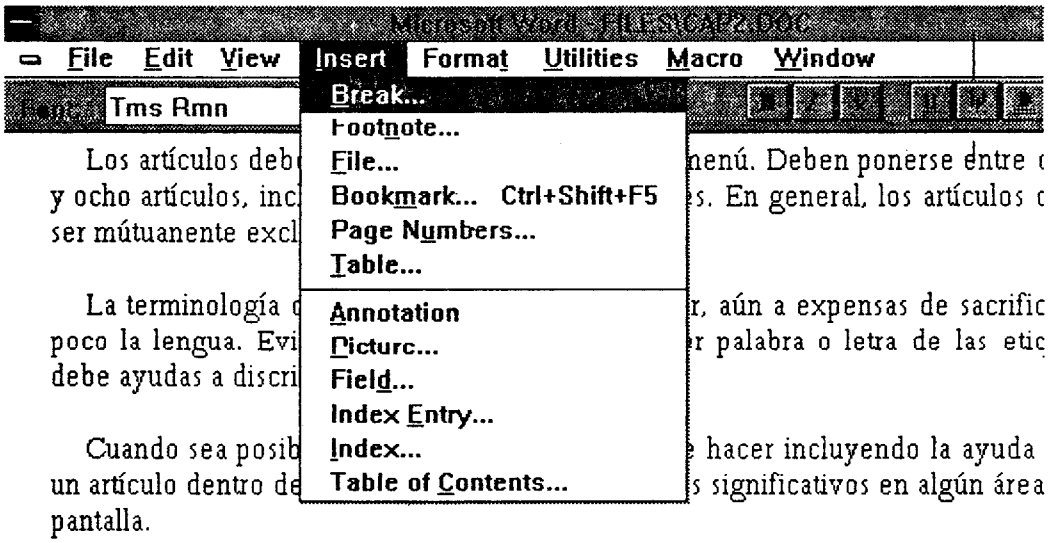


Fig. 11 Menú Desenrollado (Drop Down)

7.3.3 Formas

Las formas son el análogo electrónico a las formas que se tienen que llenar cuando se va a solicitar empleo, cuando se está realizando un senso, cuando se está llevando un inventario,etc. La idea es prorcionar un mecanismo que permita recabar importantes cantidades de información. Sin embargo se corre el riesgo de que la sesión de recolección de información resulte aburrida, o peor aún, que la

persona que está proporcionado la información no sepa lo que se le está preguntando. Por eso es importante que el título de la forma sea significativo, es decir, que claramente indique el tipo de información que se desea recabar.

Deben siempre incluirse instrucciones sobre la manera de proporcionar la información. Estas instrucciones deben ser breves pero claras. Debe dedicarse una sección de pantalla para el desplegado de mensajes de error, y siempre permita la corrección de errores de una forma sencilla.

Los distintos campos (datos que se piden) deben estar agrupados lógicamente. El cursor debe colocarse sobre el campo que se está llenando. Cada campo debe estar debidamente etiquetado, con palabras familiares al usuario, de tal forma que se entienda claramente la información que debe contener. La terminología debe ser consistente con la utilizada en todo el sistema. Además, los rangos de cada campo deben ser siempre visibles. Deben indicarse cuáles campos son opcionales. Proporcionar siempre que sea posibles valores por omisión que el usuario pueda escojer no cambiar, simplemente autirizando su uso.

El diseño de la forma debe ser agradable. Por ejemplo los campos pueden estar bien alineados y la forma en general puede tratar de parecer una forma real. Los límites hasta donde se puede teclear información deben ser también visibles. Cuando pueda existir duda sobre el formato de cierta información, proporcionar ejemplos; como con los números telefónicos, la hora, las fechas y el dinero.

Cuando sea posible, dar una señal avisando que se ha llenado la mínima información necesaria de la forma. Por último la forma debe generarse dentro de algún tipo de ventana. Es decir, cuando se alcanza el punto en que es necesario llanar un forma, debe por lo menos delimitarse él área en pantala que va a ocupar, pero de ser posible, debe haber facilidades para mover la forma y reducir su tamaño.

7.3.4 Logotipos

Un logotipo[5] es una representación gráfica de algún objeto en pantalla. En general, la forma más utilizada es un pequeño dibujo. La idea del logotipo está directamente relacionada con el paradigma de manipulación directa (sección 7.5), debido a que todas las tareas relacionadas con el objeto se llavarán a cabo por medio de la representación gráfica.

---

[5] En algunas ocasiones se les denomina Iconos

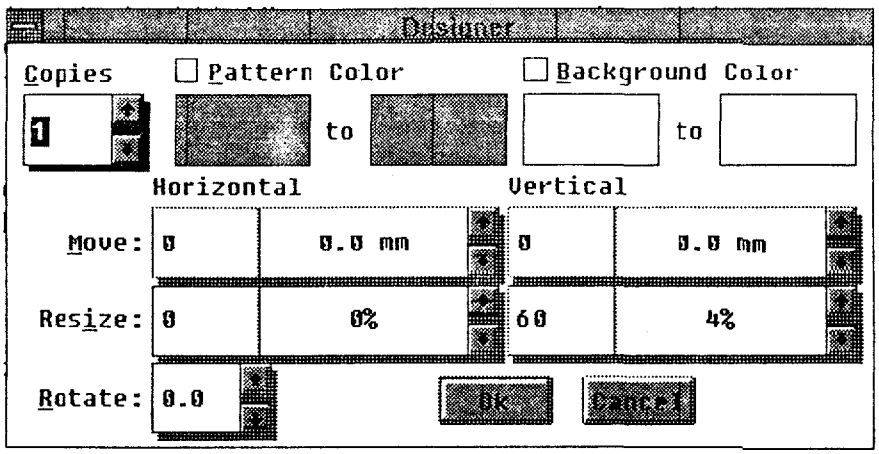


Fig. 12 Forma de Datos

7.3.4.1 Tamaño del logotipo

El tamaño de los logotipos generalmente es muy variable, aunque no se ha encontrado evidencia que indique que necesariamente tenga que ser así. Sin embargo, una imágen que realmente pueda representar algo, tiende a ocupar una parte mas bien grande de la pantalla. El tamaño estándard con el que se manejen los logotipos en un sistema tenderá a ser aquél que represente la información requerida ocupando el máximo tamaño aceptable. Dado que un logotipo de tamaño menor al estándard no sería muy útil y uno de tamaño mayor sería inaceptable, la mayoría de los sistemas estandarizan el tamaño de los logotipos. El tamaño generalmente varía entre 30x30 y 50x50 pixels.

7.3.4.2 Significado del logotipo

En sus orígenes los logotipos no eran más que pequeñas cadenas de texto. Estas cadenas tienen la ventaja de no sufrir ambigüedades, pero tienen la grave desventaja de que es muy difícil identificar artículos individuales cuando el número sobrepasa de de seis u ocho. Entonces se empezaron a utilizar representaciones pictóricas porque aprovechan las capacidad del cerebro de identificar rápidamente imágenes. Sin embargo éstas imágenes, debido a la limitación de tamaño pueden caer en ambigüedades. Es decir, el dibujo realmente no indica nada, o es difícil diferenciar las entre varios dibujos. Actualente se utiliza una conbinación de ambos enfoques, en el que el dibujo indica el tipo de objeto que está siendo representado por el logotipo, y una pequeña cadena de caracteres que sirven para resolver ambigüedades y hacer identificaciones individuales. Ejemplos característicos de esto son los logotipos que utiliza la computadora Macintosh para representar archivos y directorios. El dibujo del archivos indica la aplicación que lo generó, mientras

que la identificación individual se logra por medio de una cadena de caracteres que generalmente indican el nombre.

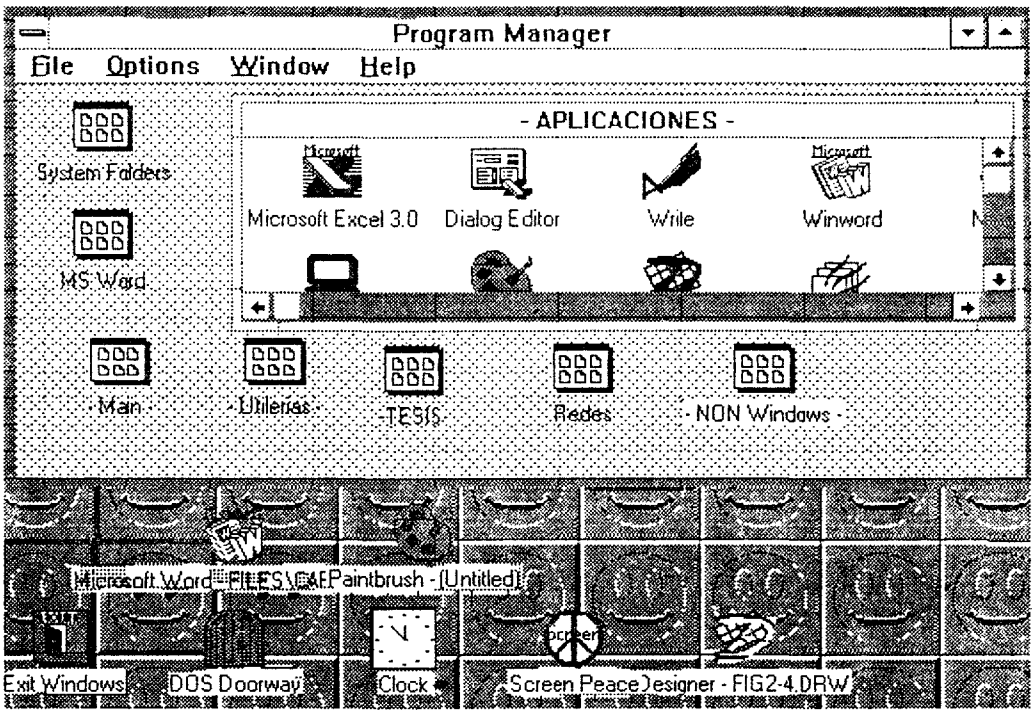


Fig. 13 Utilización de Logotipos

7.3.5 Botones

Los botones son una representación gráfica de iniciadores de acciones que se pueden llevar a cabo en un cierto momento. Su tamaño y su forma es muy variada, pero todos representan comandos completos. Un botón puede ser simplemente una pequeña área de la pantalla, hasta una completa representación tridimensional. Aparte de la imágen (aunque sólo sea un cuadro), van acompañadas de una cadena que indica el nombre del comando representado. Los botones también están directamente relacionados con el paradigma de manipulación directa porque es necesaria una acción directa sobre éstos para generar la acción que representan, por ejemplo posicionar el cursor del ratón y oprimir el botón izquierdo. En sí, los botones son parte de un menú que muestra al usuario todo el vacabulario de acciones a su disposición. Son una especie de menú oculto (pop up), que requiere de manipulación directa.

Otra categoría de botones son aquellos llamados botones de control, que sirven para prender y apagar opciones de operación del sistema.

7.3.6 Combinación Botones y Logotipos

Los logotipos no sólamente pueden representar objetos, sino también acciones. Dada que la representación natural de una acción en una interfases interactiva es el botón, en ocasiones ambos elementos utilizan en una poderoza combinación de botones con logotipos y menús. Discretamente, algunas partes de la pantalla se llenan con botones y logotipos representando las acciones más frecuentes del sistema. Esto permite al usuario no sólo olvidar por completo la sintaxis de los comandos, dado que con la utilización de botones ya no es necesaria, sino incluso, puede olvidar el mismo nombre del comando debido a que éste será sustituído con la imágen generada por el logotipo, dándo a la interfase una intuitividad que va más allá de las barreras del lenguaje. Además, estas acciones, si están presentes la mayor parte del tiempo, están al alcance de los dedos del usuario por medio de los botones quien no tendrá que accesar menús y submenús, aprender combinaciones raras de teclas ni el nombre ni sintaxis de ningún comando.

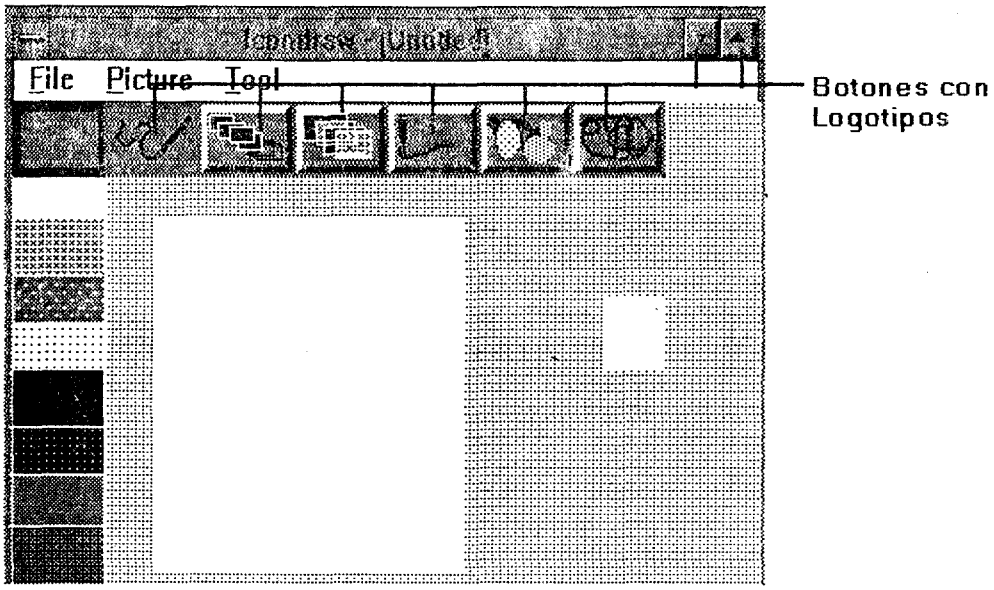


Fig. 14 Utilización de Botones con Logotipos

7.3.7 Lenguajes de Comandos y Lenguaje Natural

7.3.7.1 Lenguajes de Comandos

Antes de que apareciera el concepto de interfase con el usuario, todos los sistemas computacionales eran manejados a través de comandos. Ejemplos típicos son los sistemas operativos como el DOS y UNIX y la mayoría de las aplicaciones que corren en ellos. Aunque arcáicos, los lenguajes de comandos

tienen varias ventajas. Permiten gran control sobre lo que se está haciendo porque la mayoría de los comandos tienen gran cantidad de opciones. Los lenguajes de comandos dan una sensación de cercanía con el sistema como ningún dispositivo. Son muy rápidos. La mismas opciones que se han mencionado, comúnmente implementadas como argumentos del programa, permiten que varias acciones se lleven a cabo con un sólo comando; por ejemplo, las siguiente serie de acciones "formatea los discos a, b y c, ponles etiquetas 101, 102, 103 e inserta el sistema" pudieran llevarse a cabo con una sola línea de comando. La utilización de estos comandos resulta para los usuarios expertos mucha más rápida y cómoda por la posibilidad de hacer muchas cosas tecleando algunos caracteres rápidamente que tomando el ratón, buscando y navegando a través de varios menús, etc.

Sin embargo, los lenguajes de comando tienen una serie de inconvenientes que en opinión de la mayoría de los usuarios comunes sobrepasan sus conveniencias. Por ejemplo, incluso usuarios experimentados tienden a cometer muchos errores, ya sea por el olvido de algún argumento importante o por haber equivocado la sintaxis del comando inicial. Los lenguajes de comandos requieren de mucha memoria. Los nombre de los comandos son largos y a menudo con una sintaxis difícil y poco lógica, a menudo derivada de un lenguaje extranjero: CHKDSK, significa Check Disk, o verificar el estado del disco. Y por si eso no fuera poco, es muy raro el sistema manejado por lenguajes de comandos que incluya una ayuda en línea.

7.3.7.2 Lenguaje Natural

La utilización de lenguajes de comandos, ha recordado a los diseñadores de sistemas expertos la idea de un verdero diálogo entre el hombre y la máquina. Un diálogo no llevado por comandos impersonales, ni por tontas interfases de dibujos animados sino por una verdadera conversación entre dos seres inteligentes. Se han logrado desarrollar aplicaciones en el área de inteligencia artificial y sistemas expertos donde seres humanos consultan situaciones con expertos electrónicos. Es difícil imaginar una forma de llevar un diálogo con un erudito que no sea haciéndole preguntas directas y contestando las suyas. Aunque se ha avanzado mucho en este sentido, se ha encontrado que la utilización de algo cercano al lenguaje natural tiene varias deficiencias. Por ejemplo, el usuario nunca tiene un sentido de orientación, no sabe su situación instantánea dentro del sistema (por ejemplo en qué lección va) y no sabe qué más puede hacer. Además, el lenguaje utilizado por el mismo sistema computacional es en ocasiones oscuro y requiere de constantes diálogos de verificación. Todo esto sin tomar en cuenta que el diálogo se realiza en una forma torpe y lenta, porque se tienen que construir y entender frases enteras.

7.2.8 Cajas o Ventanas de Diálogo

Como se puede ver del título, las cajas de diálogo son ventanas en las que se lleva a cabo la comunicación entre el sistema y el usuario. En estas cajas se combinan todos los elentos anteriores. Supóngase que el usuario pide que se imprima la información, entonces aparece una ventana. Puede contener un forma donde el usuario indicará datos del formato de la página como márgenes y tamaño de página. Contendrá una serie de botones de control en los que puede indicar que desea el documento con calidad final, o sólo una impresión de prueba en pantalla. Contendrá un menú de botones con una serie de comados como "Imprimir Ahora", "Cancelar", "Opciones Avanzadas", etc. Si se tiene la opción de imprimir otros documentos, podría aparecer un menu escondido que muestra todos los archivos que se puden escoger para continuar la impresión y mucho más.

Se puede ver que las cajas de diálogo cumplen un doble propósito, muestran al usuario información relevante como mensajes de error e instrucción, y sirven como puertos de entrada para la información que el sistema necesita tomar del usuario. La utilización de una ventana ayuda a llamar la atención sobre la ocurrencia de un evento que no está relacionado con el funcionamiento estándard del sistema. Para lograr el intercambio de información se puede utilizar cualquier tipo de dispositivos virtuales además de los físicos.

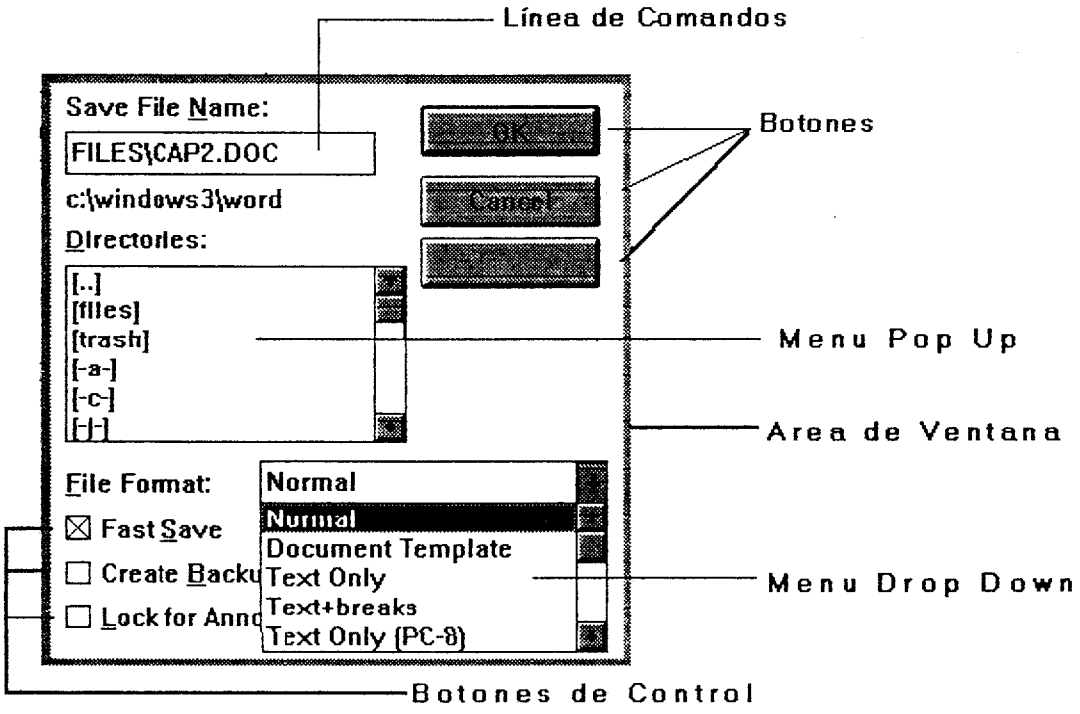


Fig. 15 Una Ventana de diálogo

7.4 Visualización

La definición normalmente dada al verbo visualizar podría ser algo así, "Formar una imágen mental", en lo que se refiere al los programas educativos e interfases, visualizar es mucho más que ver y recordar la imágen. Visualizar es ver, identificar cada uno de los componentes, identificar relaciones y acciones, y crear una estructura mental para almacenar lo que se ha aprendido de la observación. Formalmente, se podría definir **visualización** por lo menos en lo referente a los programas educativos, como **la utilización de imágenes interactivas y dinámicas para mejorar los procesos de aprendizaje y entendimiento**. La importancia de la capacidad de visualizar radica en el hecho comprobado de que más del 50% de las neuronas del cerebro están asociadas con la visión [CUNN90]. Esta especie de especialización humana en el procesamiento de imágenes deriva en que la presentación de datos por medio de imágenes tiene tres ventajas enormes:

1. Permite la identificación más rápida ya sea de objetos como de acciones. Esto tiene gran aplicación tanto en la presentación de instrucciones y comandos, como en la presentación de información.

2. La comunicación se hace de manera más compacta. Es decir que la utilización de imágenes permite comunicar más ideas en menos espacio.

3. El procesamiento de la información es más rápido y más eficiente. Bajo esta circunstancia, la presentación de conceptos en forma pictórica hace que éstos sean entendidos más rápida y fácilmente. Sin embargo, se debe tener mucho cuidado en la forma en que se presenta la información, pues si los objetos pictóricos que se presentan tienen deficiencias físicas, están borrosos, oscuros, demasiado pequeños, o demasiado ambiguos, el proceso de aprendizaje no sólo no es más deficiente, sino simplemente puede verse interrumpido.

Pero además, la meta primordial de la visualización no es sólo hechar la maquinaria neuronal a andar, que de por sí es un fin bastante deseable, sino establecer una estrecha relación entre la visión, la intuición y la percepción. Y más allá, visualizar debe implicar entender.

7.4.1 Visualización en las Ciencias

El ejemplo más típico de la utilización del razonamiento visual se encuentra en matemáticas. Gran parte del material que se cubre en los cursos de matemáticas puede tener un tratamiento geométrico en la resolución de problemas y en el descubrimiento de conceptos básicos. La geometría es un de los ejemplos más puros del razonamiento visual. El mismísimo Isaac Newton no probó sus teoremas fundamentales por medios matemáticamente rigurosos. Si se le pidiera la demostración de alguno de sus postulados, él presentaría una serie de argumentos

que aunque convincentes, hubieran sido poco formales y altamente dependientes de dibujos.

El ejemplo anterior parece indicar que la utilización de la visualización es una herramienta que sólo puede utilizarse en las ciencias aplicadas, pero este no es el caso. La utilización de la geometría como medio válido de probar teoremas cayó en desuso he incluso fué altamente desaprobada por la academia, cediendo su logar a la lógica rigurosa. Es sólo ahora con al utilización de computadoras de alta velocidad con monitores de alta resolución que la utilización de la visualización va ganando terreno. Y son estas mismas computadoras que con la utilización de los multimedios que han extendido las posibilidades de la visualización hasta las ciencias sociales y humanidades.

Aún sin la presencia de las computadoras, la visualización aplicada a las ciencias provee la habilidad de dibujar el diagrama apropiado para representar un concepto o un problema, y utilizar este diagrama para lograr el entendimiento y como ayuda para la resolución del problema. La visualización es un intento de hacer posible observar lo que usualmente no se ve. De lograrse, a través de su utilización en cómputos y simulaciones, se llega a enriquecer enormemente el entendimiento de en las ciencias.

La visualización es algo que va mucho más allá del simple ver, es una contemplación de idea ya en la mente. La visualización a través de las computadoras, permite dar al aprendizaje un enfoque intuitivo. Pero si además se conjunta con la utilización de manipulación directa (ver sección 7.5), como en los simuladores gráficos, permite hacer de la adquisición de conocimientos un proceso experimental, llevado por la exploración. La visualización es pues, la precepción a través de la intuición visual.

7.4.2 El papel de la representación Visual en el Razonamiento

Para poder utilizar adecuadamente la visualización, debe aprenderse a representar ideas de varias maneras diferentes. Las más utilizadas en las ciencias son simbólica (o sea en lenguaje matemático), numérica y gráfica. Además, deben desarrollarse la habilidad de poder utilizar estas representaciones simultáneamente. De ahí que el papel de la tecnología sea tan importante. Para poder englobar todo el alcance de la visualización, deben considerarse todas las tecnología disponibles, computadoras, videotape, filme, disco ópticos, etc.

Hay que tomar en cuenta que la visualización debe ser mucho más que la utilización de representaciones alternativas de la información entre visual, verbal y

matemático e incluso algorítmico[6]. La representación visual debe ser parte integral del razonamiento. Esto puede lograrse en tres niveles:

1. La información visual es parte de la información dada para razonar. En el caso más sencillo, se extrae información de una escena y se representa visualmente.
2. La información visual es parte del razonamiento (como lo hacía Newton), a través de diagramas y razonamientos gráficos. En algunas situaciones, es posible que un diagrama físico no sea necesario, porque el individuo puede visualizar los pasos de resolución del problema sin él. Un caso trivial es cuando una persona recibe instrucciones para llegar de un lugar a otro. Puede anotar las instrucciones en un diagrama, pero esto rara vez se hace.
3. La representación visual tiene un papel en la conclusión del razonamiento. En algunas situaciones, se pueden utilizar diagramas para presentar de forma resumida las conclusiones de los resultados, poe ejemplo con diagramas de pastel o de barras en estadística.

7.4.3 Utilizando La Representación Visual

7.4.3.1 Metodología

El transmitir información de forma visual requiere mucho más que sólo saber la información que se desea comunicar. También es necesario saber cómo comunicar esa información. Los siguientes pasos son recomendados [BROW90]:

1. Determinar los detalles críticos a presentar en la imagen. Mostrar estos detalles resaltándolos o eliminando información conflictiva.
2. Determinar el orden en que debe mostrarse el material y presentar este material en una forma lógica y congruente.
3. Ofrecer opciones que expandan el conocimiento de la persona sin confundirla o abrumarla.
4. Buscar que el material a presentar sea un proceso dinámico, con oportunidades para que la persona expuesta a él lo explore y controle.
5. Considere seriamente cómo integrar el aprendizaje visual con otras partes del aprendizaje y de ser necesario, cómo evaluar tal aprendizaje.

7.4.3.2 Técnicas de Visualización

Las imágenes que se utilicen en la visualización pueden tener una o más de las siguientes características, que sirven para codificar al información de una manera adecuada [ELLS90] :

---

[6] Lo que típicamente se conoce como aprender de memoria. o como receta de cocina

1. **Abstracción**. Las imágenes no tienen que ser representaciones fieles de la realidad. La idea es la de representar sólo los aspectos más importantes del objeto, eliminando toda la información redundante e innecesaria. Sin embargo, en ocasiones la utilización de multimedios permite la visualización de objetos y situaciones donde la abstracción es muy difícil, por ejemplo cuando se discuten consideraciones éticas [IBM90].

2. **Discretización**. Consiste en romper procesos que parecen continuos a la percepción humana en pequeñas instancias, que sin embargo, no afecten el flujo natural de la representación comparada con el objeto real. Casos típicos son los simuladores de equipo electrónico y computacional donde las computadoras a través de discretización pueden aproximar muy cercanamente a las condiciones del mundo real.

3. **Color**. El color es una de las herramienatas más usadas en la visualización. Aquí el color pierde su característica estética que tiene en el diseño tradicional de interfases y se convierte una forma rápida de identificar diferencias en el valor de variables correspondientes a características relevantes del objeto de estudio. Típicamente; calor, población, dureza, categoría gramatical, raza, etc.

4. **Geometría**. La geometría es otra forma de codificar características. Tiene la ventaja sobre el color de que se pueden codificar varias caracteríticas a la vez, correspondientes a las dimensiones que tenga la figura, pero no está limitado a esto. Por ejemplo, una figura parecida a una estrella de mar puede representar hasta cinco características; cuatro relacionadas con el tamaño de sus brazos, y una relacionada con el ancho. Con ciertas técnicas, las tres dimensiones de la figura puden utilizarse para la codificación de características. La geometría puede indicar una intensidad especial relacionándola con el tamaño, entre más grande más intenso. Si se toma como ejemplo a dos estrellas de mar, una con un brazo más largo que la otra, indicará que la característica correspondiente al brazo es más intensa en una que en otra. Pero además, la geometría es excelente para discriminar entre distintas categoría de elementos. Aquí se puede indicar como ejemplo típico el diagrama de flujo. Cierta forma geométrica corresponde a instrucciones de escritura, cierta para instrucciones de lectura y otra para instrucciones de cálculo.

5. **Movimiento**. El objetivo del movimiento o Animación es mostrar relación dinámica entre los elementos de la visualización en términos de tiempo y espacio. Por ejemplo se puede mostrar el crecimiento de cierta característica, digamos población, cambiando la geometría, el tamaño, el color o la posición de algún elemento. En general, el movimiento

muestra cambios que ocurren conforme fluye el tiempo. Pero la animación puede ayudar a mostrar relaciones entre elementos. Por ejemplo se puede visualizar geométricamente un adjetivo y un sustantivo, y haciendo que el adjetivo orbite alrededor del sustantivo se indicará que el adjetivo modifica al sustantivo, mientras que un adverbio orbitaría al rededor de verbos y sustantivos. En química, un electrón orbitaría alrededor del núcleo; la cantidades de electrones orbitando indicará la valencia del elemento.

6. **Presentación progresiva.** Presentación progresiva significa que la información es presentada al observador en forma controlada. Es decir, que se inician un proceso de discriminación de información en el que el observador decide qué variables desea observar, elimándose el resto. Es posible también, que se cambie la resolución de la información mostrada, por ejemplo en el caso de una vista global, donde se ven todos los elementos pero con gran pérdida de detalle. Por último, se puede discriminar variables que caigan fuera de un rango especificado. Por ejemplo, en una simulación sobre el proceso de inyección de plásticos, podría seleccionar aquellas secciones de la inyectora que contuvieran plástico lo suficientemente frío para solidificar.

Los usos de la visualizacion son muchos y muy variados. Se está realizando muchísima investigación en todas las áreas humanísitcas y científicas que seguramente llagará a la creación de nuevas técnicas y tecnologías. La discusión que se acaba de presentar queda corta por el simple hecho de no aludir la tecnología de multimedios que está abriendo posibilidades en el diseño de programas educativos sin precedentes.

7.5 Manipulación Directa

La manipulación directa es muy importante en el desarrollo de interfases. Utilizando la manipulación directa, el diálogo entre el sistema y el usuario se lleva de una manera mucho más natural debido a que el control del diálogo es retirado de los arcáicos lenguajes de comandos y se da paso a la tan importante interacción directa entre Hombrey Máquina [CHAB89]. Existe la manipulación directa cuando en un sistema se pueden señalar y manipular objetos de interés. La clase de objetos receptores de manipulación no tiene límites: menús, botones, logotipos, átomos, adjetivos y sustantivos, funciones y sus raíces, rectas, etc. Tony Rubin [RUBI88] define los elementos involucrados en la manipulación directa de la siguiente manera:

1. Representación contínua de objetos de interés.
2. Accción física en lugar de sintaxis compleja (selección con ratón o teclas de cursor).

2. Operaciones rápidas y reversibles cuyo impacato sobre el objeto de interés se aprecia inmediatamente.
3. Enfoque estructurado e intuitivo a la curva de aprendizaje que permite la utilización del sistema con un conocimiento mínimo, y que aliente la exploración y el incremento del conocimiento de una manera grácil.

Como se puede notar de la definición, la manipulación directa implica una representación visual del mundo de acción. Y de ésta representación depende su efectividad. Se ha mostrado que la representación apropiada de los problemas es crítica en el aprendizaje y la resolución de problemas. De aquí que esté tan estrechamente ligada la visualización con la manipulación directa. Ambos conceptos son las bases de un nuevo principio, la Virtualidad: "Virtualidad es la representación de la realidad de tal forma que puede ser manipulada".

La manipulación permite al individuo interactuar con los conceptos que debe manejar el sistema, aplicando sus conocimientos directamente a la tarea; la herramienta desaparece de la escena. Da una sensación de involucramiento directo con el mundo representado y elimina la distracción del usuario debida a la constante comunicación con el sistema.

7.5.1 Relación con el Modelo Sintáctico Y Semántico

Cuando se utiliza manipulación directa, los objetos son desplegados de tal forma que las acciones están directamente en el nivel semántico de tareas. Debido a que hay poca necesidad de descomposición mental de tareas en comandos múltiples con una sintaxis compleja permite la eficiente utilización del sistema a novatos y avanzados. Cada acción produce un resultado claro para el dominio de la tarea que es inmediatamente visible y comprensible; los tiempos de respuesta son típicamente cortos (la respuesta, aunque el resultado pueda tardar un poco).

Además la proximidad de la tarea con la sintaxis de la acción reduce la tensión y la carga cognocitiva en la resolución de los problemas. El nivel semántico es el que domina en la mente del usuario reduciendo la distracción de tratar con la semántica y sintaxis de la computadora. Verdaderamente, la manipulación directa aplicada a simulaciones interactivas resulta ser el corazón de los programas educativos exitosos.

7.6 Diseño de la Pantalla de Información

7.6.1 Organización de la pantalla

Como ya se ha mencionado anteriormente, la correcta visualización de los objetos en pantalla es muy importante para lograr efectividad en la interfase. Sigue

una guía para la organización de la pantalla que se debe a Smith Y Mosier [SHNE87]:

1. **Consistencia de la Información**. Este es un principio que es frecuentemente violado, pero es fácil de arreglar. Debe existir una estandarización de terminología. abreviaciones, formatos, etc, que deben ser controlados por un diccionario y de ser posible incluído en línea.
2. **Formato Adecuado**. La información debe desplegarse en forma ordenada y cuidadosa; en columnas correctamente tabuladas, los letreros justificados a la izquierda, los números enteros justificados a la derecha, números decimales alineados por el punto decimal, los encabezados y etiquetas deben ser comprensibles, etc
3. **Mínima Carga en la Memoria del Usuario**. No debe obligarse al usuario a recordar información de una pantalla a otra. Las tareas deben estar organizadas de tal forma que se requiera una mínima cantidad de pasos para terminarlas, disminuyendo la posibilidad de que se olvide alguno de ellos.
4. **Compatibilidad entre entrada y despliegue de información**. El formato con que la información es puesta en pantalla debe estar claramente ligado con el formato con que se alimenta de información el sistema.
5. **Flexibilidad de la información en pantalla**. El usuario puede obtener la información en la forma más conveniente para la tarea en la que se está enfocando.
6. **Orientación**. Si la información se presenta en varias secciones, proveer un mecanismo que ayude a saber qué tanto se ha avanzado y cuánto falta; ejemplo, numeración de páginas : pág 14/34.
7. **Discriminación**. Presente la información sólo si es de utilidad para el usuario.
8. **Representación**. Presente la información de manera numérica sólo cuando es necesaria o útil, de ser posible utilice dibujos y esquemas.
9. **Color**. Utilice color sólo si ayuda en algo. Uselo a discreción.

7.6.2 Llamando la Atención del Usuario

Siempre hay situaciones de excepción cuando se desea llamar la atención inmediata del usario, o información crítica que no debe pasar inadvetida. En tales circumstancias use las siguientes reglas:

1. **Intensidad**: Dos niveles. Más de dos niveles de intensidad son muy difíciles de notar.
2. **Resaltar**: Subrayar, utilizar cajas o ventanas, apunte con una flecha, utilice un indicador como asterisco, un punto grande, guión o X.
3. **Tipos de Letra**: No más de tres.
4. **Tamaño de Letra**: No más de 4.
5. **Video Inverso**: Sólo cuando sea necesario.
6. **Blinking**: Use con precausión (2-4 Hertz)

7. **Color**: Use a lo más cuatro colores agradables como estándard. Otros colores se deben reservar para uso ocasional. Si utiliza dibujos, utilice el color de una forma más liberal. pero sólo en los dibujos.
8. **Audio**: Utilice sonidos suaves para dar retroalimentación positiva y sonidos desagradables para condiciones de emergencia.

En general, no debe abusarse de esta técnicas: De hecho, sólo en raras ocasiones se necesita incluir más de tres. Los usuarios novatos no necesitan de una pantalla congestionada, sino de pantallas simples, bien organizadas y con etiquetas claras para guiar sus pasos. Los formatos de pantalla deben ser probados para verificar si son comprendidas por el usuario.

7.7 Manejo de errores

La prevensión de errores es uno de los objetivos que el diseñador de sistema debe siempre perseguir. Un alto porcentaje de errores reduce la efectividad del programa y puede conducir a pérdidas de valiosa información, además de que cohiben al usuario. Adicionalmente, en los casos en que no es posible evitarlos, debe proveerse de un sistema adecuado de mensajes que no confunda al usuario y que le indique cómo corregir el error. Claramente, para que ésto de resultado, la forma en que se corrijan los errores debe ser sencilla e intuitiva.

7.7.1 Prevención de Errores

Síganse estas recomendaciones sencillas para minimizar la ocurrencia de errores:

1. Organizar menús funcionalemte.
2. Diseñar los comandos de forma distintiva, clara e intuitiva.
3. Obligar al usario verificar sus intenciones y sus acciones cada vez pueda cometer un error irreversible.
4. Evitar simpre la existencia de modos de operación.
5. Retroalimentación constante sobre el estado del sistema.
6. Complete pares como comillas y parentesis
7. Provea macros para secuencias de comandos difíciles de memorizar o peligrosas.

7.7.2 Mensajes de error

Los mensajes normales y las respuestas del sistema tienen gran influencia en la percepción que se logra, pero los mensajes de error son críticos. Si los mensajes de error son demasiado confusos, el usuario puede confundirse. Si su lenguaje es

impetuoso, se puede llegar provocar un sentimiento de ansiedad. Los mensajes de error deben tener las siguientes características:

Claridad. Los mensajes de error pueden ser demasiado genéricos, como en "comando no entendido" o "Error de sintaxis", o demasiado técnicos, como en "FAC RJCT 004004400", o error "2345a". Siempre debe tenerse en cuenta al usuario novato cuando se diseñan estos mensajes porque en muchas ocasiones la imágen que se lleva un usuario de un sistema tiene más que ver con los sucesos generados cuando las cosas salen mal, que cuando salen bien.

Especificación. Especificación se refiere a la información proporcionada que indique qué salió mal, cuando salió mal y porqué salió mal. Esta información es importante para la correción de errores y da confianza al usuario.

Información constructiva y positiva. Es este aspecto, se trata de informar al usuario que se ha cometido un error indicando la gravedad pero sin asustarlo e indicar la forma de corregir el error.

Formato Físico Apropiado. Esto implica que las formas gramaticales, terminología y abreviaciones deben ser consistentes y claras. Es decir, que las palabras utilizadas en el mensaje de error no sólo deben ser entendidas, sino que no deben sacar al lector fuera del contexto. Si el usuario está utilizando en programa de publicación, lo correcto es que los mensajes contengan terminología de publicación y no de compiladores y bases de datos. Otro punto importante es el formato visual apropiado. Es deseable que el mensaje de error aparezca en una ventana (y siempre en el mismo lugar) que ayude a identificarlo, pero que no obstruya la vista del área de trabajo.

Diferentes Niveles de Mensajes. Debe dejarse la opción al usuario del nivel de mensajes que desea. Mientras un usuario novato puede encontrar muy útiles los mensajes con la mayor cantidad de información posible, un usuario experto puede encontrar estos mensaje demasiado distrayentes y escoger un nivel con mensajes más breves.

7.8 Conclusión

La calidad de la interfase es de primordial importancia en el desarrollo de un programa educativo, porque ésta es la que mayor peso tiene sobre la opinión final que se lleva el estudiante que lo utiliza. Si los comandos no son claros, el individuo puede sentirse incómodo por no poder recordar los. Si se incurre en errores serios muy seguido, se puede sentir inhibido ante la posibilidad de parecer un tonto o perder información derivada de horas de trabajo. Si el sistema no está hecho para ser interactivo, el estudiante puede aburrirse y dejar de poner atención, etc.

Para poder desarrollar una interfase adecuada para los programas educativos, es importante seguir una metodología de diseño ordenada y científica consistente de tres etapas: Análisis, Desarrollo y Evaluación. Se inicia con varios análisis donde se toman en cuanta una serie de factores, como el nivel de maestría del usuario en el tema (novato, intermedio y avanzado), la cultura, el sexo y la edad. Parte medular del fase de análisis son los objetivos del sistema y las tareas que va a llevar a cabo. Partiendo de un análisis de tareas, se pueden determinar aquellas tareas relacionadas con el nivel semántico del estudiante, aquellas en las que debe de concentrar, y aquellas de nivel sintáctico; tareas que deben ser lo más transparente posible al usuario.

Una manera en que se puede lograr un alto de grado de transparencia de la interfase es utilizando el paradigma de "Manipulación Directa", en el que se escogen los elementos de interés y se indican las acciones que se deben de llevar a cabo.

Otro factor muy importante es la visualización de la información. Existen varias propiedades de las imágenes que se pueden utilizar para la codificación de variables de información: El color, el tamaño, la geometría, el movimiento, la discretización, etc.

Por ultimo, se ofrecen una serie de reglas sobre cómo llevar el diálogo Hombre-Máquina:

1. **Consistencia.** Debe tratar de uniformizar todo el ambiente de trabajo: Los comandos, la terminología, las pantallas, los menús, etc
2. **Atajos.** Los menús son buenos para los principiantes, pero usuarios más avanzados los pueden encontrar irritantes. Provea de teclas de atajo, y macros para realizar tareas comunes o que requieren de mucho menús.
3. **Retroalimentación.** Para cada acción del usuario debe existir una reacción que se indique que la información se está procesando. Mantenga los tiempos de respuesta cortos, y los mensajes de sistema y error claros.
4. **Manejo de Errores.** A toda costa debe evitarse que se puedan cometer errores serios por accidente, aún a costa de quitarle el control al usuario. En lo posible provea de mecanismos sencillos para la corrección de errores y preferentemente, que no se tenga que repetir toda la secuencia de acciones.
5. **Control.** Siempre debe ser el estudiante el que inicie las acciones en el programa. Evite someterlo a sesiones largas y tediosas. Evite acciones sorprendentes del sistema. Automatice sólo los procedimientos más largos y peligrosos.
6. **Carga en Memoria.** Reducir lo más posible la carga en la memoria corta del estudiante. Las pantallas deben estar diseñadas en forma sencilla. Ruducir el movimiento entre distintas ventanas que se oculten información mutuamente. Se deben estandarizar las funciones en todas las pantallas y de ser posible, debe proveerse ayuda en línea.
7. Dar sensación de **progreso** y formas de localización dentro del sistema.

CAPITULO VIII

DESCRIPCION FUNCIONAL DE NEWT

8.1 Introducción

Newt es una aplicación que se desarrolló para probar la validez de los conceptos que se han presentado en los capítulos anteriores. Su función principal es la de ayudar en el aprendizaje del pensamiento algorítmico y lenguajes de programación básicos. Newt fué programado en el lenguaje gráfico educativo cT (Carnegie Tutorial) desarrollado por la universidad Carnegie Mellon en Pittsburg. El programa está totalmente orientado al usuario. Los comandos más comunes, como los de inserción y edición de instrucciones, y ejecución del programa están representados en botones que siempre están a la mano del usuario. Otros comandos menos comunes están en menús y submenús de muy fácil acceso cuyos encabezados son bastante claros. La información está lógicamente ordenada en tres ventanas: Ventana de Edición de Diagrama de flujo, Ventana de Listado y Ventana de Ejecución.

Newt no es un tutorial sobre diagramas de flujo sino un medio integrado, donde el alumno será expuesto a la tecnologia computacional y al pensamiento algorítmico de resolución de problemas sin tener que confrontar ninguna otra complejidad. El manejo de errores es uno de los aspectos en que se ha puesto mayor énfasis, debido a que el programa está diseñado para la utilización de personas novatas en la programación. El objetivo es que el programa pueda ejecutarse en cualquier momento, por esto, se revisa la sintáxis y la semántica de las acciones del usario constantemente. Si se escoge una instrucción, se verifica que la ejecución de esa instrucción no lleve a ningún error; por ejemplo el poner un ENDIF antes de un IF[1], lo que corresponde al nivel semántico. Tambén se verifica la sintaxis, cualquier error en este aspecto genera un mensaje que indica claramente cuál es el error, y cómo arreglarlo. La siguiente sección discute a detalle que instrucciones se pueden utilizar con Newt y cómo se utilizan.

[1] Para una descripción completa de las instrucciones ver sección 8.2

8.2 Un Nuevo Lenguaje para Principiantes

8.2.1 El Lenguaje Pico LEPP

LEPP es un lenguaje de programación que ha sido desarrollado especialmente para este proyecto. LEPP son las iniciales de Lenguaje Estructurado Para Principiantes. La idea de este lenguaje nació de la necesidad de etiquetar y especificar las instrucciones con las que se contruye el diagrama de flujo sin depender de un lenguaje de los ya establecidos. Las justificaciones por las que se desea utilizar ningún lenguaje ya conocido son dos:

1. El proyecto intenta inculcar en el intelecto de los estudiantes el pensamineto algorítmico y estructurado sin importar el lenguaje. Esto hace posible que la persona pueda después migrar con mucha facilidad entre varios lenguajes apreciando las ventajas y desventajas de cada uno.

2. El tomar lenguajes como Pascal o C para principiantes es algo que puede ser demasiado pesado para el estudiante. Sin embargo, existe en la actualidad ningún lenguaje para principiantes que cumpla los requerimientos de los lenguajes modernos. Las opciones más cercanas son el BASIC y el Logo. Estos tienen la principal desventaja de no ser estructurados. Muchos maestros han expresado la opinión de que si los alumnos tuvieran la opción de saber Basic o nada antes de llegar a cursos computacionales donde se enseñana lenguajes como Modula-2, Pascal o C, prefieren que no sepan nada. Según la opinión de varias personas involucradas en la instrucción utilizando computadoras, la exposición previa a la tecnología es un factor determinante en el desempeño[Whit,1990]. Por lo tanto, no saber nada es inaceptable. Sin embargo, aparentemente saber BASIC también lo puede ser. De ahí la necesidad de crear un nuevo lenguaje orientado a principiantes.

Pico LEPP es pues, el lenguaje LEPP en estado lactante. Se espera que eventualmente nazca un LEPP completamente implementado con el que se puedan crear aplicaciones bastante complejas. Por el momento, Newt conoce Pico LEPP, que quiere decir pequeño LEPP.

Otra opción hubiera sido tomar un conjunto reducido de instrucciones de alguno de los lenguajes ya conocidos. Sin embargo, la correspondencia requerida de uno a uno con el diagrama de flujo pudiera no lograrse completamte. Por ejemplo, en Pascal la instrucción END se utiliza para terminar todas las intrucciones de control de flujo, mientras que en el diagrama de flujo cada instrucción tiene un dibujo diferente con la que se cierra.

A continuación sigue un completa descripción del lenguaje Pico LEPP. En la descripción de la sintaxis se toman varias conveciones:

MAYUSCULAS son las instrucciones del lenguaje.

Negritas significa que la palabra es parte indispensable del comando y que por tanto siempre debe estar presente cuando se utilice la instrucción. Si se programa a través de la interfase gráfica, estas palabras serán automáticamente agregadas al listado por el interpretador de la interfase.

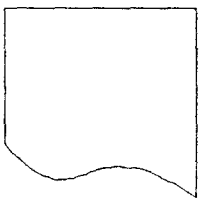
Itálica significa que el símbolo puede tomar diferentes valores dependiendo de la utilización de la instrucción pero que debe estar presente en la utilización del comando.

[] significa que los símbolos dentro de los corchetes pueden o no aparecer en la línea del comando.

El ícono es la representación gráfica de la instrucción dentro del diagrama de flujo.

8.2.2 Las Instrucciones

8.2.2.1 Instrucciones de Entrada y Salida:



1. Sintaxis : **WRITE** ['Texto' ,Variable1 ,Variable2,Variable3,...]

Imprime en la pantalla de ejecución el texto dentro de las comillas sencillas (sin comillas) y el contenido de las variables enlistadas. El texto y las variables tienen que estar separadas por comas.

Ejemplo: WRITE 'Resultado',A,b,a,x

2. Sintaxis: **READ** ['Texto' ,Variable1 ,Variable2,Variable3 ,...]

Imprime el texto entre comillas y después lee valores dados por el usuario en el cursor de la ventana de ejecución y los asigna secuencialmente a cada variable hasta encontrar un CR (Carriage Return). Las variables y el texto van separados por coma.

Ejemplo: READ 'Da x,y,z',x,y,z

8.2.2.2 Cálculo

3. Sintaxis : Assign *Variable=Expresión*

Assign evalúa Expresión y el resultado lo asigna a Variable. Las operaciones válidas en la expresión son:

+ suma
- resta
/ división
* multiplicación
^exponenciación

y una serie de funciones como:

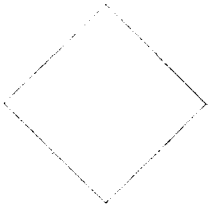
sqrt(var) raíz cuadrada de var
sqr(var) var al cuadrados
sin, cos, tan, etc.

8.2.2.3 Instrucciones de Control de programa o Apertura de Flujo

4. Sintaxis : **IF *Expresión***
[Instrucción1
Instrucción2
]
ELSE
[Instrucción 1
Instrucción 2
]
ENDIF

Ejemplo:
```
IF x < 10
        WRITE 'Mayor que 10'
ELSE
        WRITE 'Menor que 10'
ENDIF
```

Icono (IF):



Icono (EndIf) :

La expresión es evaluada y si el resultado es verdadero (es decir, diferente de 0), se ejecuta la secuencia de instrucciones hasta encontrar ELSE donde brinca hasta la siguiente instrucción después del ENDIF. Si la expresión es falsa, el control del programa pasa a la siguiente instrucción después de ELSE.

ELSE no tiene ícono puesto que el dibujo del ENDIF se indica para expresar el fin de la sección que se va a ejecutar si la expresión es verdadera (comúnmente llamada sección THEN), inicio de la sección a ejecutar si la expresión es falsa (comúnmente llamada ELSE) y fin de todo el IF.

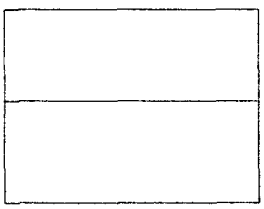
5. Sintaxis : **FOR** ***variable=valor_inicial,valor_final***
[Instrucción1
Instrucción2
.....
]
ENDFOR

FOR asigna a variable valor_inicial y se empiezan a ejecutar secuencialmente las instrucciones que siguen a la línea de FOR. ENDFOR regresa el control del programa a FOR. Cuando variable alcanza valor_final FOR regresa el control del programa a la instrucción que está inmediatamente, después de ENDFOR.

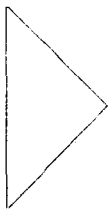
Ejemplo:
```
FOR i:=1,10
        WRITE i
ENDFOR
```

La variable que se utiliza como contador para el FOR debe ser entera.

Icono (For):



Icono (EndFor):

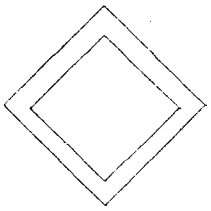


6. Sintaxis: **WHILE** ***Expesión***
[Instrucción1
Instrucción2
....
]
ENDWHILE

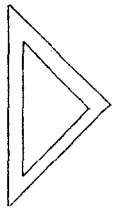
la expresión se evalúa y si es verdadera entonces se ejecuta la secuencia de instrucciones hasta el ENDWHILE. ENDWHILE entonces regresa el control del programa a WHILE. Si la expresión es falsa, el control del programa pasa a la siguiente instrucción después de ENDWHILE.

Ejemplo:
```
WHILE x<10
    WRITE x
    CALC  x:=x+1
ENDWHILE
```

Icono(WHILE):



Icono (ENDWHILE):

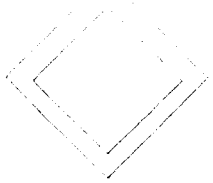


7. Sintaxis: **REPEAT**
[Instrucción1
Instrucción2
....
]
UNTIL ***Expesión***

REPEAT no ejerce ninguna acción mas que pasar el control a la siguiente instrucción. Al llegar a UNTIL, se evalúa la expresión. Si resulta verdadera se pasa el control a la siguiente instrucción después de REPEAT. La expresión marca la condición para fin de ciclo. Si la expresión es falsa, se pasa el control a REPEAT.

Ejemplo:
```
REPEAT
    CALC  x:=x+1
    WRITE x
UNTIL x>=10
```

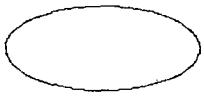
Icono(REPEAT): Icono (UNTIL):



8. Sintaxis : BEGIN
9. Sintaxis : END

BEGIN indica que inicia la ejecución del programa y END la detiene. END es opcional, si no se encuentra ninguna instrucción el programa se detiene sólo. Newt siempre inserta un BEGIN al iniciar un programa.

Icono (BEGIN): Icono (END):



8.2.3 Las Variables

El usuario de Newt, no puede escoger cualquier variable para utilizarla en el programa. Newt tiene una serie de variables que el usuario puede utilizar. Tiene tres tipos de Variables: String, Entero, Flotante.

String: Son aquellas que contiene caractéres alfanuméricos. No se pueden hacer operaciones con ellas. Cuando el valor es asignado a través de un READ o un CALC, éste debe estar entre comillas sencillas. Las variables String son todas mayúsculas:

A, B, C, D, E, F

Ejemplos:

CALC A:='Hola'
READ C -----> > 'Salario' ok

Enteras: Estas variables son las que típicamente se utilizan para contadores:

i, j, k, l, m, n

Son importantes porque son las únicas que se pueden utilizar en el estatuto FOR. Se le pueden aplicar todas las operaciones aritméticas. Las divisiones fraccionarias asignadas a variables entras son redondeadas.

Ejemplos:

```
CALC i:=1
FOR  i:=1,10
READ  i
```

Los valores mínimos y máximos son -32768 a 32767

Flotantes: Estas variables son aquellas que aceptan puntos decimales. Les dice de coma o punto flotante porque el punto decimal puede estar en cualquier parte del número.

La cantidad de cifras decimales y enteras está limitado por la precisión de la máquina, en general son seis dígitos de precisión.

```
CALC   x:=1345.6789534
READ   a:=x/2.3456
```

Además, Newt también puede generar listados en Pico C y Pico Pascal. El nombre de Pico Pascal o Pico C se derivan del parecido que se ha intentado mantener entre las instruciones de los lenguajes originales y los listados de Newt. Como también son conjuntos reducidos de instrucciones se dirá que son hijos de los lenguajes originales. Jamás se ha pretendido que Pico LEPP, Pico Pascal y Pico C tengan o lleguen a tener la flexibilidad y las capacidades de los lenguajes que tratan de emular. El objetivo es proveer a la interfase gráfica de programación de una herramienta con la cual permitir a las personas que están aprendiendo a programar hacer el brinco de una forma poco dolorosa entre el pensamiento algorítmico y la generación de código para programas. Aunque los lenguajes Pico son limitados ya que sólo cuentan con 7 instrucciones efectivas, se ha intentado mantener la filosofía de programación estructurada que tan celosamente siguen lenguajes como Pascal, C o Modula 2.

8.3 Utilización de Newt

8.3.1 La Interfase de Newt

La pantalla de Interfase de Newt está dividida en tres ventanas. Estas son:

Ventana de Diagrama de Flujo. En esta ventana, situada en la parte central de la pantalla, se dibuja el diagrama de flujo del programa que se va creando y donde se editan sus instrucciones.

Ventana de Listado. Esta ventana, en la parte superior derecha, muestra el listado del programa creado por el diagrama de flujo. En esta ventana no se lleva a cabo ninguna acción.

Ventana de Ejecución. Sobre esta ventana, en la parte inferior derecha, se lleva a cabo la interacción entre el programa creado y el usuario. Aquí se muestran los mensajes programados por medio de instrucciones WRITE y se leen los valores de las variables solicitados por medio de instrucciones READ.

Newt también muestra dos menús en pantalla:

Menú de Instrucciones (Etiquetado como Herramientas). Son 14 botones situados en la parte superior izquierda de la pantalla, con los cuales se escogen las instrucciones con las que se construye el programa descritas en la sección anterior.

Menú de Edición. En este menú, situado en la parte inferior izquierda, se señalan los modos con los que se van a hacer cambios al diagrama de flujo: Editar Y Apendizar (EyA), Cortar, Insertar y Mover. Estos se describen detenidamente en la siguiente sección.

Adicionalemente existen otros dos botones, localizados también en la parte inferior izquierda justo debajo del menú de edición, que son importantes:

Botón de Ejecución (con etiqueta "Correr"), marcado con un logotipo en forma de conejo. Con este botón se ejecutan las instrucciones contenidas en el diagrama de flujo.

Botón de Salir. Este botón termina el trabajo de Newt. Es muy importante salir con este botón o con su análogo en el menú de archivos, porque verifica que toda la información se haya guardado antes de terminar Newt, de otra manera se puede perder imformación importante.

Finalmente En la parte superior de la pantalla se puede encontrar la **Barra de Menús**. Esta barra contiene una serie de encabezados de menús de tipo desenrollables: Archivos, Edición, Diagrama, Listado, Ejemplos y Options.

Para accesar los menús desenrrollables se debe apuntar el encabezado del menú que se desea accesar en la barra de menús y oprimir el botón izquierdo del ratón. Alternativamente, se puede oprimir la combinación de teclas "Control"-"m",

y después utilizar las flechas para escoger el menú de interés. Estos menús son explicados en la seción 8.3.8.

Los elementos que se acaban de mencionar se muestran en la figura 16

8.3.2 Creación un Diagrama de Flujo y Ejecución

Para crear un diagrama de flujo, sólo es necesario "oprimir" el botón de la instrucción deseada en el Menú de Instrucciones que se encuentra en la parte izquierda de la pantalla (Fig. 16). Para oprimir el botón, posicione el cursor del ratón sobre el botón en pantalla, y oprima cualquiera de los dos botones físicos del ratón. El botón en pantalla se anima simulando haber sido oprimido. Cada botón está etiquetado con un logotipo y una palabra que indican el tipo de instrucción. Para saber cómo funcionan las instrucciones, se puede recurrir al menú de ejemplos[2]. Este está ordenado según la complejidad de los ejemplos; el ejemplo más sencillo está al inicio del menú (parte superior), y el más complejo será el último (parte inferior). Al escoger las instrucciones, el diagrama de flujo se va generando automáticamente en la Ventana de Diagrama en la parte central de la pantalla (Fig. 16), mostrando el flujo de la ejecución del programa. Al mismo tiempo, también se va generando un listado del programa en la Ventana de Listado, parte superior, derecha de la pantalla. Las figuras dentro del diagrama tendrán la misma forma que en los botones que los generaron. Si se escoge una instrucción no válida, en ese instante, aparece un mensaje de error que indica porqué la instrucción no se puede utilizar. La instrucción no es incluída en el diagrama de flujo.

Después de haber escogido la instrucción, debe llenarse el **argumento**. Es decir, si se escoge un READ, el siguiente paso es indicar qué variables se van a afectar con esa instrucción. Para hacer ésto, se presiona el botón izquierdo del ratón cuando el cursor esté sobre la instrucción que se quiere terminar de especificar dentro de la Ventana de Diagrama. Al escoger la instrucción, aparece la figura de la instrucción aumentada de tamaño, con un cursor de texto esperando a que se entre el resto de la instrucción. Una vez que se hace esto, por ejemplo las variables "a,b,c" (Fig. 17), se oprime la tecla de "Return". Newt analiza el argumento y determina si es válido. Si lo es, desaparece la figura, y se actualizan el diagrama de flujo y el listado (Fig. 18). Si la instrucción no es válida, aparece un mensaje de error (Fig 19), donde se indica cuál es el error y cómo corregirlo. Vuelve a aparecer la figura de la instrucción con el cursor de texto para modificar el argumento.

---

[2] En un futuro, Newt incluirá un tutorial y ayuda en línea.

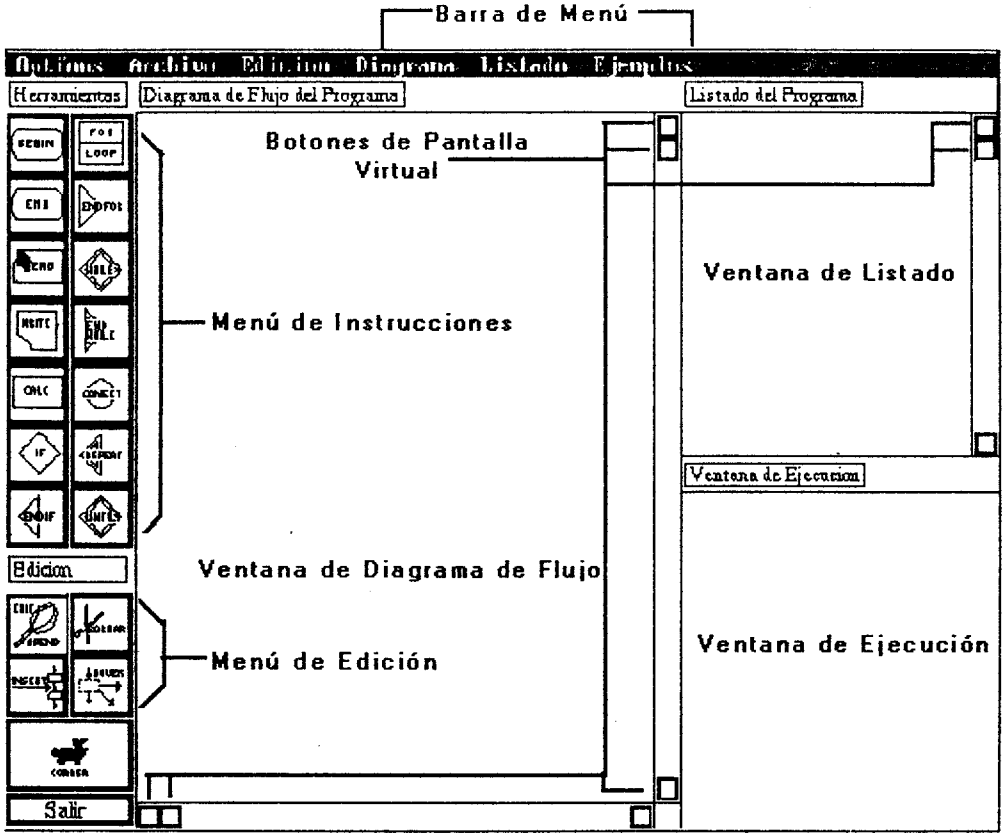


Fig. 16 Organización de la Pantalla de Newt

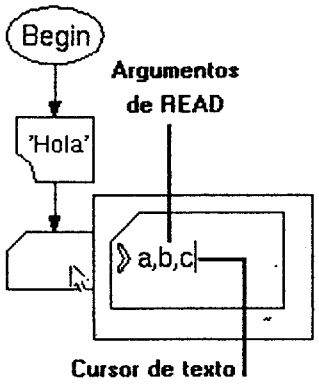


Fig. 17 Editando una Instrucción

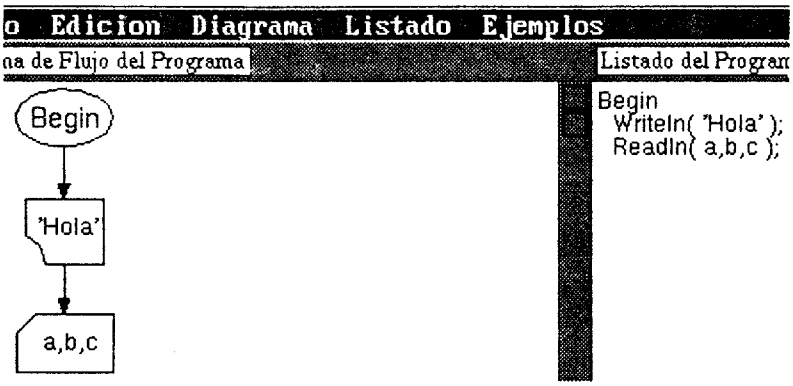


Fig.18 Diagrama de Flujo y Listado

Una instrucción que no contenga argumentos no genera errores, sino que es ignorada. Si se desea cambiar el contenido del argumento, seleccionece la instrucción dentro del diagrama de flujo con el ratón. Hecho esto, aparece nuevamente la figura agrandada de la instrucción con el argumento. El alumno puede modificar el contenido utilizando la tecla "Back Space"[3], o el cursor. Es importante notar que las flechas de cursor no se pueden utilizar para editar el argumento.

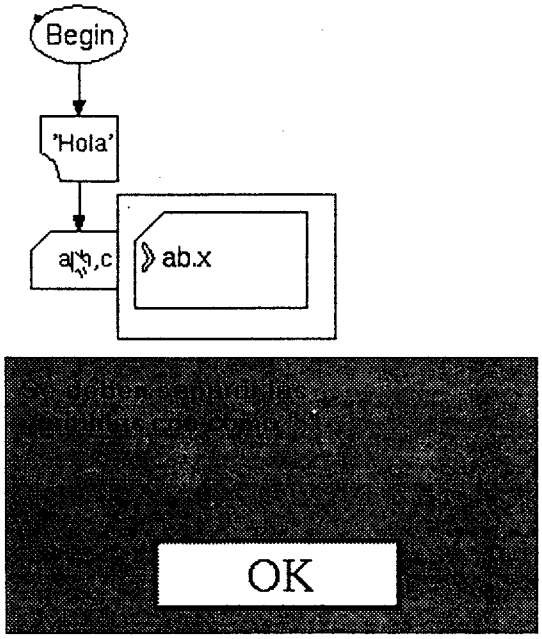


Fig. 19 Mensaje de Error del Sistema

Y esto es todo lo que se necesita para poder ejecutar el programa. Para hacerlo, seleccionece el botón de "Correr", etiquetado además con un logotipo en

[3] Típicamente indicada con una flecha hacia la izquierda en el teclado.

forma de conejo. Los resultados de la ejecución aparecerán en la ventana de ejecución en la parte inferior derecha de la pantalla.

8.3.3 Edición del Diagrama de Flujo

El alumno no puede escoger la localización física de la instrucción dentro del diagrama de flujo al crearse la instrucción. Esto es trabajo para Newt. Sin embargo, sí puede cambiar la posición una vez incluída en el diagrama de flujo. Para ésto debe accionar el botón correspondiente en el Menú de Edición el la parte inferior al Menú de Instrucciones. Estos botones cambian el modo de edición teniendose la opción entre cuatro de ellos: Edición y Apendizar, Insertar, Cortar y Mover. El modo de edición será indicado por la forma del cursor. Los botones también contienen logotipos y etiquetas indicativos de la acción que generan. A continuación sigue una descripción de ellos:

8.3.3.1 Edición y Apendizar (EyA)

Este es el modo por omisión (default). Si se oprime algún botón del Menú de Instrucciones, la instrucción escogida es apendizada al Diagrama de Flujo, o sea que aparece al final del listado. Si se selecciona alguna instrucción en la Ventana de Diagrama, se edita el argumento de la función. Este es el modo de operación por omisión y se identifica porque el cursor del ratón tiene la forma de flecha normal.

8.3.3.2 Insertar

Al oprimir algún boton del Menú de Instrucciones con el botón IZQUIERDO, se asume que se desea insertar la instrucción antes de la instrucción representada por el dibujo resaltado y así aparece dentro del listado. Al seleccionar la instrucción con el botón DERECHO, se inserta la instrucción después de la instrucción resaltada. Si se selecciona algúna instrucción dentro de la Ventana de Diagrama, se asume que se desea que tal instrucción sea el lugar donde se va a insertar una instrcción. Entoces se quita el resaltado de la instrucción anteriormente resaltada y se resalta la escogida. Cuando Newt se encuantra en modo de inserción, el cursor de ratón cambia de forma a una "I" muy grande.

8.3.3.3 Cortar (Borrar)

Al señalar una instrucción dentro de la Ventana de Diagrama, la instrucción es eliminada. Se actualizan el listado y el diagrama de flujo. Este modo no tiene ninguna acción sobre el Menú de Instrucciones. Durante este modo, el cursor toma la forma de unas pequeñas tijeras.

8.3.3.4 Mover

Para mover una instrucción, coloque el cursor del ratón sobre el dibujo que la representa dentro de la Ventana de Diagrama y oprima el botón IZQUIERDO. Aparece un "esqueleto" de la instrucción junto al cursor indicando que la instrucción esta lista. NO suelte el botón del ratón. Mueva el cursor y apunte a la instrucción ANTES de la cual se desea mover la instrucción original. Al soltar el botón, se borra la instrucción en su posición original y se inserta con todo y agumento en la posición escogida. Si se repite la misma acción con el botón DERECHO, la instrucción es movida después de la posición donde se soltó el botón. Igual que el modo Cortar, no existe efecto alguno al señalar instrucciones del Menú de Instrucciones, ambos modos sólo actúan sobre la Ventana de Diagrama. En el modo de Mover, el cursor toma la forma de cuatro flechas cruzadas en forma de cruz.

Para regresar al modo EyA, el modo normal para la creación del diagrama, oprima el botón correspondiente.

Si alguna de estas acciones ocaciona un error, como por ejemplo, el poner un ELSE antes de un IF, se genera un mensaje de error y el diagrama de flujo se restituye a su estado antes de la acción.

8.3.4 Creación de un Programa Sencillo

Un ejemplo trivial pero ilustrativo es uno como el que sigue:

```
1. WRITE 'Nombre?'
2. READ A
2. WRITE 'Hola ',A
```

Para crear este pequeño programa, se siguen los siguientes pasos:

1. Seleccionar el botón para la instrucción WRITE, en el menú de instrucciones
2. Seleccionar el botón para la instrucción READ, en el Menú de Instrucciones
3. Seleccionar el botón para la instrucción WRITE, en el Menú de Instrucciones
4. Seleccionar la primera instrucción WRITE de la Ventana de Diagrama.
5. Al aparecer la figura de la instrucción WRITE con el cursor de edición de texto se teclea exactamente lo Siguiente: 'Nombre?'
6. Ahora se selecciona la instrucción siguiente, READ. Al aparecer la figura de la instrucción READ y el cursor de edición de texto se tecles exactamente lo siguiente:A
7. Se selecciona la última instrucción, WRITE y al aparecer el cursor de edición de texto teclear exactamente: 'Hola',A
8. Puede Observarse cómo se han actualizado el listado y el diagrama de flujo

9. Ahora Oprímase el botón de "Correr"
10. En la ventana de ejecucción aparecerá lo siguiente:

```
- Programa: newtfile -
Nombre?
>|
```

Esto indica que el programa está esperando una respuesta del usuario. Se teclea cualquier cosa y el programa responde:

```
- Programa: newtfile -
Nombre?
  Raúl
Hola Raúl.
```

8.3.5 Notas avanzadas sobre la creacción del diagrama de flujo

Aparte de las tres instrucciones básicas READ, CALC y WRITE, existen una series de instrucciones de control de ciclo que van por parejas como el FOR-ENDFOR, el WHILE-ENDWHILE y el REPEAT-UNTIL. Para utilizar estas instrucciones, séase REPEAT, se oprime el botón correspondiente. Después se oprimen todas las instrucciones que van dentro del ciclo. Para cerrar el ciclo se oprime la tecla UNTIL. Hasta que se oprimió UNTIL se dice que el REPEAT estuvo abierto. Las instrucciones de control son llamadas de apertura o "Branching".

Un caso muy especial es el IF. El IF no es un par de instrucciones sino una triada. Incluye, IF, ELSE y ENDIF. Para crear un IF, se oprime primero esta tecla. Después se incluyen todas las instrucciones que se ejecutan si el argumento es verdadero. Esta es la sección del IF que comúnmente conocida como THEN[4]. A continuación se oprime el Botón de ENDIF, y esto indica que vienen las instrucciones que se ejecutan si el argumento es falso, es decir que no se crea un ENDIF, si no un ELSE. Aquí el IF sigue abierto. Finalmente, cuando se terminan de oprimir todas las instrucciones dentro del ELSE, el IF se cierra oprimiendo por segunda vez el botón ENDIF, donde ahora sí se genera un ENDIF. Si se desea crear un IF donde sólo se tienen planeadas instruciones si el argumento es verdadero (el THEN); se abre el IF, se seleccionan las instrucciones dentro del IF, y se selecciona el ENDIF dos veces seguidas.

La razón de ser de un botón para acciones que en todos los leguajes requieren de dos instrucciones, ELSE y ENDIF, es que en el diagrama de flujo queda claro que el THEN y el ELSE terminan en el mismo punto. Se cree que es conveniente

[4] Pero tal vocablo no forma parte del glosario del LEPP

que los estudiantes piensen en términos de la construcción de un diagrama de flujo y no de un texto generador de un programa computacional, aunque deben entender la relación.

8.3.6 Creación de Programas Avanzados

Se podría considerar como avanzado un programa que altera el curso de la ejecución de acuerdo a las variables de entrada, o sea, aquellos programas que utilizan las instrucciones de apertura, como IF, FOR, WHILE y REPEAT. La creación de un programa que utilice cualquiera de las últimas dos es muy sencilla y se ilustra a continuación:

Programa:

```
BEGIN
  CALC i:=1
  WHILE i<=10
     WRITE i
     CALC i:=i+1
  ENDWHILE
END
```

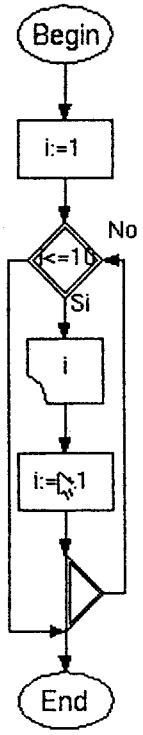


Fig. 20 Usando WHILE

Para crear este programa se deben seguir los siguientes pasos:

1. Escójase la instrucción CALC del menú de instrucciones.
2. Abrase un WHILE utilizando el botón correspondiente del menú
3. Añádanse la instrucciones que van dentro del WHILE: WRITE y CALC
4. Ciérrese el WHILE escogiendo la instrucción ENDWHILE
5. Ciérrese el programa escogiendo END
6. Edítense las instrucciones CALC (primera), WHILE, WRITE y CALC (segunda), señalándolas del la ventana de edición. Insertando el texto correspondiente:

Para CALC; i:=1
Para WHILE; i<=10
Para WRITE; i
y para CALC; i:=1+1

7. Oprima el botón "Correr"

El diagrama de flujo queda como el de la figura 20.

La ejecución del programa se ve en la pantalla de ejecución como sigue:

```
- Programa: newtfile -
1
2
3
.
.
9
10
- Fin: newtfile -
```

Y de la misma forma sería para las instrucciones FOR-ENDFOR y REPEAT-UNTIL.

La instrucción IF requiere de un poco más de trabajo. Por ejemplo un programa como el siguiente :

Programa:

```
BEGIN
    READ  x
    IF  x>10
        WRITE 'Mayor a10'
    ELSE
        CALC   x:=2*x
        WRITE 'El doble:',x[5]
    ENDIF
END
```

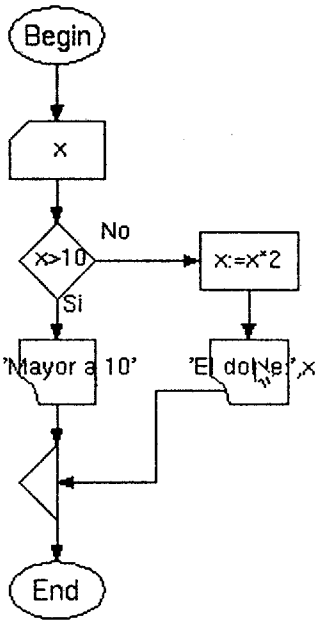


Fig. 21 Usando IF

Un programa así requiere de los siguientes pasos:

1. Escoja la instrucción READ
2. Abra el IF
3. Escoja la instrucción WRITE que va dentro de la sección THEN (evaluación de la condición verdadera) del IF

[5] Los argumentos demasiado largos se salen de la figura de la instrucción que las contienen y no se ven muy bien. Su uso no se recomienda

4. Oprima el botón de ENDIF. Aparece en el listado un ELSE
5. Escoja las instrucciones correspondientes a la sección ELSE del IF (evaluación negativa de la condición)
6. Cierre el IF con oprimiendo por segunda vez el ENDIF. Aparece en el listado un ENDIF
7. Cierre el programa oprimiendo el botón END
8. Escoja las instrucciones que lleven argumentos de la ventana de diagrama y proporciónelos
10. Ejecute (Botón "Correr")

Como se podrá notar, cada vez que se abre un IF, se debe oprimir dos veces el botón de ENDIF. La primera vez indica el fin de la sección THEN e inicio de la sección ELSE. El segundo indica el fin de la sección ELSE y fin del IF. El diagrama de flujo debe quedar como el de la figura 21.

Ahora supóngase que se desea un programa como el que sigue:

```
BEGIN
   READ  x
   IF  x>10
      WRITE 'Mayor'
   ENDIF
END
```

1. Sigánse los pasos del 1 al 3 del programa anterior, hasta el WRITE
2. Ahora Oprima el botón de ENDIF. Aparece un ELSE en el listado
3. Vuélvase a oprimir el botón ENDIF. Como no se incluyó ninguna instrucción para el ELSE, éste desaparece del listado y sólo se ve el ENDIF
4. Oprima el END
5. Llénense los argumentos necesarios
6. Ejecute

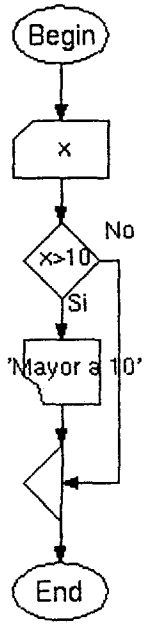


Fig. 22 IF sin ELSE

El diagrama de flujo queda como el de la figura 22.

8.3.7 Notas avanzadas sobre edición del diagrama de flujo

1. No existe opción de copiar. Esta es una implementación ya dentro de la agenda de mejoras, pero se puede lograr muy fácilmente. Cambie a modo de Inserción e inserte una instrucción de la misma categoría que se desea copiar en el lugar donde deba ir, o simplemente selecciónela del Menú de Instrucciones si la desea al final del diagrama de flujo. Cambie a modo EyA y seleccione la instrucción que quiere copiar. Oscurezca con el ratón el argumento y seleccione "Copy" del menú de "Options" en la parte superior derecha en la Barra de Menús. Después seleccione la instrucción recién creada, aparece el cursor de edición de argumento. Seleccione del menú "Options" la opción "Copy". Aparentemente la secuencia de acciones es complicada pero es realmente muy sencilla, aunque no muy intuitiva. La razón por la que esta opción no fué implementada desde un principio es que es muy raro que se necesite duplicar una instrucción en un programa con todo y argumento. Mucho menos si es un programa básico.

2. Editar las instrucciones es muy sencillo y directo. Sin embargo existe una instrucción que puede causar confusión. Esta es el ENDIF. Cortar o Insertar un ENDIF casi nunca causa problemas y no es posible Editarlos porque no llevan argumentos. Sin embargo, tratar de mover un ENDIF puede resultar un dolor de cabeza, debido que la misma figura representa dos instrucciones, el ELSE y el ENDIF correspondientes al IF. Si al intentar mover una de estas instrucciones se escoge una posición final que esté abajo de la posición ELSE en el listado (Fig. 23a, 23b, 24a, 24b), se asume que se desea mover el ENDIF. Si en cambio se escoge una posición superior, no es el ENDIF el que se va a mover puesto que esto es un error seguro[6], sino el ELSE, dejándose el ENDIF en su lugar original (Fig 25a, 25b, 26a, 26b).

3. También el utilizar un ENDIF como posición final en una acción de mover o insertar puede haber alguna confusión, puesto que en estas operaciones se asume como posición final el ELSE y no el ENDIF. Esto con la finalidad de proveer una forma sencilla de poder agregar instrucciones dentro de la estructura IF-ELSE-ENDIF. Si se inserta o se mueve una instrucción indicando "antes", esta se inserta antes del ELSE, se indica "después" se inserta después del ELSE, quedando la figura dibujada dos columnas más allá de donde se indicó.

---

[6] Claramente la secuencia IF..ENDIF..ELSE es incorrecta

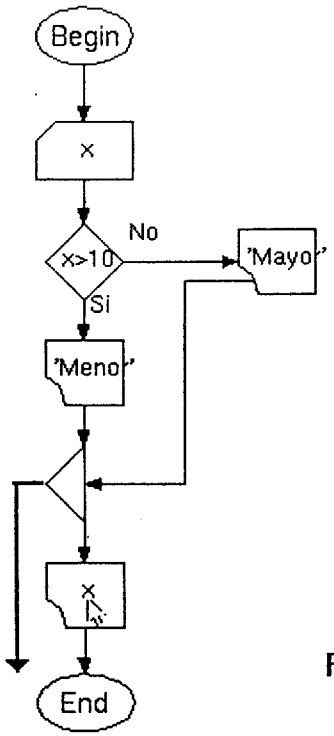


Fig. 23a Mover el ENDIF hacia abajo

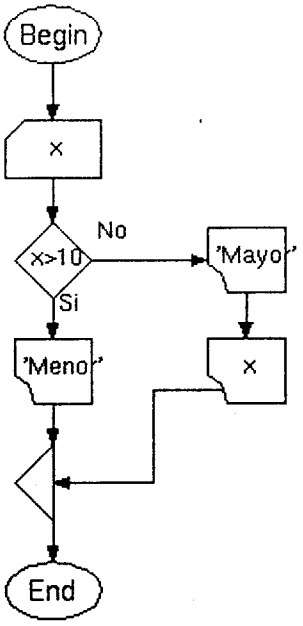


Fig. 23 b WRITE x queda entre ENDIF y ELSE

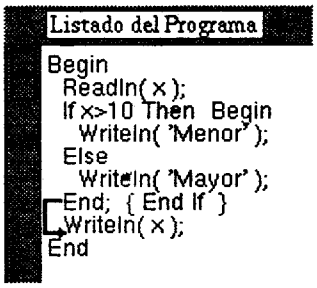


Fig. 24a

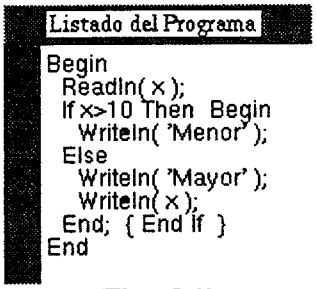


Fig. 24b

Es equivalente a mover END hacia abajo en el listado

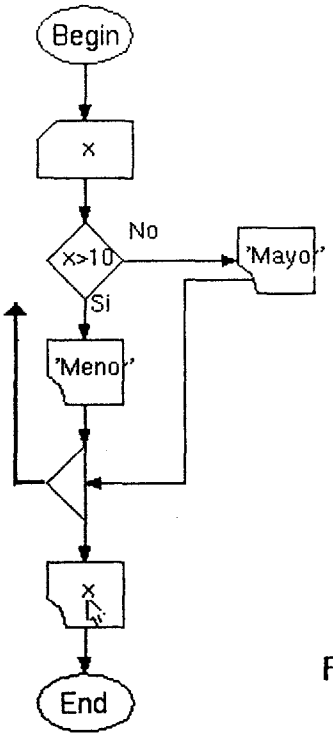


Fig . 25 a Mover ENDIF hacia arriba

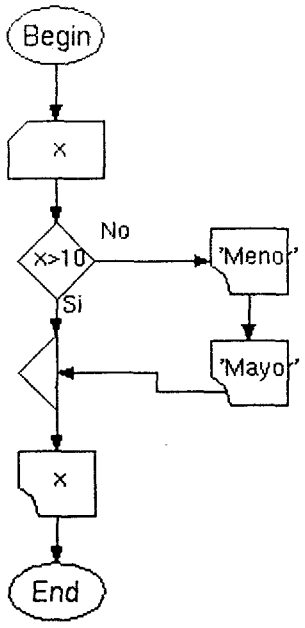


Fig. 25 b WRITE 'Menor' queda entre ELSE y ENDIF

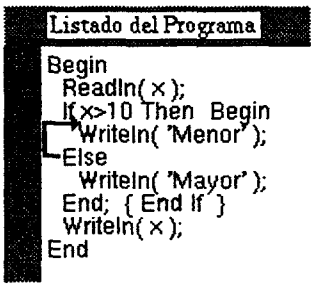


Fig . 26 a

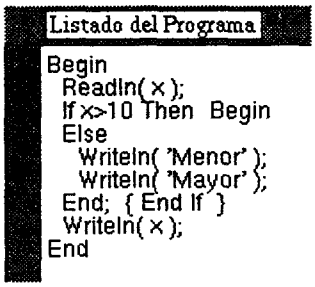


Fig . 26 b

Es equivalente a mover ELSE hacia arriba

5. Como ya se indicó, existen cuatro instrucciones que se consideran que se abren: IF, FOR, WHILE y REPEAT. Para borrar alguna de ellas, empiece borrando la instrucción de cerradura correspondiente, estas son, ENDIF, ENDFOR, ENDWHILE y UNTIL. Por ejemplo, si se borra primero el FOR, se deja un ENDFOR en el diagrama. NEWT detecta el ENDFOR sin FOR abierto, despliega un mensaje de error y aborta la acción de borrado. Con el IF, debe borrarse el ENDIF, luego el ELSE y por último en IF. Además, todo el programa se encuentra abierto debido el BEGIN inicial hasta que se crea el END. Si se desea

borrar una instrucción de apertura una vez que ya que se incluyó el END se va a originar un error porque NEWT va a detectar el END antes de que todas las instrucciones estén cerradas y también va a abortar el borrado. Dado que un programa puede ejecutar perfectamente bien aún si no se ha uncluído un END al final, esta instrucción no debe añadirse hasta que el programa está terminado y probado.

Estas "idiocincracias" de Newt, nunca derivan en errores serios, y los mensajes que Newt genera son suficientemente claros para que el usuario rápidamente entienda el error, lo corrija, lo recuerde y evite.

8.3.8 La Barra de Menús

La Barra de Menú es la que muestra los encabezados de los menús que contiene opciones menos usadas en al utilizar Newt. Estos menús son:

8.3.8.1 Archivo

Todo Nuevo. Limpia todas las ventanas e inicializa el estado del sistema. Si el programa que se estaba construyendo antes de escoger esta opción no ha sido guardado, Newt pregunta si se desea grabar en disco. Si se responde afirmativamente, la información queda guardada en un archivo.

Abrir. En esta opción, Newt pide el nombre del archivo que se quiere leer. Si el archivo existe y está en el formato de Newt, el diagrama de flujo contenido en él se carga y se actualiza el listado. Si se hicieron cambios en el archivo con el que se estaba trabajando antes de llamar este comando, Newt pregunta si se desea salvar la información en disco. Si se responde afirmativamente la información queda guardada en un archivo.

Grabar. Si se selecciona esta opción, Newt guarda el programa con el cual se está trabajando en un archivo. Si el programa no ha sido grabado anteriormente Newt pregunta el nombre del archivo en el que se desea grabar. Newt contiene un archivo por omisión que se llama newtfile. Si el programa ya ha sido grabado, Newt utiliza el archivo que ya se ha escogido.

Grabar como. Esta opción es la misma que la opción Guardar, excepto que siempre se pregunta el archivo en el que se desea grabar. Esta opción es útil cuando se desea guardar respaldo de un programa con diferente nombre.

Salir. Esta opción es análoga al botón etiquetado como Salir en pantalla. Si se han hecho cambios en el programa que no han sido grabados, Newt pregunta si se desea grabarlos. Si se responde afirmativamente la información queda guardada en un archivo.

8.3.8.2 Edición

Este menú contiene cinco opciones:

Las primeras cuatro; **Editar**, **Insertar**, **Cortar** y **Mover**, son completamente análogas a los botones correspondientes en el Menú de Edición en Pantalla. Al escogerse alguna de estas opciones, se oprime el botón correspondiente en pantalla y se cambia el cursor.

La opción de **Deshacer** aparece cuando se ha efectuado algún cambio en modo cortar, mover o insertar. En modo editar esta opción no está disponible. Deshacer restituye el diagrama de flujo al estado en que se encontraba antes de hacer el cambio. Cuando se regresa al modo Editar y Apendizar la opción se borra del menú.

8.3.8.3 Diagrama

Todo Nuevo. Tiene el mismo efecto que la opción Todo Nuevo incluída dentro del menú de Archivo.

Reducir (1/2). Al seleccionar este comando, las dimensiones las figuras del diagrama de flujo se reducen a la mitad. Esta opción es muy útil cuando el diagrama de flujo no cabe en la pantalla. Cuando se llama este comando, Reducir (1/2) es sustituído en el menú por **Agrandar (x2)**, que es la acción inversa de Reducir (1/2), es decir, que el diagrama vuelva a tomar sus dimensiones originales.

Correr. Esta opción es análoga al botón de correr en pantalla. Cuando se ejecuta un diagrama de flujo, ya sea por botón o por menú, la opción de Correr es sustituída en el menú Diagrama por Alto. Esta opción es muy útil cuando se crea un programa que se cicla indefinidamente. Cuando algún diagrama se está ejecutando, en ocasiones es difícil encontrar el ratón, por lo que se recomienda accesar esta opción utilizando la combinación de teclas "Control"-"m".

8.3.8.4 Listado

No Actualizar. Aunque la actualización del diagrama de flujo es bastanre rápida con un tiempo de respuesta de 1 seg aproximadamente, la

actualización del listado es mucho más lenta, 3 seg más o menos. La opción de No Actualizar, inhibe el listado de tal forma que éste ya no reflejará los cambios realizados en el diagrama de flujo. Se recomienda activar esta opción mientra se estén editando los argumentos de las instrucciones dentro del diagrama de flujo. No Actuliazar es sustituída por Actualizar que activa la actualización del listado.

PicoC, PicoPascal, PicoLEPP. Estas opciones indican el lenguaje en el que se desea contruir el listado, correspondiendo a C, Pascal y el nativo de Newt que es LEPP descrito en la sección 8.2.1. El lenguaje por omisión es Pascal por ser el más utilizado como lenguaje introductorio en las universidades. El lenguaje que se esté utilizando es marcado por un asterisco.

8.3.8.5 Options

Este menú sólo contine opciones válidas cuando se está editando el argumento de alguna instrucción. No aparece en la Barra de Menús en ningún otro momento. Sus etiquetas están en inglés porque es un menú que provee el sistema con el que se desarrolló Newt. Para utilizarlas debe estar activado el cursor de texto al editar el argumento de alguna instrucción seleccionada en la Ventana de Diagrama. Selecciónese con el ratón, oprimiendo el botón izquierdo y arrastrando el cursor del ratón sin soltar el botón, el texto que se desea utilizar con éstas instrucciones, el texto se oscurece. Las opciones posibles son:

Copy. Copia el texto seleccionado a una sección de memoria llamada "buffer" donde es guardado, el texto oscurecido no es borrado.

Paste. Copia el contenido del buffer a partir de donde se encuentra el cursor de texto. Si se encuentra algún texto seleccionado al escojer Paste, este texto es sustituído por el contenido en el "buffer".

Cut. Se borra el texto seleccionado y se guarda en el "buffer".

Substitute. Sustituye el texto seleccionado por el contenido en el "buffer".

CAPITULO IX

DESCRIPCION TECNICA DE NEWT

9.1 Introducción

Aunque es cierto que la veracidad de los conceptos que se han discutido en esta tésis aún está a prueba, no se puede dudar que representan un gran avance teórico a lo que se tenía hace unos cuantos años. A través de todo este tiempo, se ha podido acumular y estructurar un gran caudal de conocimientos y experiencias que todavía tienen que fundamentarse para poder seguir adelante en la investigación. Todo este material puede ser muy interesante e incluso impresionante, pero ¿Funciona?.

Para contestar este justo cuestionamiento, es necesario desarrollar extensivamente sistemas que signifiquen una oportunidad para experimentar con la tería con que se cuenta. Para que un programa educativo pueda servir como base para una comprobación científica de todas o algunas de las premisas que ya se han expuesto, debe ser un sistema útil, es decir, que resuelva alguna necesidad.

Newt es pues, un intento de resolver un problema real, y al mismo tiempo un experimento donde se prueba la validadez de muchos conceptos y teorías. Difícilmente un programa puede ser diseñado siguiendo al pie de la letra los procedimientos e incluyendo todos los elementos que que aquí se han expuesto. No obstante, en el diseño de Newt, se ha intentado mantener un apego a la teoría con la mayor cantidad de elementos de interfase e instrucción posibles, siempre y cuando sean útiles. En este capítulo, se explica la utilización de la teoría expuesta en capítulos anteriores aplicada al diseño de Newt.

9.2 El Lenguaje cT

El lenguaje cT fué desarrollado en el Laboratorio de Investigación de Educación por Computadora (CERL; Computer Education Research Laboratory) por un grupo de investigadores encabezados por Judith y Bruce Sherwood. Los trabajos se iniciaron en 1985 instalándose el primer ambiente cT en las estaciones de trabajo Andrew del Centro de Tecnología Informativa (ITC). Para 1988, ya existían versiones completas y estables en máquinas compatibles con IBM PC y Macintosh.

El lenguaje cT y su ambiente de trabajo son descendientes directos de los lenguajes TUTOR y MicroTutor. TUTOR es un lenguaje desarrollado como parte integral del proyecto PLATO (ver sección 4.2). Por motivos de eficiencia, gran cantidad de aspectos de este lenguaje están estrechamente ligados a los componentes físicos de PLATO. De

aquí nació la idea de MicroTutor, a su vez derivado de TUTOR como un lenguaje que puede ser transportado entre varias computadoras con diferentes procesadores y tipos de pantallas. cT es muy similar en su estructura básica a MicroTutor, sin embargo cT fué desarrollado con la meta de explotar ambientes computacionales modernos con énfasis en ventanas y ratón. Además, tiene mayor portabilidad que MicroTutor debido al reescalamiento automático de los desplegados de pantalla.

Las principales características del lenguaje cT son las siguientes:

1. Utilización de la pantalla gráfica
2. Diferentes tipos de letra y estilos de texto
3. Interacción a trevés de ratón y menús
4. Análisis sofisticado de entrada de datos
5. Secuenciamiento complejo del flujo del programa
6. Ayuda y ejemplos en línea
7. Interacción entre edición de código fuente, pantalla de ejecución y ayuda.

cT es un lenguaje especialmente diseñado para poder ser usado por programadores principiantes al mismo tiempo que ofrece una gran cantidad de funciones avanzados que pueden ser explotadas por programadores expertos también.

Adicionalmente, cT ofrece una compatibilidad total entre una gran cantidad de computadoras. Un programa escrito en IBM puede ser compilado y ejecutado sin cambios en una Macintosh. Esto es algo muy difícil de lograr en lenguajes que manejan gráficas debido a que las instrucciones para dispositivos gráficos dependen en gran medida del hardware. Además, los programas realizados en cT mantienen compatibilidad entre diferentes versiones del lenguaje. Es decir que un programa viejo puede ser ejecutado por ambientes cT nuevos, incluso si se han realizado cambios mayores al lenguaje. Esto se logra por medio de un cargador de ejecutable que toma al archivo binario generado por el ambiente cT y checa el nivel de sintaxis que ofrece. Si el nivel de sintaxis es viejo, el ejecutor hace una conversión al nivel más reciente.

9.3 Diseño de la Instrucción en Newt

Para el diseño de Newt se utilizó una parte significativa de la teoría que se presentó en los capítulos anteriores. Se siguieron rigurosamente las fases de análisis, y desarrollo tanto en el diseño de la instrucción como en el diseño de la interfase. Al momento de escribir esta tesis, la fase de evaluación está todavía en proceso debido a que es la que toma mayor tiempo.

9.3.1 Análisis

De acuerdo a la metodología discutida en el capítulo IV, se realizaron tres diferentes análisis: análisis de necesidades, análisis de metas y análisis de tareas.

9.3.1.1 Análisis de Necesidades

Para realizar este análisis es necesario determinar la cantidad de conocimientos que deben tener los alumnos para iniciar la instrucción, es decir el estado actual, y la cantidad de conocimientos que deben tener cuando terminen la fase de la instrucción de la que Newt forma parte; el estado final.

La primera parte es fácil, dado que desde un principio se afirmó que se intenta ayudar a los alumnos que nunca han sido expuestos a la tecnología computacional. Mientras que se caracterizó el estado final de la siguiente forma:

1. Familiarización con los componentes físicos de una computadora, como monitor, teclado, ratón y disco duro principalmente.
2. Familiarización con los conceptos no físicos de la computadora como, programa, sistema operativo y archivos.
3. Familiarización con los conceptos primordiales de los lenguajes computacionales como instrucciones de entrada y salida e instrucciónes de modificación de flujo, evaluación de expresiones, ciclos y variables.
4. Dominio sobre el concepto de pensamiento algorítmico y estructurado.

Dado que se parte de cero en el nivel de conocimiento semántico, son estos mismos puntos que representan el estado deseado el conjunto de necesidades que se desean cubrir con la utilización de Newt. Cabe notar, que el cuarto punto es el que se considera de mayor prioridad, el tercero como necesario y el primero y segundo como muy deseables.

9.3.1.2 Análisis de Metas

Para que pueda considerarse que un alumno ha alcanzado el estado deseado debe cumplir con las siguientes metas planteadas en forma de objetivos:

1. El alumno podrá seguir por sí mismo todos los pasos necesarios para iniciar el sistema newt, crear un programa con él y guardarlo para su posterior utilización.
2. El alumno sabrá como accesar cualquier programa elaborado a través del sistema Newt que esté almacenado en disco dentro del sistema newt y ejecutarlo.

3. El alumno entenderá y sabrá manipular los archivos generados por newt que contengan programas de tal forma que podrá copiar, renombrar y borrar tales archivos a través del sistema operativo MS-DOS.
4. El alumno será capaz de resolver cualquier problema de programación de los que se presentan en los cursos normales de computación tomando en cuenta las limitaciones del sistema Newt en cuanto al uso de variables.

9.3.1.3 Análisis de Tareas

El análisis de tareas consistió en los siguientes pasos:

A. Organización jerárquica de objetivos por prioridad:

1. Resolver problemas de programación algorítmica.
2. Familiarización con los conceptos de computadora y programa.
3. Dominar la utilización del sistema Newt, iniciar Newt, cargar programa, modificar programa y grabar programa.
4. Dominar la manipulación de archivos con el sistema operativo.

B. Para lograr ésto el alumno debe poseer los siguientes conocimientos ordenados de mayor a menor prioridad:

1. Una gran práctica de solución de problemas que requieren programación.
2. Un dominio sobre la utilización de las distintas instrucciones que conforman un diagrama de flujo y un programa.
3. Una amplia familiarización con las instrucciones del sistema Newt que ayudan a modificar un programa.
4. Una amplia familiarización con las instrucciones del sistema Newt que almacenan la información para poder utilizarla después.
5. Una amplia familiarización con los comandos de manejo de archivos del sistema operativo.

C. Por el momento, Newt no ofrece un tutorial que muestre el significado de cada instrucción y su utilización. Debido a esto, la instrucción deberá ser complementada por una breve explicación previa por parte del maestro sobre la composición de una computadora y la utilización de los diagramas de flujo. Tomando en cuenta esto, la secuencia de instrucción que se sugiere es la siguiente:

1. Explicación en clase de los componentes de una computadora, de ser posible con una muestra física en clase.
2. Explicación sobre instrucciones de diagrama de flujo y su utilización.

3. Realización de ejemplos y ejercicios en el sistema Newt.
4. Explicación sobre archivos y sistema operativo.
5. Realización de ejercicios complejos y tareas en Newt, poniendo énfasis en la entrega de los resultados en discos.

9.3.2 Desarrollo

Dentro del modelo de Tennyson de Desarrollo Integral de la Instrucción se puede identificar que Newt salta los primeras fases de la adquisición de conocimiento directamente a la fase de complejidad cognocitiva. Esto se puede lograr debido a que lo que el alumno debe aprender para lograr el pensamiento algorítmico no es demasiado, ocupando generalmente unas cuantas páginas de apuntes y a lo más dos horas de exposición en clase. Sin embargo, el esfuerzo mental que se necesita para poner a funcionar tales conocimientos sí es considerable. Newt entonces, enseña haciendo que el alumno practique.

Dentro de los objetivos del aprendizaje, Newt ataca directamente el desarrollo de estrategias cognocitivas para la resolución de problemas y la utilización de todo el mecanismo cognocitivo para desarrollar la creatividad en el alumno.

Para poder lograr la complejidad dinámica, Newt necesita de la ayuda del maestro. Dado que Newt por sí sólo no presenta diversos problemas al estudiante, debe ser el maestro el provea de condiciones variantes en los problemas.

No obstante, debido a que newt puede ejecutar los diagramas de flujo que se desarrollen en él, puede mostrar al alumno si su solución es correcta, propiciando la exploración y experimentación con las instrucciones, y proporcionando un método ideal para desarrollar experiencias autodirigidas directamente relacionadas con el proceso creativo.

9.3.3 Evaluación

En un futuro, cuando se haya completado el tutorial en línea, Newt podrá presentar problemas al estudiante y verificar la solución aunque sea de forma parcial. Es decir, Newt podrá plantear un problema concreto, y cuando el alumno indique que su respuesta está completa, Newt verificará los elementos de la solución y evaluará el programa. Si el programa no cumple con una serie de condiciones como la ausencia de cierta instrucción o que el resultado esté equivocado, Newt podrá hacer sugerencias para mejorar el programa. Ejemplo:

"Lee dos valores, x,y, en la primera línea de tu programa. Diseña el programa de tal forma que eleve x a la y potencia. Deposita el valor que resulta de elevar x a la y en la variable z."

Newt podría hacer sugerencias como:

"Es conveniente que el programa incluya una instrucción FOR"

o simplemente indicar alguna falla:

"La variable z nunca es utilizada en el programa. No sé en cuál variable está el resultado"

"La variable z no contiene el resultado correcto. Tal vez tienes mal un contador, asignaste el valor final a otra variable, o el ciclo tiene un límite incorrecto"

9.4 Diseño de la Interfase de Newt

9.4.1 Análisis: Caracterización de Usuarios y de Tareas

No es necesario hacer un análisis exhaustivo para descubrir que los usuarios de Newt son personas con muy poco o ningún conocimiento semántico de las computadoras y de los diagramas de flujo. Por esto, la cantidad total de tareas que puede llevar a cabo Newt es muy limitado. Típicamente una sesión de Newt consistiría de las siguientes tareas:

1. Creación del esqueleto de un diagrama de flujo por medio de las instrucciones del menú de instrucciones.
2. Edición de los argumentos señalándolos en la ventana de diagrama en modo editar y apendizar.
3. Modificación del diagrama de flujo cambiando el modo de edición en el menú de edición y manipulando las instrucciones de la ventana de diagrama.
4. Ejecución del diagrama de flujo
5. Si el programa no funciona ir a 1.
6. Salvar el programa en archivo.

Cada una de estas tareas es atómica, es decir que se lleva a cabo con un solo comando.

9.4.2 Desarrollo

9.4.2.1 Especificación Funcional

El número de funciones de que lleva a cabo Newt es muy reducido. Se clasifican en cuatro categorías:

1. Funciones de creación de diagrama de flujo (agregar instrucciones)
2. Funciones de edición de diagrama de flujo (cambio de modo de edición)
3. Funciones para guardar la sesión de trabajo
4. Funciones de modificación de la forma de visualización del diagrama y listado

Cada función se lleva a cabo por medio de una tarea atómica.

Las tareas de tipo 1 y 2 se llevan a cabo oprimiendo lod botones del menú de instrucciones y del menú de edición. Para editar una instrucción, sólo se necesita editar la instrucción en la ventana de diagrama. La operación de edición depende del botón de edición que esté oprimido.

Las tareas del tipo 3 y 4 se llevan a cabo a través de una sola selección de menú.

9.4.2.2 Prototipos y Pruebas Piloto

Desde un principio se desarrollaron prototipos de Newt con funciones limitadas en los que se iba probando la validez del análisis de tareas, la efectividad del diseño de pantalla y el impacto de los mensajes del sistema, progresivamente.

Además desde las fases tempranas se hicieron varias pruebas piloto, las cuales representan los únicos resultados sobre la efectividad de Newt en el aprendizaje. Cabe señalar que estas pruebas piloto resultaron muy alentadora.

9.4.3 Factores Humanos

En todo memento se tomó en cuenta la sensibilidad del estudiante. Se intentó no utilizar comandos y etiquetas con nombres verbalmente violentos o distrayentes.

Además, la interfase se diseñó de forma lo suficientemente intuitiva como para que su utilización hiciera sentirse agusto a personas con diferentes personalidades. Por ejemplo una pesona sensitiva encontrará que la selección de tareas es lo suficientemente reducida con pocos opciones de tal forma que conforme vaya avanzando en el manejo de Newt no tendrá que aprender nuevos comandos sino simplemente como utilizarlos mejor. Una persona intuitiva encontrará que el sistema lo estimula a explorar y encontrar nuevas formas de utilizar las instrucciones. Una persona extrovertida encontrará la utilización del color es muy estimulante, mientras que una persona introvertida se sentirá a gusto con la sobriedad de los tonos grises de la pantalla primaria. Una persona ordenada podrá

hacer primero un bosquejo global del diagrama de flujo, alimentando todas las instrucciones y luego los argumentos, mientras que una persona desordenadad podrá alimentar las instrucciones y sus argumentos una a una, probándolas conforme las vaya incluyendo en su programa.

9.4.4 Bloqueos Psicológicos

Debido a que los usuarios novatos son especialmente propicios a caer en bloqueos emocionales, se tomó mucho cuidado para evitarlos.

9.4.4.1 Aburrimiento. Para evitar el aburrimiento, se mantuvieron tiempos de respuesta menor 2 segundos en el dibujado del diagrama de flujo. Debido a que la actualización del listado podría tardar hasta 4 segundos, se incluyó una opción para inhibir la actualización del listado de tal forma que el usuario no tiene que pasar por tiempos prolongados de espera. Además, Se va indicando las instrucciones conforme se van ejecutando para que el alumno siga con la vista la acción de la ejecución del programa y evitar que se aburra cuando los programas contengan pocas instrucciones de salida.

9.4.4.2. Pánico. Este bloqueo se atacó animando las instrucciones en ejecución para que el alumno no piense que su programa está inhibido y adicionalemente existe una instrucción para detener la ejecución de un programa en caso de que el alumno piense que su programa está ciclado.

9.4.4.3. Frustración. En este renglón, primeramente se proveyeron la cantidad adecuada de opciones en los comandos. Por ejemplo, un alumno puede escojer entre insertar antes o después de cualquier instrucción con sólo utilizar el botón derecho o izquierdo del ratón, y lo mismo para mover. Los archivos pueden ser grabados con cualquier nombre pero se prevee de un archivo default. Además, el listado puede aparecer en tres lenguajes diferentes.

En segundo lugar, el sistema no sólo perdona los errores cometidos sino que automáticamente los corrige. El insertar, apendizar, editar, borrar o mover una instrucción de forma incorrecta provoca un mensaje de error inmediato y una corrección, si es posible. Además, se incluyó un comando "Deshacer" por menú de tal forma que el alumno puede arrepentirse de una gran parte de los cambios hechos al programa.

9.4.4.4. Confusión. La confusión se atacó de varias formas:

1. Todos los comandos más comunes están en botones disponibles todo el tiempo con logotipos que tratan de indicar de una forma pictórica qué es lo que hace cada comando.

2. Cada error cometido por el usuario genera un mensaje que indica cuál fué el error cometido y cómo arreglarlo si es que Newt no lo corrige automáticamente.
3. El cursor cambia visiblemente de forma cuando se cambia el modo default, de tal forma que el usuario no borra una instrucción al señalarla cuando lo que realmente quería hacer era editarla.

9.5 Elementos de la Interfase

9.5.1 Dispositivos Físicos

Idealmente un programa debe tratar de mantener compatibilidad entre funciones accesadas por medio de dispositivo de señalamiento como el ratón y el teclado. Debido a las limitaciones del lenguaje cT, esto no fué completamente posible. De tal forma que no es posible operar Newt si no se cuenta con un teclado y un ratón. De estos dos dispositivos, el ratón resulta ser el de mayor utilidad, no obstante, sólamente a través del teclado se pueden alimentar los argumentos de las instrucciones.

9.5.2 Dispositivos Virtuales

Newt hace amplio uso de estos dispositivos, que son los que le permiten mantener un ambiente innovador y moderno.

9.5.2.1 Ventanas. Utiliza ventanas para separa las categorías de información; Ventana de Diagrama, Ventana de Listado, Ventana de Ejecución. Además, los argumentos de las instrucciones se leen en ventanas y los mensajes de error y de sistema aparecen en sus propias ventanas estándard.

Las ventanas de diagrama y listado son hacia pantallas virtuales. Aunque no pueden ni cambiar su tamaño, ni su posición y ni superimponerse, tienen controles de "scroll" para cambiar la vista de la pantalla virtual.

9.5.2.2 Menús. Existen menús desenrollables desde la barra de menús y menús permanentes.

9.5.2.3 Botones y logotipos. La acciones más utilizadas están en forma de botones con logotipos. Los botones dan al estudiante la sensación de que él está iniciando todas las acciones y que él lleva el control sobre el sistema. Los logotipos permiten que se identifique rápidamente la función de cada botón.

9.5.2.4 Ventanas de Diálogo. En ciertas ocaciones, Newt utiliza las ventanas de diálogo, en las que dá y pide información. Por ejemplo cuando se va a borrar un diagrama o se pide abrir uno nuevo, Newt dialoga con el usuario indicando que se va a borrar el diagrama que está en pantalla y pregunta si se desea guardarlo. Si se le pide que grabe un archivo con el nombre de un archivo que ya existe, en una ventana, Newt indica que el archivo ya existe y pregunta se desea borrar el más viejo. A su vez, da retroalimentación en ventanas sobre acciones, por ejemplo cuando se borró un diagrama y no se guardó en archivo.

9.5.3 Visualización

De las técnicas de visualización, se utilizaron las siguientes:

9.5.3.1 Abstracción. El diagrama de flujo es la mejor forma de abstraer. El diagrma de flujo representa de una forma las acciones que se van a realizar en un programa, los argumentos sobre los cuales se llevan a cabo esas acciones de otra forma y las posibilidades y flujo de las acciones en una tercera forma.

9.5.3.2 Discretización. Un programa es un proceso discreto por naturaleza. Para enfatizar ésto, cada una de las instrucciones que se van ejecutando, es decir, en cada paso de la ejecución, se va resaltando. Esto da una perfecta idea de las acciones que se llevan a cabo al ejecutar un programa. En el futuro es posible incluir una lista de depuración en la que el alumno podrá incluir las variables cuyos valores desea estar constantemente observando.

9.5.3.3 Color. El color es utilizado para mostrar el modo en el que está el programa, como editar, insertar, borrar o mover, coloreando el botón que representa ese modo. También se utiliza color para mostrar la instrucción sobre la cual se va a realizar la próxima inserción. Un cambio de tono va mostrando las instrucciones que se van ejecutando. Finalmente, el color se utiliza para proveer a Newt de una atmósfera sobria y visualmente relajante.

9.5.3.4 Geometría. Cada instrucción tiene una diferente forma geométrica ayudando su rápida identificación dentro del diagrama de flujo y del menú de instrucciones. Se utiliza también la geometría para indicar cuál es la instrucción que se está moviendo. Por último, el cursor cambia su forma dependiendo del modo de edición; un flecha para editar, un cruz para mover, una tijeritas para cortar y una gran "I" para insertar.

9.5.3.5 Movimiento. El movimiento se utiliza para animar las acciones de movimiento de una instrucción de un lugar en el diagrama a otro. También su utiliza al ir cambiando de color las instrucciones que se van ejecutando para dar una sensación de avance de la ejecución sobre el diagrama de flujo y de localización.

9.5.3.6 Presentación Progresiva. Para cumplir con este requisito, se incluyó una opción en la que el usuario puede disminuir de tamaño el diagrama de flujo, poniendo dentro de la ventana casi la totalidad del diagrama de flujo, pero haciéndo más difícil distinguir entre diferentes instrucciones y el contenido de los argumentos.

9.5.4 Manipulación Directa

La forma que se lleva el diálogo entre el estudiante y Newt es a través de la manipulación directa. La necesidad de utilizar comandos con sintaxis compleja es totálmente eliminada al sustituir los comandos por acciones directas sobre objetos. Todos los objetos de interés están constantemente representados en pantalla; botones, instrucciones y menús.

Para utilizar Newt, el usuario lleva el control del diálogo siendo el iniciador de todos los eventos. Ninguna función se lleva a cabo si el usuario no realiza alguna manipulación sobre algún objeto. Además de proporcionarle el control, la manipulación directa hace que el sistema en Newt desaparezca como intermediario y que el usuario se concentre en el significado semántico de sus acciones, en este caso, la creación de un programa que resuelve un problema.

9.5.5 Diseño de la Pantalla

El diseño de pantalla fué uno de los factores en que más cuidado se puso al diseñar Newt.

1. La terminología es consistente con el lenguaje de las computadoras, sin embargo se cuidó de la correcta utilización del castellano.
2. Los títulos de los menús y los artículos que contienen son breves pero perfectamente claros sobre la acción que generan.
3. La pantalla minimiza la carga en la memoria del usuario. La información que se presenta en pantalla es la mínima necesaria para que el usuario lleve a cabo su tarea. La cantidad de pasos que se tienen que llevar a cabo para completar una tarea es mínima; la mayoría de las tareas son atómicas.
4. Se logró bastante flexibilidad en la forma de desplegar la información en pantalla. El usuario puede escoger entre ver el diagrama aumentado o reducido, ver el listado actualizado o nunca actualizarlo y puede mover los elevadores de las ventanas para lograr vistas diferentes de las pantallas virtuales.

5. La utilización del color es discreta. Sólo se utilizan tonos de gris en la pantalla principal, amarillo para resaltar botones oprimidos, naranja para instrucciones seleccionadas para inserción en el diagrama de flujo, y violeta para los mensajes del sistema.
6. Los elevadores de ventana proveen una forma de orientación sobre qué parte de la pantalla virtual es observada.
7. Para llamar la atención del usuario se utiliza sólamente el cambio de color y la utilización de ventanas de mensaje.

9.5.6 Manejo de Errores

En todo momento se lleva un monitoreo de la posibilidad de errores. Existen tres tipos de errores:

1. Errores de Secuenciación de Instrucciones. Estos errores se presentan cuando se introduce una instrucción en un logar inapropiado. Por ejemplo un ENDFOR antes de un FOR, o el END antes de cerrar todos los ciclos.

2. Errores de Sintaxis. Estos se presentan cuando se comente un error al alimentar el argumento de alguna instrucción.

3. Errores de Archivo. Estos errores ocurren cuando se intenta abrir un archivo que no existe o se intenta grabar en disco ilegalmente, tal vez cuando el disco está lleno o no se tiene permiso del sistema.

Todos estos errores son monitoreados por Newt. Cuando Newt encuentra un error, intenta corregirlo; por ejemplo un error del primer tipo. Si Newt no puede corregirlo, indicará al usuario cómo corregir el error. Los errores tipo tres no se pueden corregir, entonces Newt sólo indicará el error. No importando el tipo de error, Newt indicará exactamente en qué consiste, porqué ocurrió y si de ser posible o necesario cómo corregirlo.

9.6 Conclusión

El diseño de programas educativos con interfases modernas e intuitivas, y una instrucción adecuada a su utilización en la computadora no sólo es posible sino necesario. El sólo hecho de desarrollar este tipo de aplicaciones de una forma estructurada, aunque se sigan los lineamientos aquí expuestos en forma mínima, significa un avance real para la educación como un todo. La idea de convertir el aprendizaje en un proceso no solo productivo sino agradable puede lograrse si se utilizan las grandes ventajas que ofrecen las computadoras. Y ahora, más que nunca, se tiene la tecnología y el conocimiento para hacer de esta idea una realidad.

CAPITULO X

CONCLUSIONES

Después de haber hecho una profunda investigación sobre el desarrollo de Instrucción Ayudada por Computadora e Interfases Gráficas se puede decir que se tiene suficiente teoría para desarrollar Programas Educativos con la suficiente capacidad para tomar un papel significativo en la educación. Es posible desarrollar, siguiendo una metodología adecuada, sistemas que pueden funcionar como cursos bases de la instrucción o como auxiliares en ella. Los Programas Educativos diseñados como curso base tendrán un papel proponderante en la instrucción sustituyendo totalmente o casi totalmente al maestro. Los Programas Educativos que sean diseñados como auxiliares, pueden tener un papel muy importante al demostrar conceptos que son difíciles de entender en el pizarrón utilizando las capacidades de simulación, graficación y animación de las computadoras modernas. También pueden fortalecer secciones de la instrucción que no pueden ser cubiertas extensivamente en el salón de clase por falta de tiempo o por fallas u omisiones en los conocimientos que se asume el alumno debe tener al iniciar el curso.

Dentro de un Programa Educativo existen dos elementos diferentes pero relacionados que merecen un proceso de diseño paralelo, siempre tomando en cuenta el uno con el otro: La Interfase y La Instrucción. La Instrucción es la que dará al Programa Educativo su importancia y papel dentro del curso, y contiene los conocimientos que se desea transferir al estudiante. La Interfase es la que hará que el programa sea una herramienta efectiva, logrando una motivación en el estudiante y una satisfacción subjetiva.

La Instrucción lleva al alumno a través de una serie de niveles mentales o semánticos, que van desde una simple exposición de conceptos, hasta el desarrollo de procesos creativos de orden superior. La Interfase debe llevar correctamente el diálogo entre el estudiante y el sistema, evitando los errores fatales y dando constántemente retroalimentación sobre el estado del sistema, las posibles acciones a llevar a cabo y el resultado de estas acciones.

Para que un Programa Educativo pueda tener un desempeño satisfactorio, debe pasar por varias fases de diseño: Análisis, Desarrollo y Evaluación. Sin embargo, el diseño de un programa educativo es un proceso creativo en el que las tres fases interactúan constantemente debido a que la evaluación no es algo que simplemente se deje al final de la implementación, sino que se realiza en cada paso de la creación del sistema.

Esta evaluación, lleva al descubrimiento de errores y nuevas metas, que a su vez conducen a nuevos análisis y desarrollos.

Finalmente se puede preveer que el constante mejoramiento de herramientas para el desarrollo de Programas Educativos lograrán que el desarrollo de estos programas sea cada vez más una tarea exclusiva de las personas expertas en las áreas académicas, prescindiendo de la ayuda de programadores profesionales o diseñadores gráficos. Además, tiempos y costos de desarrollo se disminuirán substancialmente debido a la utilización de herramientas como HyperCard, Redes Computacionales, Multimedios y Sistemas Basados en Conocimientos.

Sin duda, con la reducción de los costos del equipo computacional, la disponibilidad de microcomputadoras con gran capacidad de procesamiento y gráficas, la existencia de herramientas de desarrollo orientadas a personas con escasos conocimientos de programación, y con 30 años de experiencia y teoría sobre desarrollo de Programas Educativos, ha llegado el momento de afrontar el reto de la tecnología en la educación, y vencerlo.

REFERENCIAS BIBLIOGRAFICAS

[ALES89] Alessi S. M. "Computer Based Instruction". 1989.

[BLAS90] Blaschke C. L. "Integrated Learning Systems/Instructional Networks: Current Uses and Trends". En Educational Technology. Noviembre 1990.

[BOWE90] Dennis B. y Tsai C. "HyperCard in Educational Research: And Introduction and Case Study". En Educational Research. Febrero 1990.

[BROW90] Brown J. R. y Cunningham S. "Visualization in Higher Education". En Academic Computing. Marzo 1990.

[CHAB89] Chabay R. W. y Sherwood B. A. "Computer Assisted Instruction and Intelligent Tutoring Systems". Hillsdale, NJ: Earlbaum. 1989.

[CUNN90] Cunningham S. y Zimmerman W. "Visualization in Teaching and Learning Mathematics". Sufragado por Mathematical Association of America. 1990.

[ELLS90] Ellson R. "Visualization at Work". En Academic Computing. Marzo 1990.

[FOLE74] Foley J. D. y Wallace V. L. "The Art of Natural Graphic Man-Machine Conversation". En Proceeding of the IEEE. Abril 1974.

[FORS74] Forsythe, A. I. y Keenan, T. A. "Lenguajes de Diagramas de Flujo". Editorial Limusa, México. 1974

[HANS90] Hansen E. "The Role Of Interactive Video Technology in Higher Education: Case Study and Proposed Framework". En Educational Technology. Septiembre 1990.

[HUNK89] Hunka S. "Designing Guidelines for CAI Authoring Systems". En Educational Technology. Noviembre 1989.

[IBM90] "A Right to Die? The Case of Dax Cowart". IBM Applications Brief. 1990.

[KEAR85] Kearsley G. y Halley R. "Designing Interactive Software". Park Row Press. La Jolla, California. 1985.

[MITZ79] Mitzel H. E. "Computer Based Education". En Encyclopedia of Educational Research. Volumen 1. The Free Press, N. Y. 1979.

[OKEY90] Okey J. R. "Tools of Analisis in Instructional Development". En Educational Technology. Junio 1990.

[PERE90] Perez R. S. y Seidel R. J. "Using Artificial Intelligence in Education: Computer-Based Tools for Instructional Development". En Educational Technology. Marzo 1990.

[RUBI88] Rubin T. "User Interfase Design for Computer Systems". Halsted Press. N.Y. 1988.

[SCHA90] Schaefermeyer S. "Standards for Instructional Computing Software Design and Development". En Educational Technology. Junio 1990.

[SHNE87] Shneiderman B. "Designing the User Interfase". Addison-Wesley Company, Publishing Inc. 1987.

[SMIT89] Smith P. E. "Some Learning and Instructional Theory Considerations for the Development of Computer Related Instructional Materials". En Educational Technology. Noviembre 1989.

[TENN90a] Tennyson R. D. "A Proposed Cognitive Paradigm of Learning for Educational Technology. En Educational Technology. Junio 1990.

[TENN90b] Tennyson R. D. "Integrated Instructionnal Design Theory: Advancements from Cognitive Science and Instructional Technology". En Educational Technology. Julio 1990.

[THOR90] Thornburg M. S. "Knowledge-Based Tutoring of Special Education Classification Concepts". En Educational Technology. Marzo 1990.

[WHIT90] Whitney R. E. y Urquhart N. S. "Microcomputers in the Mathematical Science: Effects on Courses, Students, and Instructors". En Academic Computing. Marzo 1990.