UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES
ARAGÓN

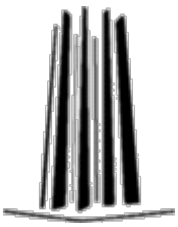
"ENDER UNA HERRAMIENTA DE CÓMPUTO COMO APOYO EN LA DOCUMENTACIÓN DE PROGRAMAS"

T R A B A J O E S C R I T O

EN LA MODALIDAD DE DESARROLLO
DE UN CASO PRÁCTICO
QUE PARA OBTENER EL TÍTULO DE:
INGENIERA EN COMPUTACIÓN
P R E S E N T A:

D E L I A G A L L A R D O L U C K I E

ASESOR: ING. JOSÉ GONZÁLEZ BEDOLLA



MÉXICO, 2011

A mis **padres**, por ser un ejemplo digno a seguir.

A mis **hermanos** por la unidad que nos caracteriza.

A **Edgar** por sus sabios consejos y enorme apoyo.

A mis **sobrinos** y sus **madres** por la alegría.

A mis **amigos** por escucharme y siempre estar ahí.

A los **profesores** por el aprendizaje.

A la **vida** que me ha enseñado el **Ciclo sin fin** …

Desde el día que al mundo llegamos
Y nos ciega el brillo del sol
Hay mucho más para ver de lo que se puede ver
Más para hacer de lo que da el vigor
Son muchos más los tesoros
De los que se podrán descubrir
Mas bajo la luz del sol jamás habrá distinción
Grandes y chicos han de convivir
En el ciclo sin fin que nos mueve a todos
Y aunque estemos solos, debemos buscar
Hasta encontrar nuestro gran legado
En el ciclo, el ciclo sin fin

A **todos** mil gracias, les quiere y admira

DeL

I N T R O D U C C I Ó N

El presente trabajo pretende crear en el desarrollador una cultura informática, que aplique a la documentación del código fuente; para que en un futuro inmediato los desarrollos, programas, aplicaciones y sistemas sean entendibles, tanto para el desarrollador que lo elaboró, como algún otro que requiera darle mantenimiento.

En el primer capítulo se mencionan algunos conceptos de carácter importante para este caso práctico, como es la documentación y los diferentes significados que se le dan a ésta palabra. Crisis de software: la importancia que ha tenido durante más de dos décadas y el gran auge que en la actualidad ha revolucionado junto con el Hardware. UML como lenguaje para diseño de sistemas: dicho lenguaje que ha tenido mayor éxito, sobretodo la teoría que le apoya, que cualquier persona con nivel de estudios profesionales independientemente del área en el que se desarrolle, puede entender el funcionamiento del sistema diseñado por dicho lenguaje. Tipos de programación: es importante señalar cuales son los tipos de programación para saber el estado del arte que existe en la actualidad. Lenguajes de programación; este tema menciona específicamente la importancia de Visual Basic como lenguaje de programación de última generación, debido a que la herramienta del presente trabajo fue desarrollada en este lenguaje.

Es importante señalar que en el último tema del capítulo uno, encontraremos todo el análisis que se ha derivado del caso práctico de “ENDER una herramienta de cómputo como apoyo en la documentación de programas”.

Al llegar al segundo capítulo se crea todo el ambiente del diseño del sistema a construir, para poder llevarlo a cabo se toman en cuenta las siguientes fases: Modelo conceptual: en este tema notaremos el flujo que lleva el diseño, mismo que nos ayuda a comprender mejor las fases en las que se divide y la

interacción del programador (usuario). Diagrama de casos de uso: encontraremos los diferentes casos que se presentan en la programación del presente caso práctico, con esto encontraremos mejor las fases en las que se divide nuestro sistema. Diagrama de secuencias: este tipo de diagrama se presenta al final, para mostrarle al programador (usuario) el flujo que debe seguir durante las pruebas del sistema y de esta forma pueda conocer el funcionamiento del mismo.

Encontraremos en el tercer capítulo el desarrollo de las etapas de la programación del diseño mencionado en el capítulo dos. Se comienza por el diseño de las pantallas según lo propuesto en el diseño del sistema y se procede a la codificación como es leer del Sistema Operativo la fecha y la hora, detectar las diferentes unidades de almacenamiento y unidades extraíbles, lectura de un proyecto de Visual Basic 5.0 o 6.0, crear un árbol de nodos que contenga las formas, módulos y clases que se encuentren en el proyecto leído, identificar variables y constantes no declaradas o no utilizadas por el sistema, agregar comentarios a los procedimientos, funciones o métodos, guardar el proyecto con los ajustes realizados al proyecto como tal. Después, la instalación de ENDER para documentar código fuente: con este tema lo que se pretende es dar una breve explicación de seguimiento de la instalación del sistema construido. La Fase de pruebas: contiene una lista de los errores que se encontraron durante esta faceta; como gran apoyo se pudieron conseguir algunos proyectos realizados en el centro de cómputo de la FES Aragón, éstos están hechos en Visual Basic versión 6.0. Al final encontramos las Perspectivas: éstas nos sirven como el cierre de la documentación del código fuente para el lenguaje de programación Visual Basic versión 5.0 y 6.0, pero la idea fundamental es que se continúe con éste proyecto en otros lenguajes de programación como, C, Java, PHP, .NET, etc.

CAPÍTULO 1

ANÁLISIS DE LA PROBLEMÁTICA Y FUNDAMENTOS TEÓRICOS

1.1. Antecedentes

En la actualidad empresas e instituciones cuentan con diversos proyectos de desarrollo de software, cuyo destinatario final requiere de documentación de código fuente para algún proyecto, mismo que en muchas ocasiones no es documentado con anterioridad, y no resulta tan sencillo entender para los desarrolladores, ya que se torna en hacer una recapitulación de cada uno de los procedimientos, que requiere tiempo específico para la construcción del proyecto funcional. Trayendo como consecuencia la necesidad de contar con mejores herramientas de desarrollo y documentación y con la firme intensión en todo momento de cumplir con los tiempos y tener una calidad integral mejorada.

Para verificar la existencia de herramientas de desarrollo y documentación del presente proyecto, se realizó una inspección en el centro de cómputo de la FES Aragón, donde se realizan desarrollos de Sistemas y Software orientados a la reutilización del código fuente. Mismos que a su vez, no contienen la documentación adecuada, y como no se tiene idea de qué tratan, se vuelven obsoletos y difíciles de interpretar.

Esto no se ha convertido en un problema crítico hasta el momento, ya que en gran parte se debe a que en el centro de cómputo lo que le interesa es la funcionalidad del proyecto. Por lo que, una vez que el tiempo de desarrollo ha terminado, el programador entrega el proyecto sin hacer una planeación a largo plazo del problema al que se enfrentaran los desarrolladores futuros que tengan que realizar un mantenimiento, retomarlo o para adecuarlo a nuevas plataformas.

El presente trabajo pretende contribuir a resolver parte de este problema desarrollando una herramienta de cómputo, que se encargue de documentar el código fuente de los proyectos desarrollados en Visual Basic 5 o 6, aplicando así

una nueva forma de trabajo, donde el programador no tendrá problema en documentar su código cuando lo requiera.

A continuación se describe aquellos conceptos que se utilizan de manera frecuente en el desarrollo del presente documento. Por lo tanto su definición pretende dejar en claro la forma en que se entiende y se maneje dicho concepto.

1.2. Utilización del concepto “documentación”

Antes de involucrarse en la documentación hay que considerar una parte muy importante que no se debe olvidar: “El ciclo de vida de un proyecto”, este es el que lleva a los programadores a darle un correcto seguimiento a cada proyecto, si este no es bien realizado, entonces se tendrá un proyecto que a futuro va a ser difícil de comprender y sobre todo de mantenimiento complicado. (Ver APÉNDICE A)

Si se considerará en todos y cada uno de los proyectos informáticos la importancia del ciclo de vida, entonces no se tendrían tantos problemas en retomar algún proyecto o simplemente darle seguimiento a uno ya comenzado.

A partir del ciclo de vida se debe considerar la “documentación”, la cual no acostumbramos a tomar en cuenta como se ha mencionado con anterioridad.

Muchas de las palabras que los seres humanos hemos aprendido al paso del tiempo, tienen mucho que ver con la formación y cultura que vamos adquiriendo. Pero las palabras, así como algunos conceptos que asumimos como comprendidos tienen un significado totalmente diferente de lo que se piensa, además del medio ambiente en el que se desarrolla, para la adopción de modismos. Esto también forma parte de la documentación.

En un ambiente informático, se tienen modismos muy parecidos a nuestro lenguaje coloquial, por ejemplo “Documentación” [1] en la Real Academia Española es un: “Documento o conjunto de documentos, preferentemente de carácter oficial, que sirven para la identificación personal o para documentar o acreditar algo”, “Acción y efecto de documentar”. Donde “Documentar[2]” significa: “Probar, justificar la verdad de algo con documentos”. “Instruir o informar a alguien acerca de las noticias y pruebas que atañen a un asunto”. En el área de Informática el concepto de documentación no varía mucho, solo que existen diferentes tipos de documentación, y son los siguientes:

a) Documentación analítica. (Ver APÉNDICE A)
b) Documentación de sistema. (Ver APÉNDICE A)
c) Documentación de programa. (Ver APÉNDICE A)
d) Documentación de operaciones. (Ver APÉNDICE A)
e) Ayuda usuario/ dirección. (Ver APÉNDICE A)

Cada uno de estos tipos de documentación tienen mucho en común, ya que estos mismos pertenecen al ciclo de vida de cualquier proyecto informático. En el caso particular de este proyecto se hablará y se tratará del tema “Documentación de Programa”.

Como se ha manejado hasta ahora, el tema de la documentación que contiene el código de cualquier proyecto informático, no ha sido hasta ahora muy importante que atender, sin embargo, no perdamos de vista el tema “Crisis de Software”, donde se observará como éste problema no ha podido resolverse.

En el caso particular del centro de cómputo de la FES Aragón, para sus proyectos se tiene la documentación de sistema, ésta es una guía para cualquier

[1] www.rae.es
[2] www.rae.es

modificación a los sistemas. A la documentación de código fuente no se le da la mayor importancia, ya que no cae en un rubro comercial y solo se utiliza explotándolo en ese momento. La documentación de manuales para usuario, sí es requerida, este debe de contener la operación de cada uno de los procedimientos de cómo utilizar dicho sistema en particular.

Hablando particularmente de la documentación de código fuente, algunos proveedores de software comercial, en ciertas ocasiones ponen a disposición el código fuente de sus desarrollos, donde se observa la forma que ellos documentan, la cual normalmente es explícita. Esto es específicamente lógico, ya que los desarrolladores de software comercial están siempre pensando hacia futuro para generar nuevas versiones.

La documentación interna es muy importante, principalmente por que es la interfaz con otros desarrolladores de código fuente. Mientras se tengan referencias del desarrollo se tendrá una guía permanente que permitirá realizar un mantenimiento productivo.

Como se ha mencionado anteriormente, en el caso del centro de cómputo, no importa si existe una documentación de código, sin embargo este hecho ha sido el causante de que no se tomen en cuenta los desarrollos ya finalizados, estos podrían ser reutilizados por los programadores y así ahorrar tiempo.

Este trabajo realiza una propuesta de un formato para documentar código fuente, ya que no existe un seguimiento uniforme para la programación y mucho menos para la documentación interna de un proyecto informático.

Una de las características principales de este proyecto es apoyar y/o crear en los desarrolladores una cultura informática, para que en un futuro inmediato les sea más fácil retomar o dar mantenimiento a sus desarrollos.

La propuesta de un formato para documentar código fuente se muestra en el APÉNDICE B.

1.3. Crisis de Software

Durante muchos años se ha caracterizado la evolución de la informática como parte de la vida cotidiana del ser humano. Sin embargo, esta evolución ha llegado a cambiar las perspectivas del hombre, que cada vez exige más y más en el ámbito del desarrollo del software, sin embargo no ha tenido la respuesta adecuada. Por este motivo se ha provocado un fenómeno conocido como "Crisis de Software". (Ver APÉNDICE C)

Antes de comenzar a describir el tema, se definirán los términos "Crisis" y "Software". Crisis[3]: "Momento decisivo y grave de un negocio, la política, etc.". Software[4] (conjunto de tres elementos a conocer): "(1) instrucciones (programas de computadora) que cuando se ejecutan proporcionan la función y el comportamiento deseado, (2) estructuras de datos que facilitan a los programas manipular adecuadamente la información, y (3) documentos que describen la operación y el uso de los programas".

Con la unión de estas dos palabras tenemos que éste es un problema de desarrollo de software. Incluso no solo es de este ámbito, es desde como desarrollar un software, hasta como mantenerlo. En la actualidad al tema "Crisis de Software" se le comienza a dar más importancia, ya que ha perdido la mejora de calidad y productividad en los desarrollos informáticos.

A medida que las computadoras tomaron mayor importancia en la sociedad, el hardware evolucionó de manera tal que ahora los precios son muy bajos, sin

[3] http://www.diccionarios.com
[4] "Ingeniería del software un enfoque práctico" Roger S. Pressman. España 1993.

embargo, para el software los precios siguen subiendo cada vez más. Esto es debido a que la demanda social del software cada día es más grande y compleja y la atención que se le ha puesto es muy baja. (Ver APÉDICE D)

Con lo anterior sólo cabe mencionar que cada día que pasa es importante y que un desarrollador de software tome en cuenta cual es su labor y hasta donde puede llegar, si mejora la calidad de su trabajo actualizándose conforme lo establecido en su perfil. Donde también poco a poco se han ido incrementando las herramientas, para darle a estos la pauta de la creación de software nuevo y versátil, y con ello realizar un mejor desempeño laboral. (Ver APÉNDICE E)

Por estás causas se considera que una de las soluciones se encuentra en la medida que se realizan las funciones para las que fue construido dicho software de modo apropiado, consistente y predecible. Mientras que la calidad del código y las pruebas son parte de esta meta, en su sentido más amplio la corrección incluye la sensibilidad y tolerancia a fallas. Estas cualidades pueden lograrse mas prácticamente si se tiene cuidado en el diseño y la construcción, mas que en las pruebas y reparaciones subsecuentes. Estos son los aspectos más importantes que debemos considerar en la crisis del software.

Con lo que se ha mencionado en este tema es importante mencionar que la documentación de código fuente esta ligada de forma directa, tomando en cuenta sobretodo que la falta de ello es una de las causas por las cuales no se retoman los desarrollos informáticos.

1.4. UML una herramienta de diseño

El UML (Lenguaje Unificado para la construcción de Modelos), es un lenguaje que se encarga de representar físicamente los objetos (llamados módulos) que en un futuro serán programados. En éste se puede especificar visualmente, lo

que el cliente requiere o necesita. Una de las ventajas mayores de este lenguaje, es que todo lo que se represente dentro de los modelos, casos de uso y diagramas, serán fácilmente (teóricamente) entendidos por cualquier persona.

En los principios de la programación, los programadores se encargaban de realizar su tarea “codificar”, y nunca llegaron a tener problemas de desarrollo, ya que cuando el cliente quisiera agregar o quitar algo de su sistema, sabía que contaba con la presencia de su programador en cualquier momento. Esto nos lleva a pensar que no existían miles de personas que se dedicaran a programar, ni mucho menos que tuviesen una computadora en su hogar para seguir laborando. Como esto no sucedía, el objetivo del cliente con el programador se cumplía de forma directa, es decir, el cliente pedía y el programador lo resolvía.

En la actualidad, existen miles de personas que se dedican a programar tanto laboralmente (profesionista) como académicamente (estudiante). Y no siempre son las mismas personas las que programan, sino que están cambiando regularmente, esto en el ámbito laboral.

Debido a lo anterior, con el paso del tiempo se han creado varias herramientas para modelar los diseños requeridos por el cliente. En el presente trabajo se hablará del lenguaje que ayuda en el modelado (UML) más actual y utilizado a nivel mundial para modelar los sistemas, con ello los programadores ya tienen una visión mas completa de los requerimientos de una empresa. Lógicamente éste lenguaje (UML) ayudará a organizar y planear mucho mejor un análisis, para un diseño más específico y un programa sin tantos errores.

Una de las causas por las cuales se eligió este lenguaje, es por que al ir realizando el análisis se pudo ir adelantando el diseño y visualizar gráficamente los posibles errores que pudiera haber tenido el programa. Además de que estuvo involucrado siempre un usuario, ya que fue una parte fundamental del desarrollo.

En el APÉNDICE F, se puede tener una visión más general de la herramienta UML.

Los modelos que se presentarán en este desarrollo de software serán:

- Modelamiento de Clases (unido a este el Modelo Conceptual)
- Casos de Uso
- Diagrama de Secuencias

Para el caso del Modelamiento de Clases, se dice que unido a este se encuentra el Modelo Conceptual, ya que el modelo de clases esta diseñado a partir de los modelos conceptuales.

En los casos de uso, se describen las acciones del sistema, es decir, se diseñan modelos desde el punto de vista del usuario. Con este tipo de modelo se le da al usuario la mayor confianza, ya que es el tipo de modelado al que más se le acerca a cualquier persona, incluso no es necesario saber computación.

El diagrama de secuencias muestra la mecánica de las fases de los casos de uso, basados en el tiempo, es decir, cada caso de uso tiene un tiempo determinado, el cual tiene seguimiento, ya sea para con otro caso de uso o para hacer un ciclo.

1.5. Tipos de programación

Conforme pasa el tiempo, el creciente empleo de la computadora ha conducido un costo elevado del desarrollo de software específico, ya que para algunos empresarios es más sencillo el mantenimiento de sus desarrollos, que aportar capital para renovarlo. Con los altos costos del mantenimiento de las

aplicaciones en producción normal, también ha surgido la necesidad de mejorar la productividad del personal de programación.

Afortunadamente, gracias a esta crisis la respuesta ha sido una mejora de la programación, aunque no se le explote al máximo. En la actualidad se emplean más a menudo dos tipos de programación: “Estructurada” y “Orientada a Objetos”, las cuales han ido evolucionando conforme a los lenguajes de programación van asumiendo nuevas propiedades y ventajas.

Con el paso del tiempo, la Programación Estructurada se ha convertido en la forma de programar más usual, ya que su secuencia es lineal, sólo tiene una entrada y una salida, además de que en la mayoría de los lenguajes de programación podemos utilizar las estructuras básicas de control. (Ver APÉNDICE G)

En cambio la Programación Orientada a Objetos tiene una regla importante que se debe considerar: “estar basado en objetos, basado en clases y capaz de tener herencia de clases”, su secuencia no es lineal, y puede tener varias salidas, y algunos lenguajes de programación no contienen características para soportar la POO. (Ver APÉNDICE H)

Hasta ahora se han descrito dos tipos de programación que se utilizan en la actualidad. En el desarrollo del sistema en mención se trabajará con programación estructurada, esto por que el sistema en su ciclo de vida tiene un tiempo determinado que no se cubriría programando orientado a objetos.

1.6. Lenguajes de programación

Es extraño que siendo los lenguajes de programación las herramientas bases para desarrollar software, y que los desarrolladores los usen, donde muchos aun

desconozcan que son y como usarlos. Al parecer se ha creado un juicio de superioridad sobre la programación, que hace creer que es para personas con capacidades muy superiores a las de otros.

Para empezar, un lenguaje como se sabe sirve para comunicamos. En el mundo de la informática, un lenguaje de programación lo utilizamos para comunicarnos con la máquina u ordenador, es decir, mediante una serie de instrucciones la computadora realiza lo que dicho programador requiere.

Un lenguaje de programación es un conjunto de símbolos y caracteres, que para la computadora son órdenes que permiten comunicación de humano a computador.

Es cuantiosa la cantidad de lenguajes de programación que existen en la actualidad, pero cada uno de ellos se ha creado específicamente para algo en concreto. En el caso del presente proyecto se ha elegido utilizar Visual Basic versión 6.0, debido a que las empresas con las que he estado laborando, utilizan esta programación visual con mayor soporte a nivel mundial, se puede obtener ayuda e información por diversos medios, y es realmente sencillo encontrar desarrolladores especializados, la facilidad de aprendizaje es muy comprensible y su interfaz es amigable al programador. (Ver APÉNDICE I)

Una de las causas por las cuales se crean los lenguajes de programación, es precisamente la gran dificultad que se requiere para utilizarlos. Tal es el caso de los lenguajes de programación de los años 70s, que a nuestras fechas es muy poca la población del mundo de la informática que conoce como programar en estos complejos lenguajes de programación.

No quiere decir que no funcionen los lenguajes, sino simplemente con el pasar de los años se han creado algunos programas que se acercan a lo mas elemental, sobre todo por la facilidad que tienen dichos lenguajes.

1.7. Teoría de Autómatas

"Esta teoría provee modelos matemáticos que formalizan el concepto de computadora o algoritmo de manera suficientemente simplificada y general para que se puedan analizar sus capacidades y limitaciones. Algunos de estos modelos juegan un papel central en varias aplicaciones de las ciencias de la computación, incluyendo procesamiento de texto, compiladores, diseño de hardware e inteligencia artificial.

Los tres principales modelos son los autómatas finitos, autómatas con pila y máquinas de Turing, cada uno con sus variantes deterministas y no deterministas. Los autómatas finitos son buenos modelos de computadoras que tienen una cantidad limitada de memoria, los autómatas con pila modelan los que tienen gran cantidad de memoria pero que solo pueden manipularla a manera de pila (el último dato almacenado es el siguiente leído), y las máquinas de Turing modelan las computadoras que tienen una gran cantidad de memoria almacenada en una cinta. Estos autómatas están estrechamente relacionados con la teoría de lenguajes formales; cada autómata es equivalente a una gramática formal, lo que permite reinterpretar la jerarquía de Chomsky en términos de autómatas."[5]

Aspectos importantes de la teoría de autómatas.[6]

1. La teoría de los autómatas juega un rol muy importante cuando se hace un software para diseño y chequeo detrás de un circuito digital.
2. El "analizador léxico" del compilador típico, es el compilador del componente que rompe la entrada del texto dentro de la unidad lógica como un identificador, palabras clave y puntuaciones.

[5] http://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_computaci%C3%B3n#Teor.C3.ADa_de_aut.C3.B3matas

[6] Discrete structures and autómata theory, Rakesh Dube, Adesh Pandey, Ritu Gupta, Ed. Alpha Science International Ltd, India 2006.

3. Se escanea el texto del cuerpo del software como una colección de páginas web y encuentra las ocurrencias de palabras, frases u otros patrones.
4. La teoría de los autómatas es la llave del software para verificar sistemas de todo tipo que tienen un número finito de estados distintos, como la comunicación de protocolos o el protocolo de seguridad de intercambio de información.
5. La teoría de los autómatas utiliza más el concepto de software "Procesamiento Natural de Lenguaje".

Los teoría de los autómatas nos proporciona el lenguaje para la especificación de procesos algorítmicos, se divide en tres módulos y se muestran en la figura 1.

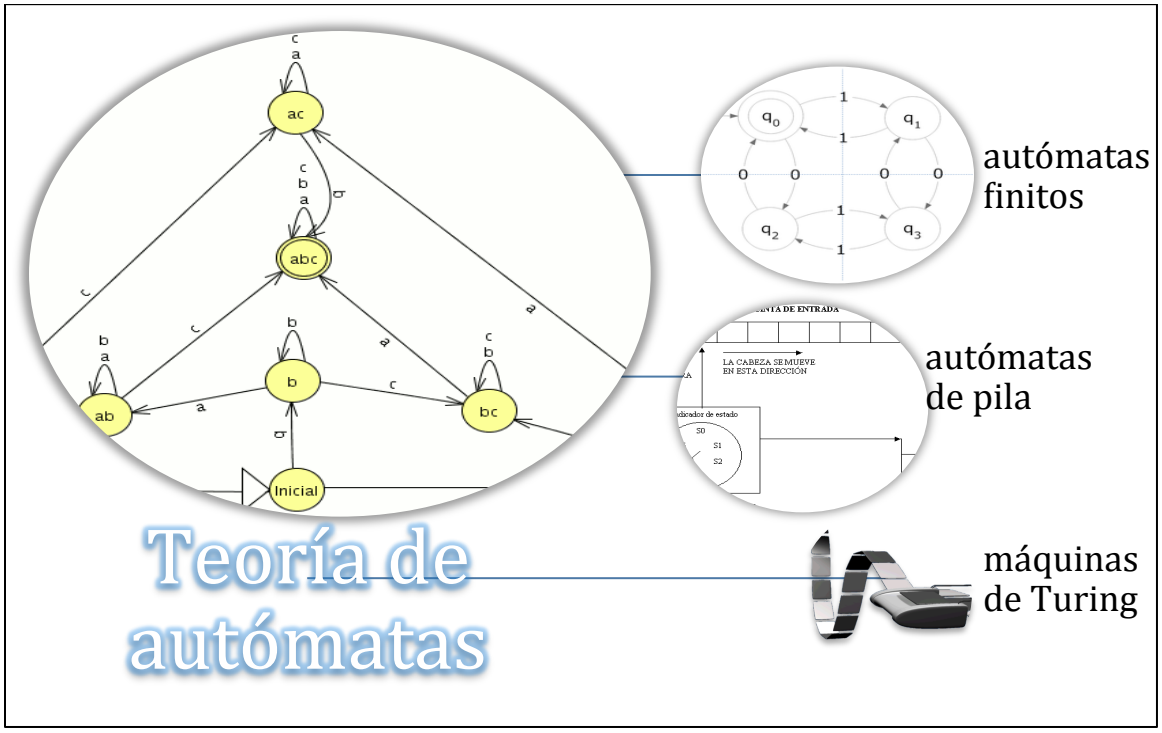


Figura 1. Modelos de teoría de autómatas.

Autómatas finitos.[7]

Un autómata finito (AF) o máquina de estado finito es un modelo matemático que realiza cómputos en forma automática sobre una entrada para producir una salida.

Este modelo está conformado por un alfabeto, un conjunto de estados y un conjunto de transiciones entre dichos estados. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

La finalidad de un autómata finito es el hábito de utilizar expresiones regulares para fines prácticos, especialmente el uso de analizadores léxicos y la búsqueda y reemplazo de texto. Los autómatas finitos se dividen en dos importantes modelos que se muestran en la figura 2.

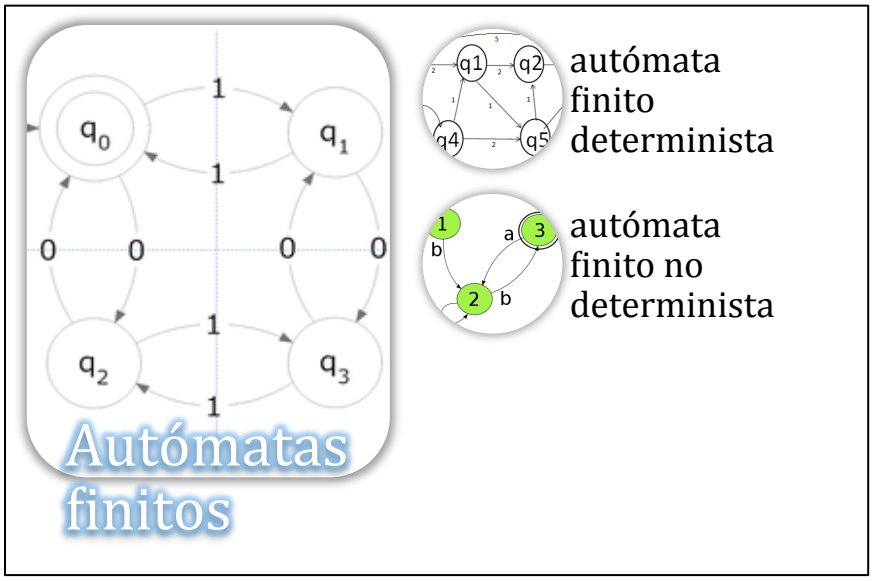


Figura 2. Los autómatas finitos y su división.

[7] http://es.wikipedia.org/wiki/Aut%C3%B3mata_finito

Autómatas finitos deterministas y no deterministas.[8]

Un autómata finito determinista contiene una función de transición que indica a qué estado se va a pasar sabiendo cuál es el estado actual y el símbolo que se está leyendo. Es importante notar que se trata de una función y no simplemente una relación; esto implica que para un estado y un símbolo del alfabeto dados, habría un y sólo un estado siguiente. Esta característica, que permite saber siempre cuál será el siguiente estado, se llama determinismo. La definición dada arriba corresponde a los autómatas finitos deterministas, abreviado "AFD".

Los autómatas finitos no determinísticos son una extensión a los autómatas finitos deterministas, su función es la de permitir que de cada nodo del diagrama de estados salga un número de flechas mayor o menor que el alfabeto de entrada. Así, se puede permitir que falte la flecha correspondiente a alguno de los símbolos del alfabeto, o bien que haya varias flechas que salgan de un sólo nodo con la misma etiqueta. Inclusive se permite que las transiciones tengan como etiqueta palabras de varias letras o hasta la palabra vacía. A estos autómatas finitos se les llama no determinísticos o no deterministas (abreviado AFN). En los autómatas no determinísticos se presenta una dificultad para poder saber qué camino tomar a partir de un estado dado cuando se presenta un símbolo, pues puede haber varias opciones.

Autómata con pila.[9]

Un autómata con pila, autómata a pila o autómata de pila es un modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce. El lenguaje que reconoce un autómata con pila pertenece

[8] http://homepages.mty.itesm.mx/rbrena/AyL.html
[9] http://es.wikipedia.org/wiki/Aut%C3%B3mata_con_pila

al grupo de los lenguajes libres de contexto en la clasificación de la Jerarquía de Chomsky.

Los autómatas de pila, en forma similar a como se usan los autómatas finitos, también se pueden utilizar para aceptar cadenas de un lenguaje definido sobre un alfabeto A. Los autómatas de pila pueden aceptar lenguajes que no pueden aceptar los autómatas finitos. Un autómata de pila cuenta con una cinta de entrada y un mecanismo de control que puede encontrarse en uno de entre un número finito de estados. Uno de estos estados se designa como estado inicial, y además algunos estados se llaman de aceptación o finales. A diferencia de los autómatas finitos, los autómatas de pila cuentan con una memoria auxiliar llamada pila. Los símbolos (llamados símbolos de pila) pueden ser insertados o extraídos de la pila, de acuerdo con el manejo last-in-first-out (LIFO). Las transiciones entre los estados que ejecutan los autómatas de pila dependen de los símbolos de entrada y de los símbolos de la pila. El autómata acepta una cadena x si la secuencia de transiciones, comenzando en estado inicial y con pila vacía, conduce a un estado final, después de leer toda la cadena x. En la figura 3 se muestran los modelos de los autómatas de pila.

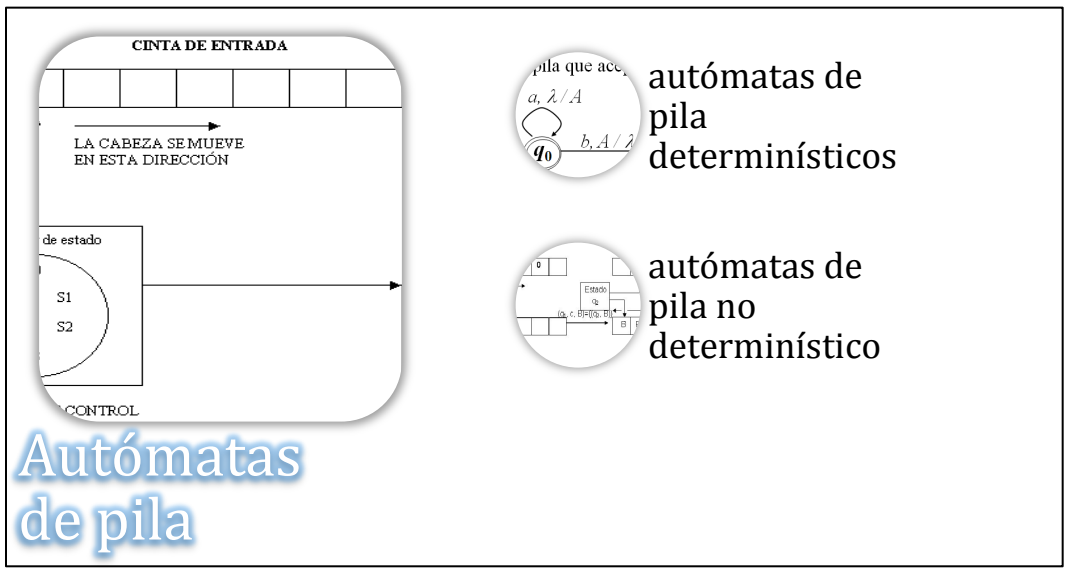


Figura 3. Los autómatas de pila y su división.

Autómatas de pila determinísticos y no determinísticos.

A diferencia de un autómata finito o una máquina de Turing, la definición básica de un autómata con pila es de naturaleza no determinista, pues la clase de los autómatas con pila deterministas (AFPD), a diferencia de lo que ocurría con aquellos modelos, tiene una potencia descriptiva estrictamente menor. Para calificar a un autómata con pila como determinista deben darse dos circunstancias; en primer lugar, por supuesto, que en la definición de cada componente de la función de transición existan un único elemento lo que da la naturaleza determinista. Pero eso no es suficiente, pues además puede darse la circunstancia de que el autómata esté en el estado inicial y en la pila aparezca el símbolo inicial de la pila, entonces, si existe una definición de transición posible para algún símbolo cualquiera del alfabeto de entrada, pero, además existe otra alternativa para la palabra vacía, también esto es una forma de no determinismo, pues podemos optar entre leer un símbolo o no hacerlo. Por eso, en autómata determinista no debe existir transición posible con lectura de símbolo si puede hacerse sin ella, ni al contrario. Un autómata finito con pila no determinista (AFPN) consta de los mismos parámetros de un AFPD.

Máquina de Turing.[10]

Una máquina de Turing (MT) es un modelo computacional que realiza una lectura/escritura de manera automática sobre una entrada llamada cinta, generando una salida en esta misma. Este modelo está formado por un alfabeto de entrada y uno de salida, un símbolo especial llamado blanco, un conjunto de estados finitos y un conjunto de transiciones entre dichos estados. Su funcionamiento se basa en una función de transición, que recibe un *estado inicial* y una cadena de caracteres (la cinta, la cual puede ser infinita) pertenecientes al alfabeto de entrada. La máquina va leyendo una celda de la cinta en cada paso, borrando el símbolo en el que se encuentra posicionado su

[10] http://es.wikipedia.org/wiki/M%C3%A1quina_de_Turing

cabezal y escribiendo un nuevo símbolo perteneciente al alfabeto de salida, para luego desplazar el cabezal a la izquierda o a la derecha (solo una celda a la vez). Esto se repite según se indique en la función de transición, para finalmente detenerse en un *estado final* o *de aceptación*, representando así la salida.

La máquina de Turing consta de un cabezal lector/escritor y una cinta infinita en la que el cabezal lee el contenido, borra el contenido anterior y escribe un nuevo valor. Las operaciones que se pueden realizar en esta máquina se limitan a:

- Avanzar el cabezal lector/escritor hacia la derecha.
- Avanzar el cabezal lector/escritor hacia la izquierda.

El cómputo es determinado a partir de una tabla de estados de la forma: (estado, valor) → (nuevo estado, nuevo valor, dirección). En la figura 4 se muestran los modelos de la máquina de Turing.

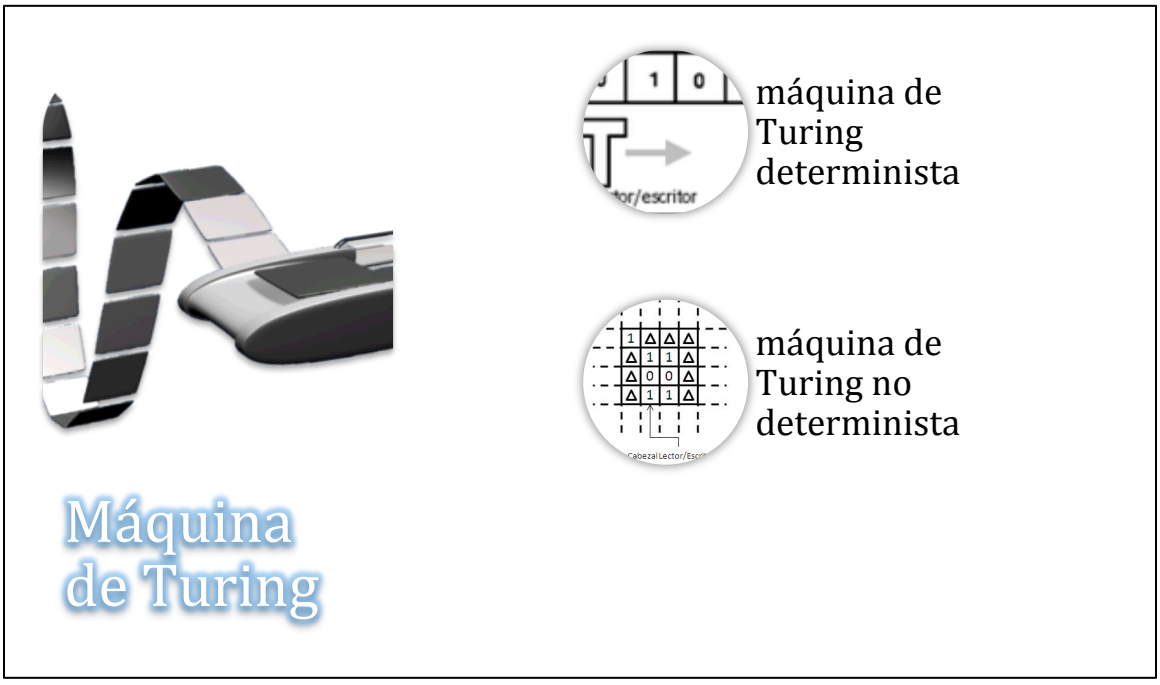


Figura 4. La máquina de Turing y su división.

Máquina de Turing determinista y no determinista.

La entrada de una máquina de Turing viene determinada por el estado actual y el símbolo leído, un par (estado, símbolo), siendo el cambio de estado, la escritura de un nuevo símbolo y el movimiento del cabezal, las acciones a tomar en función de una entrada. En el caso de que para cada par (estado, símbolo) posible exista a lo sumo una posibilidad de ejecución, se dirá que es una máquina de Turing determinista, mientras que en el caso de que exista al menos un par (estado, símbolo) con más de una posible combinación de actuaciones se dirá que se trata de una máquina de Turing no determinista.

Aplicaciones de las expresiones regulares y los autómatas finitos.[11]

Los autómatas finitos se usan frecuentemente en los problemas que implican el análisis de cadenas de caracteres. Tales problemas incluyen problemas de búsqueda e identificación, tales como la búsqueda de la existencia de una cadena en un fichero o el reconocimiento de cadenas de entrada que satisfagan ciertos criterios. Un autómata finito es, él mismo, un modelo de un procedimiento para reconocimiento de cadenas por medio de la expresión regular asociada. Por tanto, en la búsqueda de una cadena en un fichero, podemos aplicar el autómata finito de forma sistemática a las cadenas del fichero hasta que se acepte la cadena o se termina el fichero.

Un problema común en la programación de computadoras es el de tener la seguridad de que los datos de entrada de un programa son correctos. Por ejemplo, si se espera un entero sin signo como dato de entrada y el usuario confunde uno de los dígitos con un carácter no numérico, se puede dar todo tipo de resultados impropios, desde una terminación anormal hasta el cálculo de resultados incorrectos (basura dentro, basura fuera). La programación cuidadosa pretende construir un programa a “prueba de balas”, incluyendo unas rutinas de

[11] Teoría de autómatas y lenguajes formales, Dean Kelley, Ed. Prentice Hall, Madrid 2001.

entrada que analicen la información introducida por el usuario y, de alguna forma, prevenir que se aplique información incorrecta al programa. Si pudiéramos construir un autómata finito que aceptara solamente las cadenas que representan información correcta, entonces tendríamos un modelo para dicha rutina de entrada. Puesto que los autómatas finitos se corresponden con las expresiones regulares, el problema se reduce a especificar la información correcta por medio de expresiones regulares.

Estructura de los autómatas.

La teoría de autómatas que se explica anteriormente con los autómatas finitos, de pila y máquina de Turing en conjunto con sus contenidos, son técnicamente los modelos que se siguen en algunas fases de programación, no hay que olvidar estos mismos tienen una relación muy estrecha con los analizadores léxico y sintáctico. En la figura 5 se observa la estructura que denota autómatas que podrían utilizar un analizador léxico o sintáctico.

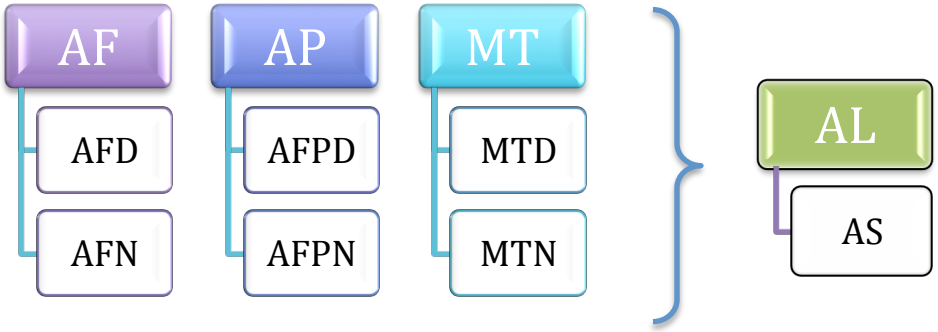


Figura 5. Estructura de los autómatas.[12]

Dentro del desarrollo de software ENDER se demostrará un autómata finito que se basa en un analizador sintáctico (parser) para crear una herramienta que determina bajo palabras reservadas un análisis.

[12] AF. Autómata finito, AFD. Autómata finito determinista, AFN. Autómata finito no determinista, AP. Autómata de pila, AFPD. Autómata de pila determinístico, AFPN. Autómata de pila no determinista, MT. Máquina de Turing, MTD. Máquina de Turing determinista, MTN. Máquina de Turing no determinista, AL. Analizador léxico, AS. Analizador sintáctico.

Analizador léxico y sintáctico.[13]

El **analizador léxico** es la primera fase de la que consta un compilador. La parte del compilador que realiza el análisis léxico se llama analizador léxico (AL), scanner o explorador. La tarea básica que realiza el AL es transformar un flujo de caracteres de entrada en una serie de componentes léxicos o tokens. Se encargaría, por tanto, de reconocer identificadores, palabras clave, constantes, operadores, etc.

La secuencia de caracteres que forma el token se denomina lexema. No hay que confundir el concepto de token con el de lexema. A un mismo token le pueden corresponder varios lexemas. Por ejemplo, se pueden reconocer como tokens de tipo ID a todos los identificadores. Aunque para analizar sintácticamente una expresión, sólo nos hará falta el código de token, el lexema debe ser recordado, para usarlo en fases posteriores dentro del proceso de compilación. El AL es el único componente del compilador que tendrá acceso al código fuente. Por tanto, debe de encargarse de almacenar los lexemas para que puedan ser usados posteriormente. Esto se hace en la tabla de símbolos. Por otro lado, debe enviar al analizador sintáctico, aparte del código de token reconocido, la información del lugar dónde se encuentra almacenado ese lexema (por ejemplo, mediante un apuntador a la posición que ocupa dentro de la tabla de símbolos). Posteriormente, en otras fases del compilador, se irá completando la información sobre cada item de la tabla de símbolos.

Si durante la fase de análisis léxico, el AL se encuentra con uno o más lexemas que no corresponden a ningún token válido, debe dar un mensaje de error léxico e intentar recuperarse. Finalmente, puesto que el AL es el único componente del compilador que tiene contacto con el código fuente, debe encargarse de eliminar los símbolos no significativos del programa, como espacios en blanco,

[13] Teoría de autómatas y lenguajes formales, Navarrete Sánchez Isabel, Martínez Béjar Rodrigo, Ed. Diego Marin Librero Editor, S.L. España 2003.

tabuladores, comentarios, etc.

Es conveniente siempre separar esta fase de la siguiente (análisis sintáctico), por razones de eficiencia. Además, esto permite el uso de representaciones diferentes del programa fuente, sin tener que modificar el compilador completo.

El **analizador sintáctico** es la segunda fase de la que consta un compilador. La parte del compilador que realiza el análisis sintáctico se llama analizador sintáctico o parser. Su función es revisar si los tokens del código fuente que le proporciona el analizador léxico aparecen en el orden correcto (impuesto por la gramática), y los combina para formar unidades gramaticales , dándonos como salida el árbol de derivación o árbol sintáctico correspondiente a ese código fuente. De la forma de construir este árbol sintáctico se desprenden los dos tipos de analizadores sintácticos existentes:

- Cuando se parte del axioma de la gramática y se va descendiendo, utilizando derivaciones más a la izquierda, hasta conseguir la cadena de entrada, se dice que el análisis es descendente.
- Por el contrario, cuando se parte de la cadena de entrada y se va generando el árbol hacia arriba mediante reducciones más a la izquierda (derivaciones más a la derecha), hasta conseguir la raíz o axioma, se dice que el análisis es ascendente.

Si el programa no tiene una estructura sintáctica correcta, el analizador sintáctico no podrá encontrar el árbol de derivación correspondiente y deberć•a dar mensaje de error sintáctico. La división entre análisis léxico y sintáctico es algo arbitraria. Generalmente se elige una división que simplifique la tarea completa del análisis. Un factor para determinar cómo realizarla es comprobar si una construcción del lenguaje fuente es inherentemente recursiva o no. Las construcciones léxicas no requieren recursión, mientras que las sintácticas suelen requerirla.

Las gramáticas libres de contexto (GLC) formalizan la mayoría de las reglas recursivas que pueden usarse para guiar el análisis sintáctico. Es importante destacar, sin embargo, que la mayor parte de los lenguajes de programación pertenecen realmente al grupo de lenguajes dependientes del contexto.

1.8. Conceptualización y análisis del sistema de la herramienta para documentar código fuente (ENDER)

Como se mencionó al inicio de este documento la principal razón por la que se ha generado este desarrollo de software, es por que en el centro de cómputo de la FES Aragón se han creado sistemas informáticos en Visual Basic versión 5.0 y 6.0, que no han tenido la documentación de código fuente requerida.

Motivo por el cual se presento la propuesta de generar un desarrollo de software que coadyuvará al programador a documentar dichos sistemas de información, y de esta forma en un futuro retomar dichos proyectos y corregirlos o simplemente aplicar la reingeniería sobre estos mismos.

Al inicio de la propuesta se realizo una reunión con los programadores del área y se recabaron los puntos más importantes para comenzar con un análisis. Dentro de los puntos a tomar en cuenta se pudieron recabar los siguientes:

- Que se genere un árbol de nodos en donde se visualicen los formularios, módulos y clases.
- Cada formulario, módulo o clase del árbol, debe contener sus propios métodos, funciones, procedimientos, etc.
- Debe realizar un análisis del estado de las variables, ya sean no declaradas o no utilizadas.
- Conforme se vaya formando el árbol de nodos se debe realizar una inspección por módulo, y en un apartado ir llenando un grid donde se irán visualizando los formularios, módulos y clases, para que dentro de estos

se coloque el número de procedimientos, funciones y variables que existen.

- Se debe agregar una caja de textos con etiqueta “Comentario general”, en donde el usuario pueda poner un texto que será anexado en el encabezado del código fuente por módulo.
- Este desarrollo informático debe ser capaz de encontrar valores por referencia y por valor, a estos mismos se les debe encadenar una caja de textos para que el usuario pueda anexar comentarios por procedimiento y función, si el usuario así lo requiere.

Después de recabar los requerimientos del usuario, se realizó una propuesta de calendario para cada fase del desarrollo de software como se muestra en la figura 1.

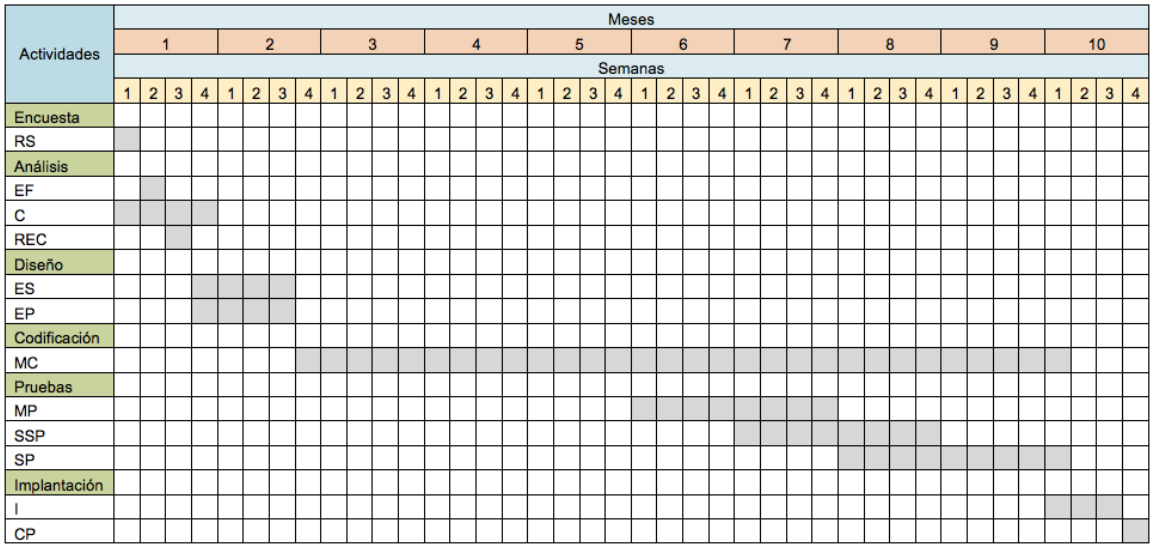


Figura 1. Calendario de actividades.

Las actividades que se desarrollaron y que en el calendario se encuentran abreviados, se muestran en las tablas 1 y 2.

Encuesta	Análisis	Diseño
· Requerimientos del usuario (RS).	· Especificación funcional (EF). · Calendario (C). · Requerimientos del equipo del cómputo (REC).	· Especificación del sistema (ES). · Especificación del programa (EP).

Tabla1. Desglose de actividades de (parte 1).

Codificación	Pruebas	Implantación
· Módulos codificados (MC).	· Módulos probados (MP). · Subsistemas probados (SSP). · Sistema probado (SP).	· Instalación (I). · Capacitación (CP).

Tabla 2. Desglose de actividades de (parte 2).

Terminado el calendario de actividades del desarrollo de software, se comenzó con la etapa de los requerimientos del equipo de cómputo. Este paso trata tanto de software como de hardware y se desglosa en la tabla 3.

Requerimiento	Observación
Sistema Operativo Windows	A partir de la versión XP
Visual Basic	Versión 6
Procesador superior a Pentium IV	De preferencia la arquitectura debe ser a 32 bits
Memoria RAM superior a 256 Mb	Se recomienda 512 Mb

Tabla 3. Requerimientos de equipo de cómputo para el desarrollo de software.

La herramienta lleva el nombre de ENDER por las siguientes siglas (herramiENta para DocumEntaR código fuente).

Se realizo el análisis para la creación del desarrollo ENDER, el cual se desglosa en los siguientes puntos:

1. Principalmente crear en el desarrollador una cultura informática, para que en un futuro inmediato, los programas sean entendibles, tanto para el desarrollador que lo elaboró, como el que desee hacer algún mantenimiento.

2. Adquirir información de los módulos que se generaron en el proyecto realizado. Como el hecho de obtener en una lista de forma descendente tres diferentes carpetas, en las cuales se encuentran ordenadas las formas, módulos y clases, con las cuales funciona el desarrollo.

3. Visualizar código fuente por medio de selección. Con esto se refiere a que cada que se le de un clic a algún miembro de la lista, se visualizará el código fuente.

4. Dar a conocer al usuario por medio de subniveles el contenido del proyecto, esto quiere decir que dentro de cada una de las carpetas de Formas, Módulos y Clases, se obtendrán de forma separada las variables, constantes, procedimientos, funciones, etc.

5. Conocer visualmente una estadística donde nos informe el contenido de cada una de las carpetas en concreto.

6. Opción a imprimir el código, ya sea por Módulo, Procedimiento o por pantalla.

7. Visualizar el código no utilizado y/o no declarado. Dando a conocer la línea específica donde se encuentran estos errores.

CAPÍTULO 2

DISEÑO DEL SISTEMA A CONSTRUIR

2.1. Algoritmo y Modelo conceptual

Durante el capítulo 1 se encuentra la primer fase del ciclo de vida, donde se hablo de conceptos teóricos y análisis del desarrollo de software ENDER. En el presente capítulo se desarrollará la segunda fase del ciclo de vida que consta del desarrollo del diseño a construir.

Comenzamos por definir que un algoritmo es un conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema.

Antes de comenzar con el diseño del sistema, a continuación se muestra el algoritmo principal en la figura 1, que muestra el parser el cual se ocupara en gran parte del desarrollo de software.

```
función parseo (palabras reservadas, cadenas)
        arreglo [nodos,cadena]
        para i = 0 hasta (longitud(palabras reservadas))
                para cada estado en arreglo [i] hacer
                        si estado incompleto, entonces
                                si la próxima cadena (estado) es terminal entonces
                                        árbol (estado, i, cadena) // no es terminal
                                sino
                                        escaneo (estado, i) // es terminal
                        sino
                                completo (estado, i)
                final
        final
        volver a cadenas
-----------------------

procedimiento árbol (estado, i, cadena)
        para cada (estado, cadena) hacer
                anexa (estado, cadena[j])
        final

-----------------------

procedimiento escaneo (estado, i)
        si palabra reservada [j], entonces
                anexa (estado, cadena[j + i]
        final

-----------------------

procedimento completo (estado, i)
        para cada (estado) en arreglo [j] hacer
                anexa (estado, arreglo [j+i])
        final
```

Figura 1. Algoritmo parser.

Después de tomar como base el algoritmo de parser se puede observar en la figura 2 un modelo conceptual que del diagrama de flujo de la secuencia de parser, aplicado a ENDER.

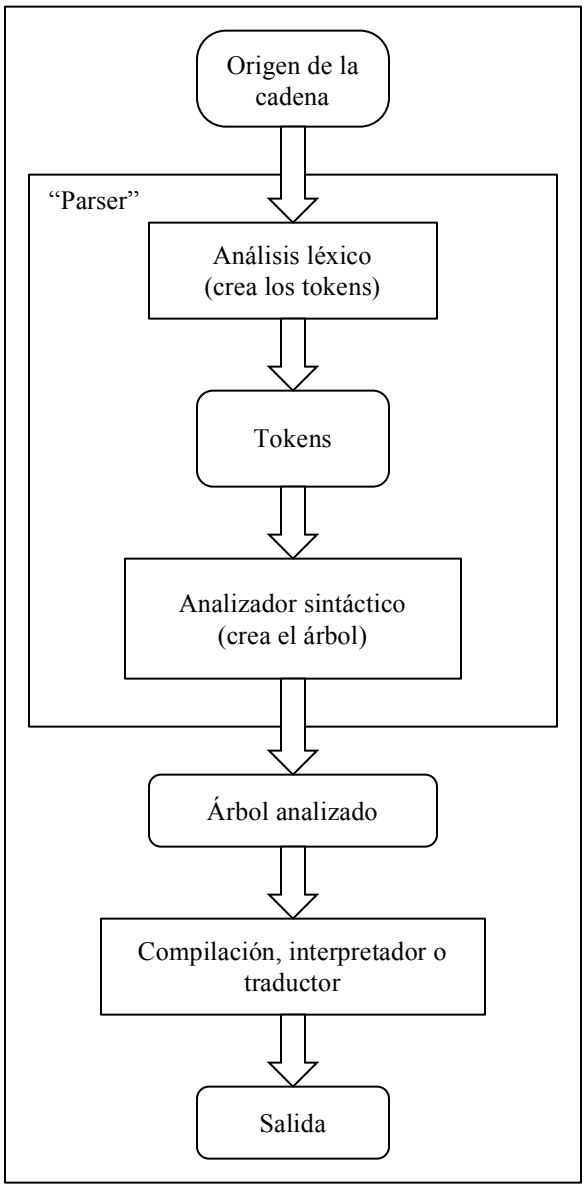


Figura 2. Modelo conceptual de parser.

El modelo conceptual puede entenderse como un mapa de conceptos y sus relaciones, los cuales son considerados fundamentales para la comprensión de lo representado. (Ver esquema representativo de parseo en APÉNDICE L)

Para comenzar con el diseño, se realizaron los modelos conceptuales de ENDER, comenzado por la conceptualización inicial figura 3

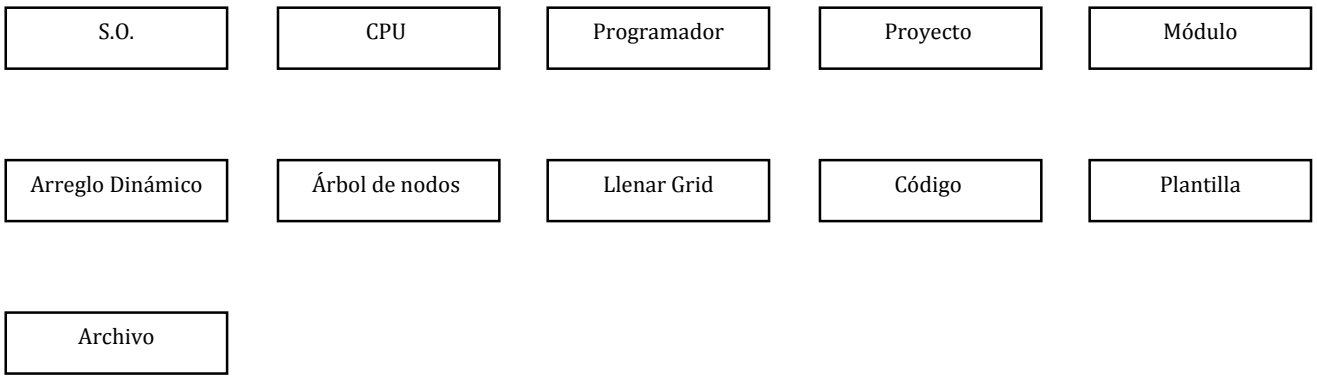
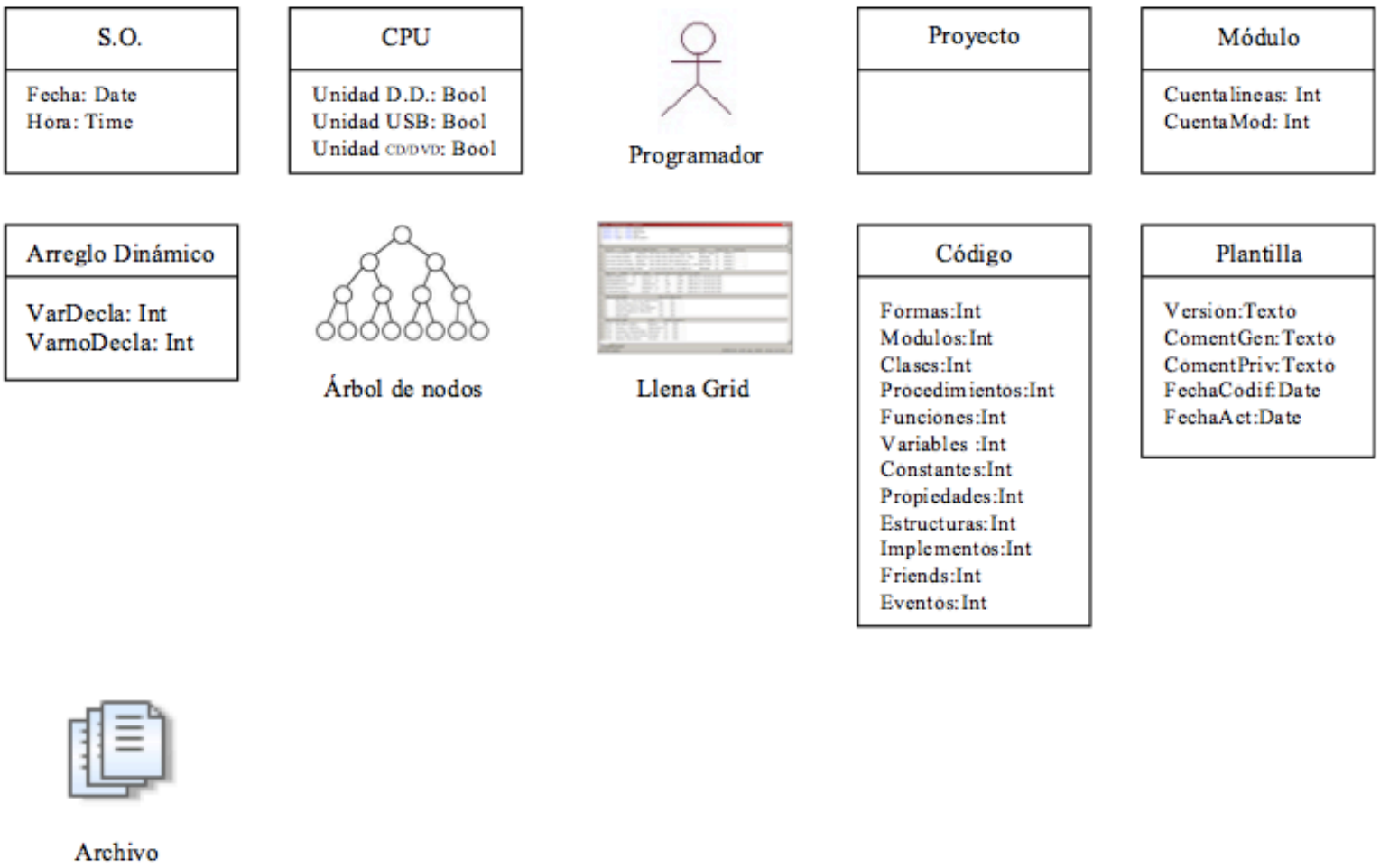


Figura 3. Modelo conceptual inicial.

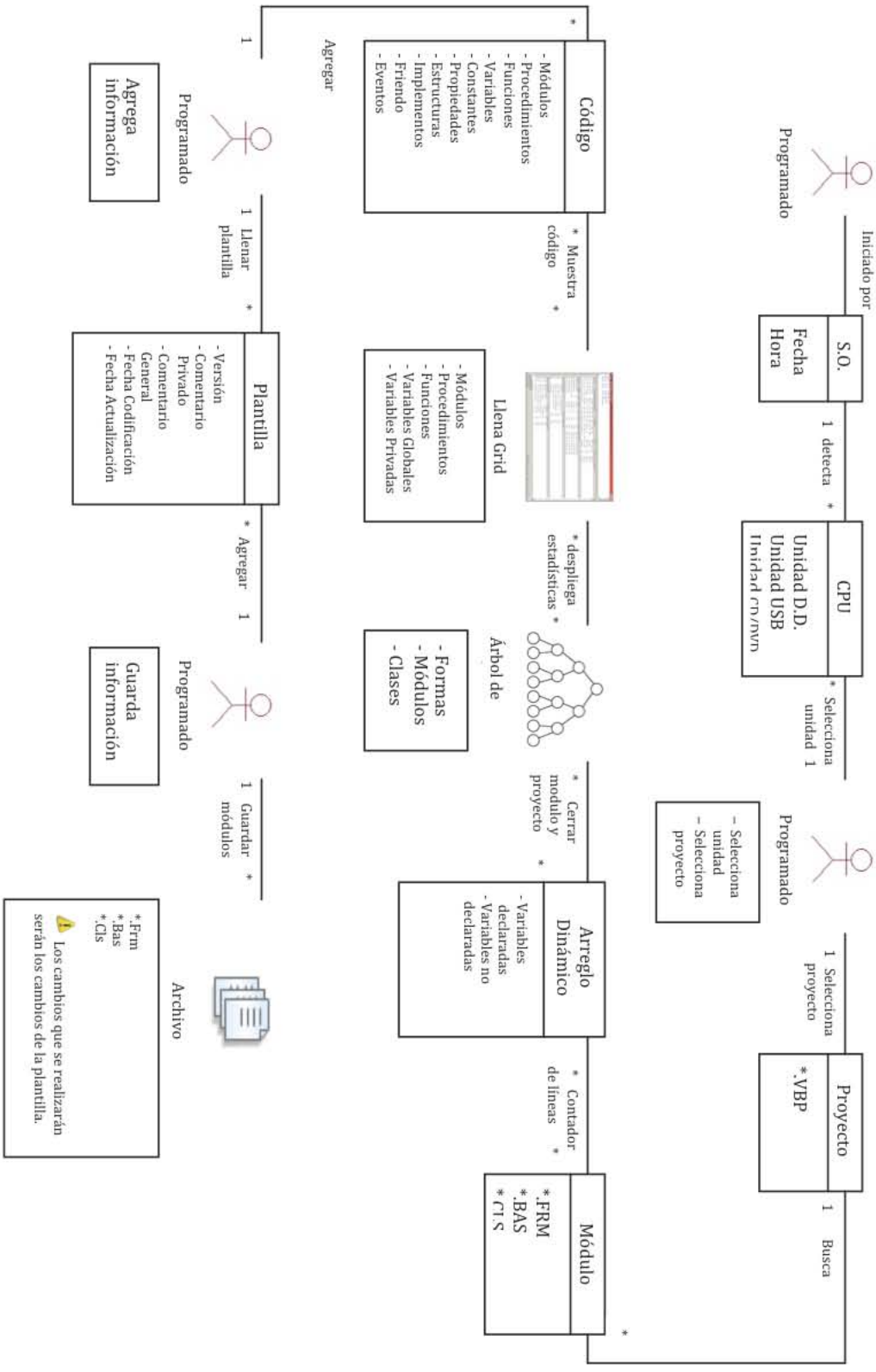
Después de tener el primer modelo conceptual bien definido, se especifican los modelos conceptuales siguientes:

- Muestran atributos.
- Generaliza procedimientos de ENDER.
- Aplicado a ENDER.
- Del dominio de ENDER.

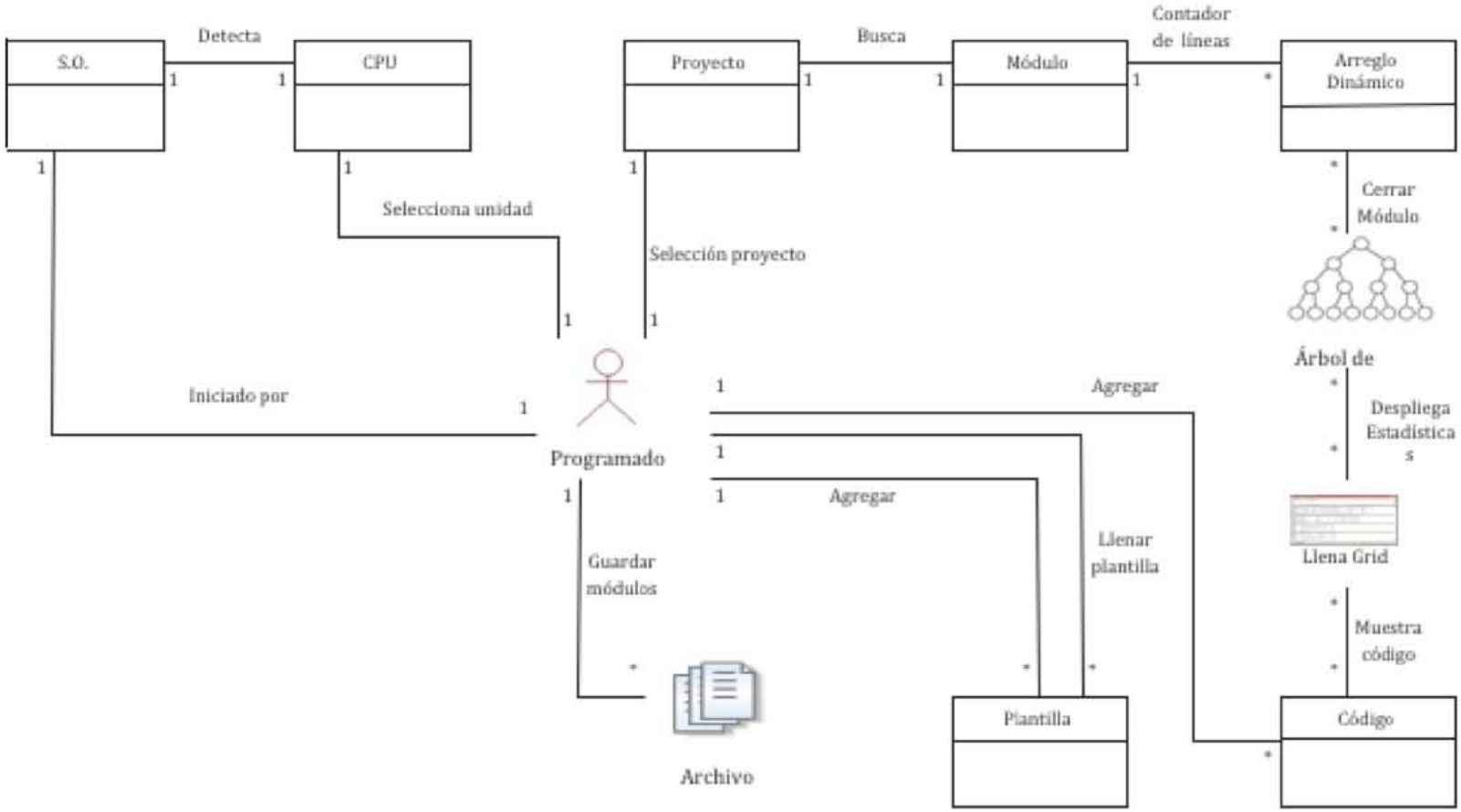
Modelo conceptual que muestra los atributos



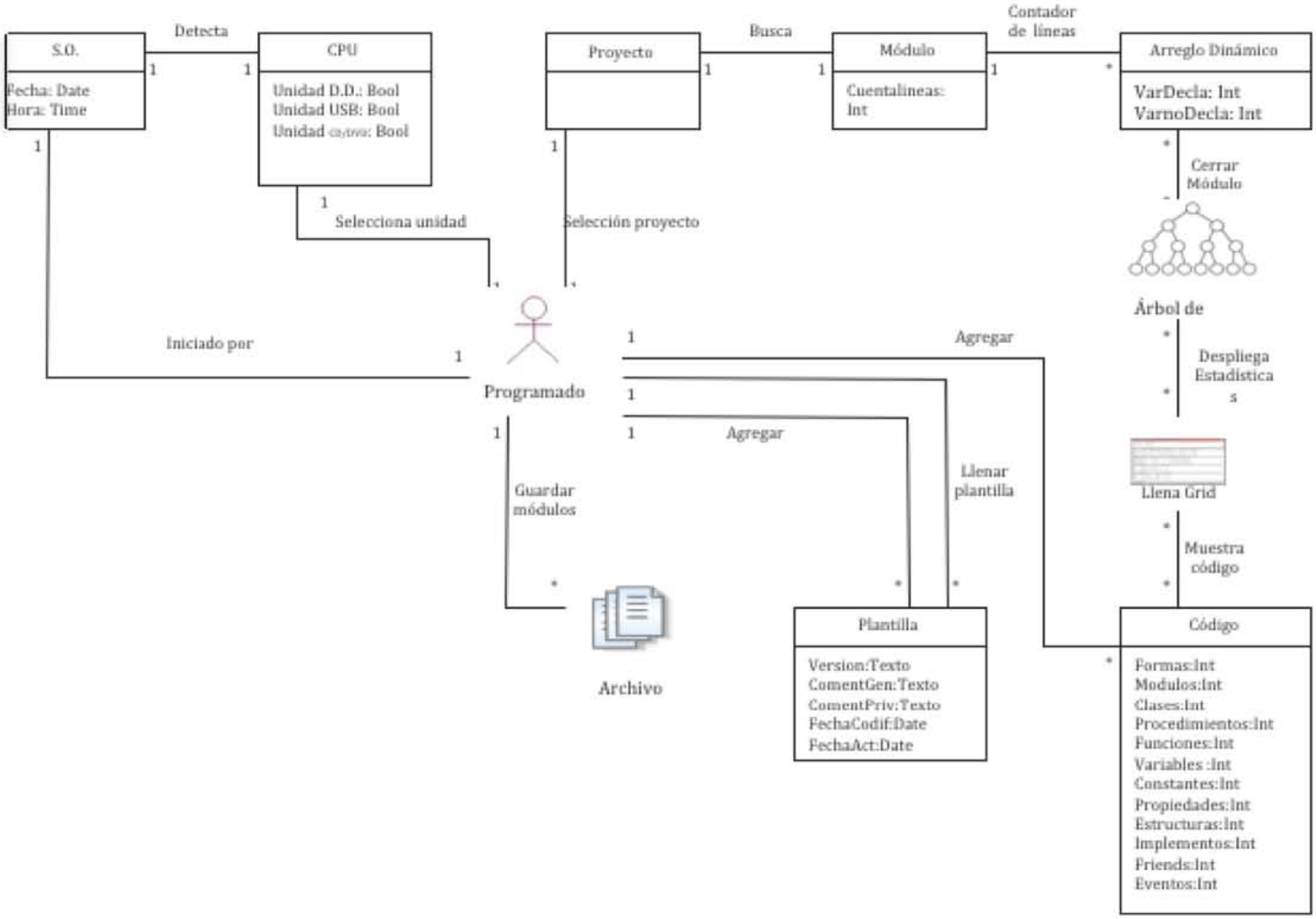
Modelo conceptual que generaliza los procedimientos de ENDER



Modelo conceptual aplicado a ENDER



Modelo conceptual del dominio de ENDER



2.2. Diagrama de casos de uso

De las necesidades recabadas en este desarrollo de software que se requiere construir, es importante mencionar que se hará un sistema con los siguientes 4 casos de uso: Abrir proyecto, analizar proyecto, guardar proyecto y proyecto documentado, los cuales deberán tener las funciones que se enlistan en la siguiente tabla.

Funciones del sistema

No.	Función
	Función del caso de uso abrir proyecto
1	Obtención de las unidades de disco duro y extraíbles que contiene el equipo.
2	Obtención de un proyecto de Visual Basic (versión 5 o 6), en las unidades que tiene el equipo.
	Función del caso de uso analizar proyecto
3	Obtención de formas, módulos y clases del proyecto.
4	Determinación de cuales son las variables que están declaradas y no utilizadas, así como las variables que no están declaradas.
5	Crear árbol de nodos, el cual contiene 3 carpetas: Formas, Módulos y Clases. Cada una de ellas contiene su correspondiente código.
6	Contabilizar cuantos módulos existen en el proyecto, determinando su tipo y contenido.
7	Obtener código fuente mediante la asignación de una selección determinada por el usuario.
	Función del caso de uso guardar proyecto
8	Obtener el texto introducido por el usuario, para anexarlo al código fuente del proyecto.
9	Guardar los cambios efectuados en “Plantilla” en los archivos correspondientes.

Los siguientes diagramas de casos de uso representan las funciones del sistema en cuestión.

Diagrama de caso de uso ENDER

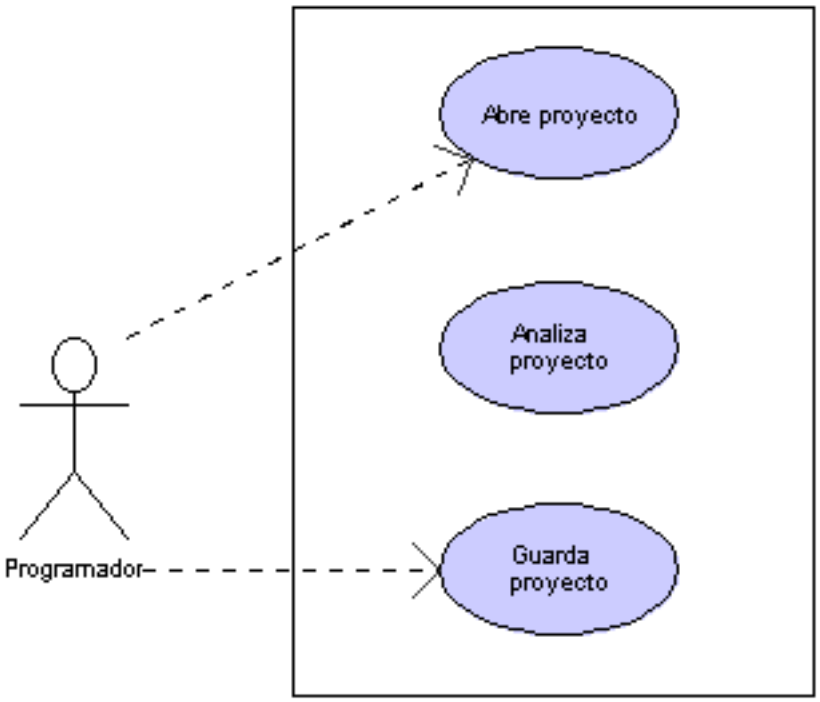


Diagrama de caso de uso abrir proyecto

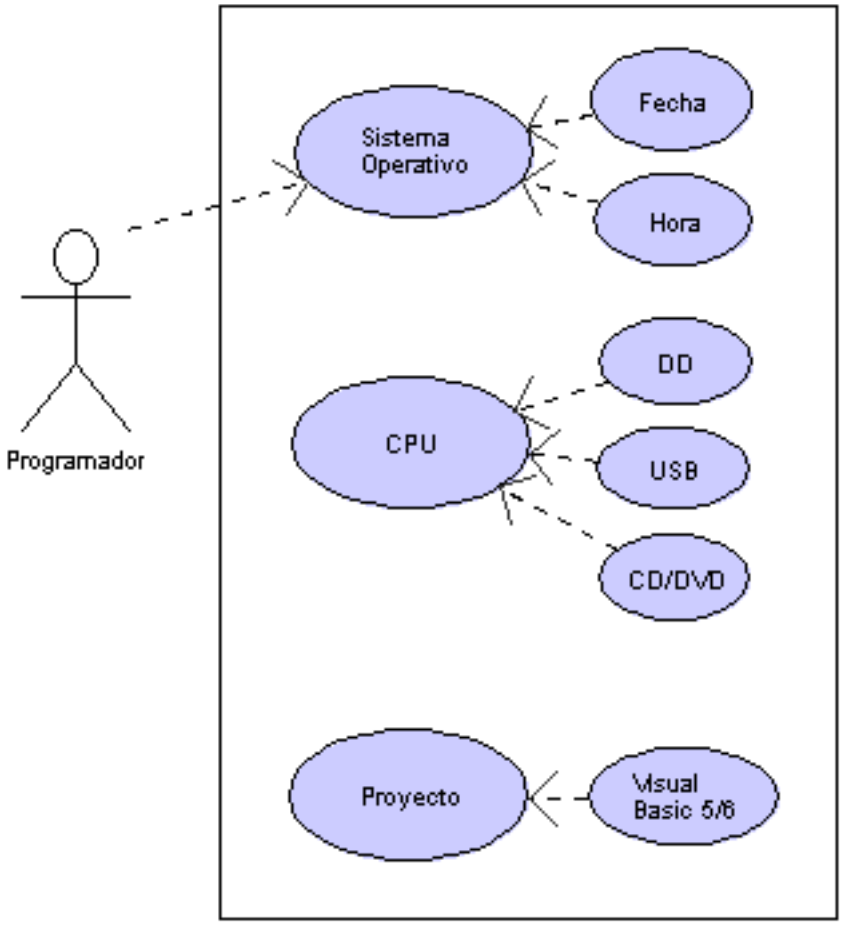


Diagrama de caso de uso analizar proyecto

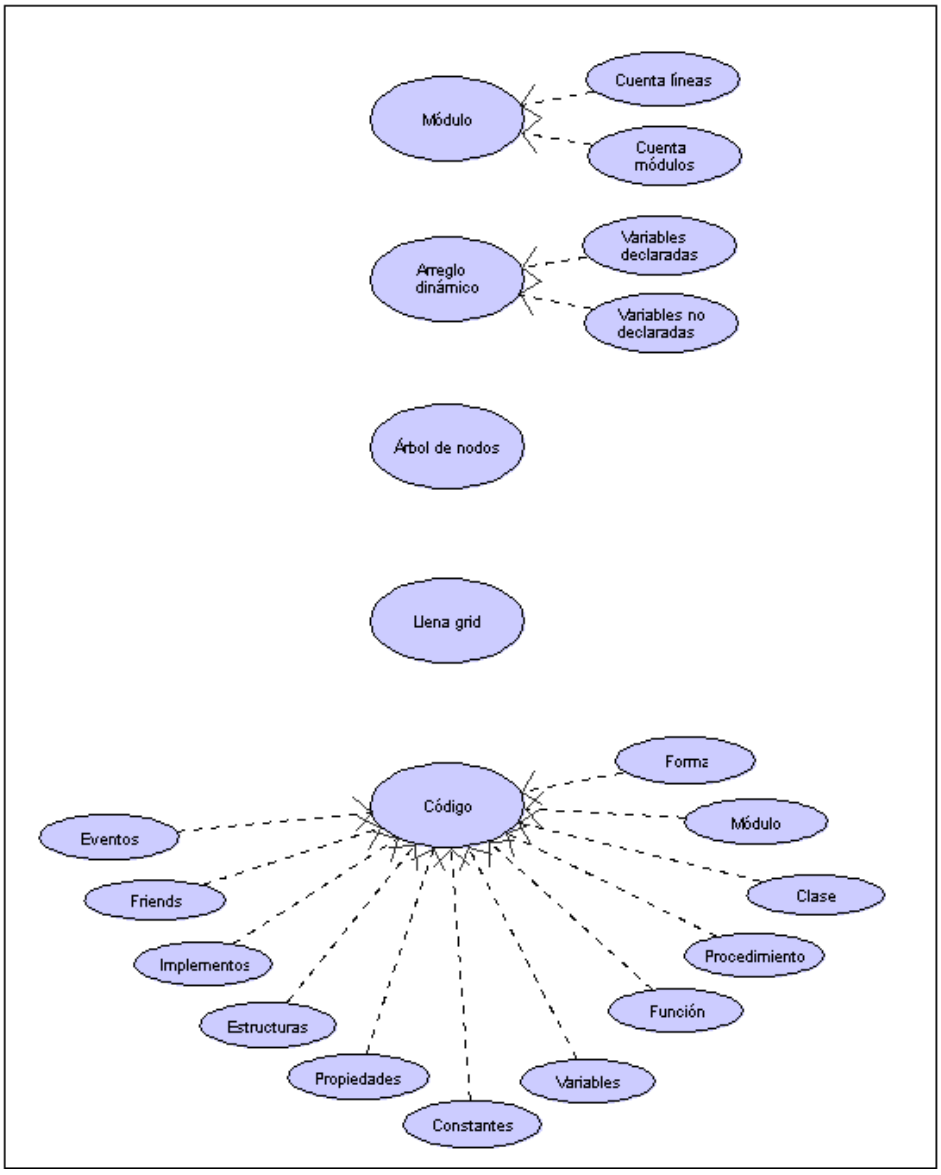
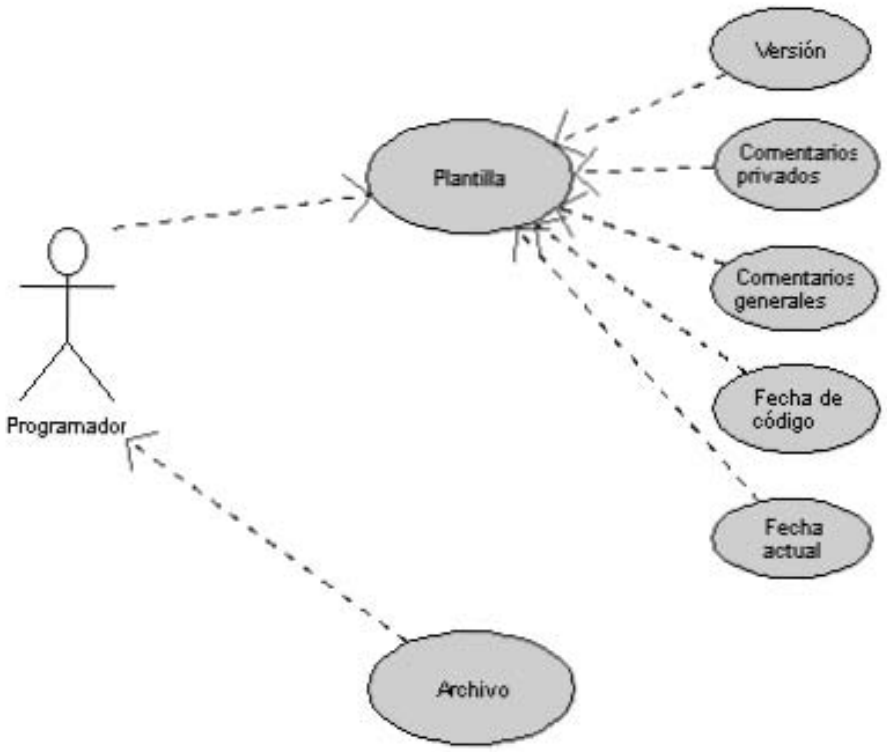


Diagrama de caso de uso guardar proyecto



Caso de uso:	ENDER
Nombre:	ENDER
Actores:	Programador
Propósito:	Entrar al sistema y utilizar la herramienta ENDER
Tipo:	Primario
Descripción:	Cuando el programador entra al sistema, elige un proyecto de Visual Basic (versión 5 o 6) para ser analizado, y después guardado.
Referencias:	1, 2

Curso de Eventos: ENDER

No.	Acción del programador	No.	Respuesta del sistema
1	El caso de uso inicia cuando el programador ejecuta el sistema, y el mismo elige una unidad de almacenamiento	2	Muestra en el sistema a petición del programador las unidades de almacenamiento del equipo de cómputo, para que éste mismo elija una de ellas
3	El programador selecciona un proyecto Visual Basic versión 5 o 6, para ser analizado por el sistema y al final guardar el proyecto documentado	4	El sistema realiza el análisis correspondiente del proyecto elegido, y es cuando el programador documenta al proyecto para poder guardarlo con dichos cambios

Caso de uso:	Abrir proyecto
Nombre:	Abrir proyecto
Actores:	Programador
Propósito:	Una vez dentro del sistema se abre un proyecto Visual Basic, versión 5 o 6.
Tipo:	Primario
Descripción:	Cuando el programador entra al sistema, elige un proyecto de Visual Basic (versión 5 o 6) para ser analizado.
Referencias:	3, 4, 5, 6, 7

Curso de Eventos: Abrir proyecto

No.	Acción del programador	No.	Respuesta del sistema
1	Elegir el proyecto a analizar	2	Organiza en un árbol las formas, módulos y clases que encuentra en el proyecto
3	Observar el ordenamiento del árbol de nodos, de variables declaradas y no declaradas	4	Identifica dentro del proyecto las variables no declaradas y variables sin utilizar
5	Observar el ordenamiento de árbol de nodos, de formas, módulos y clases	6	Ordena dentro de los nodos formas, módulos y clases el contenido de cada uno de ellos
7	Observar la contabilización de módulos que existen en el proyecto	8	Crea un contador del total de módulos que ha encontrado en el proyecto
9	Seleccionar un módulo, variable, o cualquier elemento que se encuentre contenido en el árbol de nodos	10	Muestra el código del elemento elegido, y se agrega si es requerido un comentario a nivel general o específico

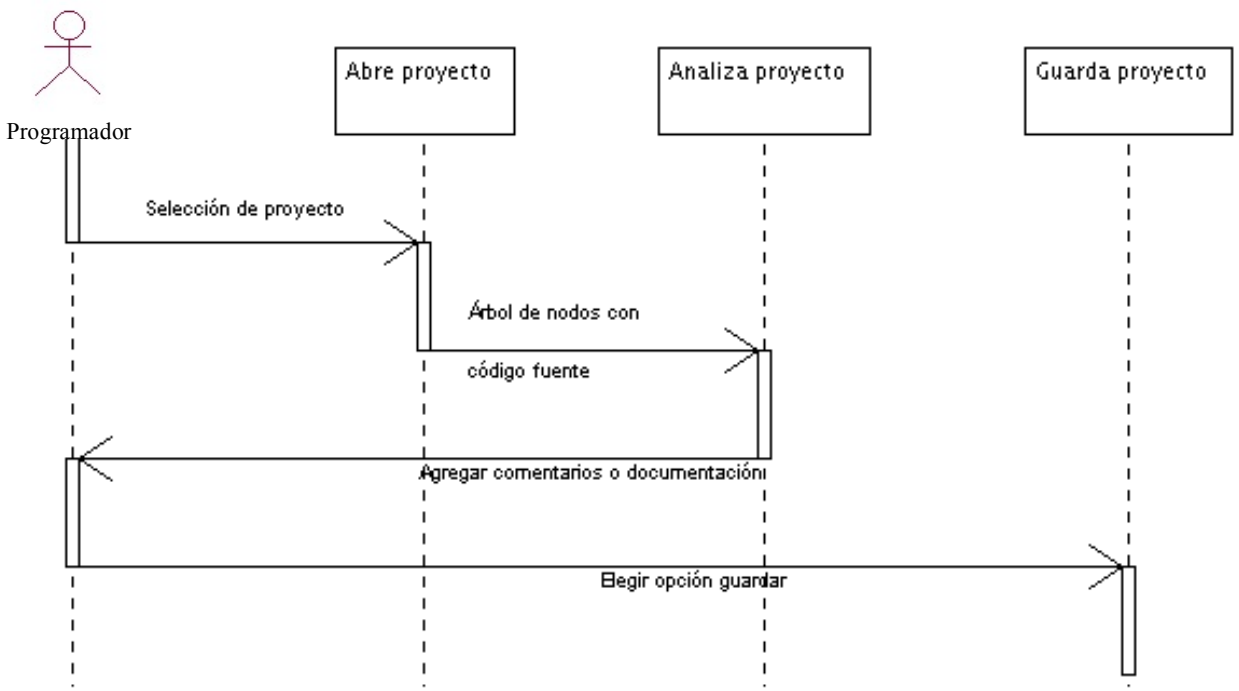
Caso de uso:	Guardar proyecto
Nombre:	Guardar proyecto
Actores:	Programador
Propósito:	Guardar un proyecto Visual Basic, versión 5 o 6.
Tipo:	Primario
Descripción:	El programador elige guardar un proyecto de Visual Basic (versión 5 o 6).
Referencias:	8, 9

Curso de Eventos: Guardar proyecto

No.	Acción del programador	No.	Respuesta del sistema
1	Elegir guardar proyecto	2	El sistema analiza la versión del proyecto, y extrae de una plantilla temporal los cambios
3	Confirmar guardar proyecto	4	Una vez que se ha confirmado guardar el proyecto, los cambios efectuados de la plantilla se agregan al proyecto

2.3. Diagrama de secuencias

La forma en que interactúan los programadores con ENDER, se muestra en la siguiente imagen.



CAPÍTULO 3

DESARROLLO, INSTALACIÓN Y PRUEBAS DEL SISTEMA A CONSTRUIR

3.1. Codificación de ENDER en Visual Basic

Una vez que se tiene terminada la fase de Análisis y Diseño y teniendo la ayuda de la herramienta para programar Visual Basic, es el momento de iniciar el desarrollo del sistema ENDER.

Para comenzar a codificar dicha herramienta hay que tomar en cuenta que se programará en Visual Basic versión 6, dado que dicha herramienta puede funcionar para las versiones 5 y 6 del lenguaje de programación.

Inicialmente se generaron los formularios, debido a que ésta es la interfaz gráfica que se comunica directamente con el usuario, por lo tanto se tuvieron que generar tres estos. El primer formulario se muestra en la figura 1, y éste contiene la información de bienvenida a la aplicación y una breve descripción de lo que realiza dicho sistema.

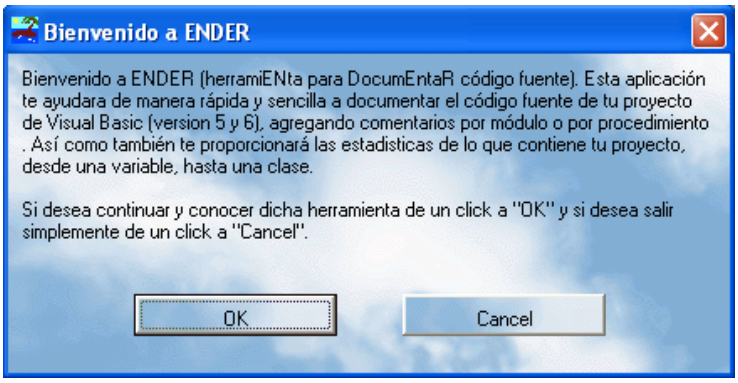


Figura 1. Formulario de bienvenida a ENDER

El segundo formulario que se muestra en la figura 2, contiene las acciones que podrá realizar un usuario, ya sea para analizar el proyecto o simplemente para documentar su código fuente.

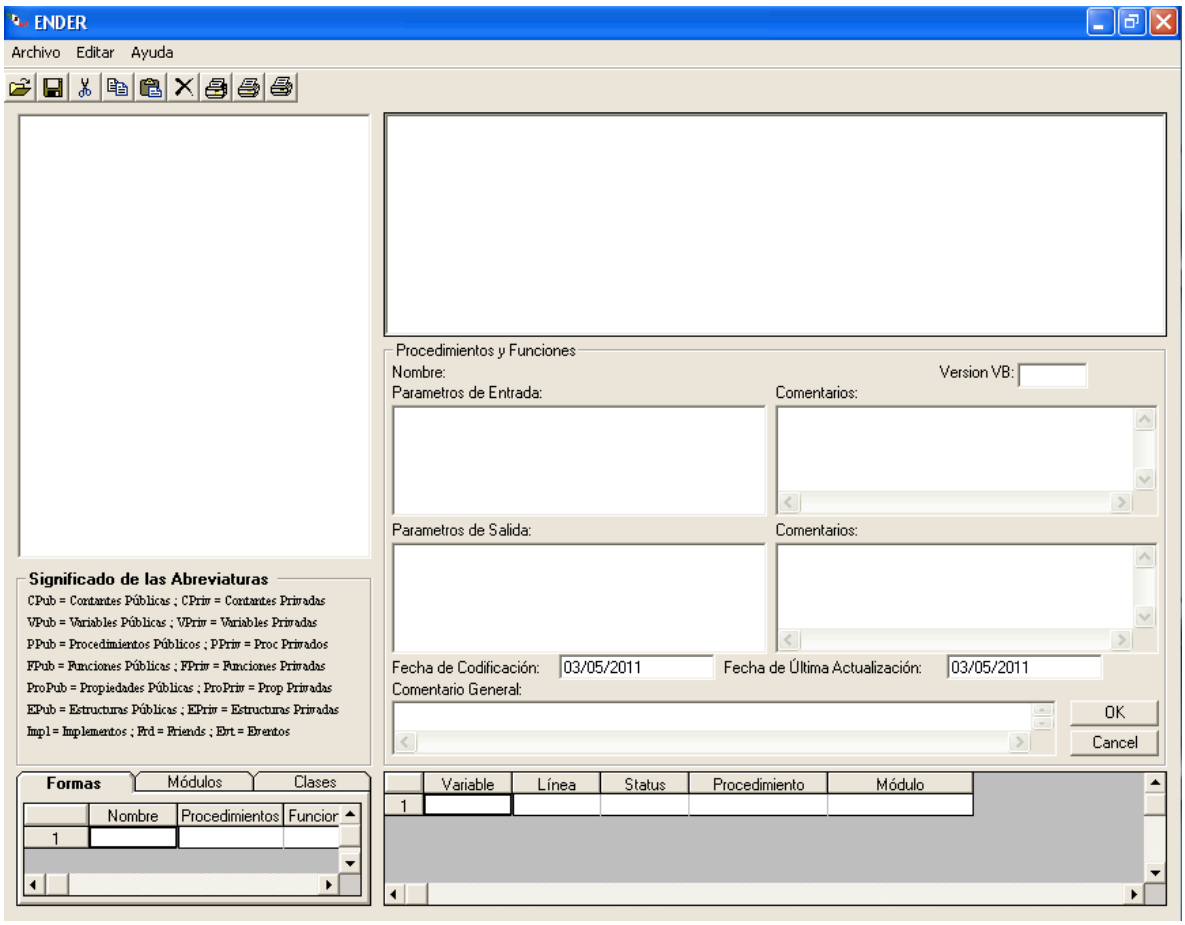


Figura 2. Formulario que documenta y analiza código fuente

Por último el formulario de ayuda como se muestra en la figura 3, el cual contiene información de la aplicación y la versión de la misma.

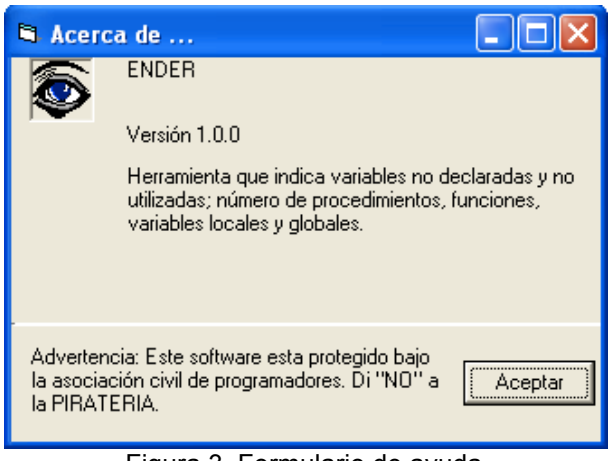


Figura 3. Formulario de ayuda

Una vez que se ha generado la interfaz gráfica (formularios), es el momento de comenzar con la codificación de los requerimientos del sistema a construir.

El código fuente de ENDER se encuentra dividido en dos módulos, cabe aclarar que no existe codificación en los formularios que se han creado.

El primer módulo contiene las acciones que se derivan del formulario que se presenta cuando se ejecuta ENDER, se trata de la bienvenida a la herramienta, contiene el código de los dos botones, uno de los cuales es para seguir utilizando dicho sistema y otro para cancelar y cerrar la aplicación, tal como se muestra en la figura 1; También contiene toda la funcionalidad del formulario donde el usuario podrá incorporar documentación a su proyecto, como se ve en la figura 2. Éste módulo está compuesto por procedimientos como el abrir un proyecto, generar un árbol de nodos, agregar documentación y guardar proyecto. Finalmente contiene el código del formulario acerca de, que se observa en la figura 3.

El segundo módulo, busca las variables y constantes del proyecto que se está analizando y verifica si se han utilizado o sólo fueron declaradas y no utilizadas. También cuantos procedimientos, funciones y variables, contienen los formularios, módulos y clases. Éste módulo es el encargado de revisar las palabras reservadas. Finalmente contiene los avisos del formulario Acerca de, que se muestra en la figura 3.

3.2. Instalación de ENDER para documentar código fuente

Una de las principales acciones al terminar de codificar dicha aplicación, es la de realizar la documentación apropiada y la instalación de la misma, así se concluirá la etapa de análisis, diseño y desarrollo.

Para comenzar con el proceso de instalación, es necesario tener en cuenta los siguientes requerimientos, para el buen funcionamiento de la herramienta. El procesador debe ser superior a Pentium IV (como mínimo y contar con una arquitectura a 32 bits), 256 Mb en RAM, Windows XP (o superior), 5MB de espacio en D.D. y unidad de CD-ROM. Una vez cumplidos con estos requerimientos ahora se inserta el CD-ROM en la unidad y se busca el archivo "setup.exe" que viene en el disco extraíble. Ya que se ejecuto el archivo, ahora hay que indicarle la dirección en donde se quiera instalar. Una vez concluido este paso se instala y entonces se procede a reiniciar el Sistema Operativo.

Una vez ya instalada la aplicación se puede utilizar dicho sistema. Podrán ser analizados y procesados dichos proyectos del lenguaje Visual Basic, siempre y cuando no esté en uso el proyecto, ya que si está en uso no podrán hacerse los cambios en los módulos afectados.

3.2.1. Funcionamiento del sistema

El funcionamiento del sistema está incluido en el manual de uso y manejo que se encuentra en el APÉNDICE K.

3.3. Fase de pruebas

En este tema hay que señalar que existieron dos tipos de pruebas. En la primera etapa de pruebas, se enlistan los errores que se presentaron en la tabla 1.

Número de prueba	Nombre de la prueba	Resultado de la prueba	Corrección de ENDER
1	Correr proyecto Visual Basic versión 4	No funcionó con ésta versión de proyecto, arrojo un error y salió del proyecto repentinamente	Se realizó la corrección de este error, lanzando un mensaje al usuario en donde se le mencione que no es posible hacer pruebas en versiones que no sean 5 y 6 de Visual Basic
2	Correr proyecto Visual Basic versión 5	Si funcionó y cuando comienza a crearse el árbol de las formas, módulos y clases, arrojó un error de desbordamiento de código	Este error se presentó debido a que no existía un ciclo en el código que encontrara la última línea, por lo tanto, al terminar de leer el código fuente del proyecto abierto, volvía a leerlo de nuevo, hasta que ocurría un desbordamiento de código. La solución fue poner un ciclo de fin de línea del código fuente del proyecto analizado

Número de prueba	Nombre de la prueba	Resultado de la prueba	Corrección de ENDER
3	Correr proyecto Visual Basic versión 6	Si funcionó y termino de ordenar el árbol de formas, módulos y clases, el error surgió a partir de dar un clic en cada procedimiento, no mostraba el código fuente	No mostraba el código fuente debido a que no estaba abriendo el archivo correcto, apuntaba a otro directorio cuando se requería leer algún archivo. La solución fue leer el archivo de cada módulo al que le corresponde apuntando al directorio correcto
4	Correr proyecto Visual Basic versión 6	Funcionó correctamente el análisis del proyecto, y arrojo un error al intentar guardar el proyecto con documentación anexada	Estaba guardando el archivo en una nueva carpeta, y al intentar escribir en el archivo, no encontraba un archivo existente. La solución fue guardar en el archivo original la documentación anexada

Tabla 1. Fase de pruebas 1

Al término de esta fase de pruebas, se hicieron las correcciones pertinentes como se muestran en la tabla 1.

La segunda prueba fue realizada en el centro de computo de la FES Aragón, esto con el fin de que otros usuarios distintos tuvieran acceso a efectuar pruebas en sus proyectos, en este caso no hubo que hacer cambios. La prueba fue realizada con éxito.

3.4. Implementación y capacitación

La etapa de implementación incluye un documento que contiene el análisis, desarrollo, instalación y capacitación, misma que se le entrega al usuario final para que se tenga control sobre dicho sistema.

Con el enfoque antes mencionado se realizó la implementación de ENDER al centro de cómputo a la vista del encargado del departamento de sistemas, también se entrego un manual de instalación (APÉNDICE J) y el de uso y manejo (APÉNDICE K) para la capacitación que tendría lugar un día después de la instalación.

Después de haber realizado la capacitación, se dejo abierta la línea de soporte técnico al usuario para cualquier duda o comentario que haya surgido.

Cabe señalar que el usuario no ha tenido dudas acerca del uso y manejo de ENDER. También se ha preguntado al encargado de sistemas si han tenido alguna complicación con el desarrollo de software, y las respuestas han sido negativas. Todo parece indicar que ha funcionado como se esperaba.

3.5. Perspectivas

En el presente proyecto, se han incorporado algunas de las etapas principales que se acercan a la documentación de proyectos en Visual Basic (versión 5 y

6), pero hay que señalar que no se ha concluido en su totalidad, con este proyecto se quiere llegar a que sea una herramienta que se utilice a todos los niveles de los programadores en general, y una de las metas seria completar mas este proyecto, esto se puede lograr haciendo un complemento de lo ya expuesto en este tema, esto es, hacer una continuación del análisis de proyectos pero que sean de otros lenguajes de programación como son: Lenguaje C, C++, Java, FoxPro, C#, Visual Basic .Net, PHP, Pascal y Cobol.

C O N C L U S I O N E S

A medida que han ido avanzando las técnicas y herramientas utilizadas para el desarrollo de sistemas, nuevas ideas han inundado el mundo de la programación o desarrollo, muestra de ello es la construcción de esta herramienta como apoyo a la documentación, que con la ayuda de Visual Basic como plataforma que cuenta con herramientas de un lenguaje de programación de alto nivel, puede incluir programa o sistemas complejos para crear una estadística y documentar lo requerido.

Durante el desarrollo de este proyecto, se han conocido algunas herramientas que forman parte del desarrollo de documentación y diferentes aplicaciones que se necesitan para ello, que van desde el análisis y modelado hasta el desarrollo y puesta en marcha, pasando por el diseño de interfaz gráfica y los distintos casos de uso.

Este caso práctico desarrollado para el Centro de Cómputo de la Facultad de Estudios Superiores plantel Aragón, se concluye que el análisis, modelado y desarrollo han dado como resultado la optimización de un proyecto desarrollado en Visual Basic (versión 5 o 6) que contiene las características solicitadas al inicio del proyecto.

Por otro lado se recomienda las siguientes mejoras del sistema para irlo perfeccionando. Versión 2.0.

- Analizar proyectos creados en Visual Basic .Net, C, C++ y Java.
- En la sección “versión VB”, se debería agregar automáticamente el lenguaje de programación del proyecto que se este analizando.
- Poner en una nueva pestaña los significados de las siglas, ya que va a crecer dicha lista por los diferentes lenguajes de programación que se añadirán.

- Crear una nueva columna dentro de las estadísticas que contenga una sugerencia, del que hacer con la variable declarada y no utilizada, o variable no declarada y utilizada.

No obstante, dado que el esquema general del presente proyecto solo pretende coadyuvar al usuario a documentar su código fuente, existe la posibilidad que al darle seguimiento en la versión 2.0., surjan inquietudes por parte del programador, para agregar más funcionalidades.

De manera personal quiero agregar que la documentación en general es una buena práctica que se debe aprender en la formación académica del alumno, en todas y cada una de las materias que lo amerite. Al final con ésta acción se irá fomentando esta práctica y en un futuro inmediato los códigos fuente de cualquier sistema y en cualquier lenguaje de programación, serán reusados y se les podrá aplicar reingeniería.

GLOSARIO

Actor: Se le llama actor a toda entidad externa al sistema que guarda una relación con éste y que le demanda una funcionalidad. Esto incluye a los operadores humanos pero también incluye a todos los sistemas externos, además de entidades abstractas, como el tiempo.

Análisis: Estudio, mediante técnicas informáticas, de los límites, características y posibles soluciones de un problema al que se aplica un tratamiento por ordenador.

Arreglo: Un arreglo puede definirse como un grupo o una colección finita, homogénea y ordenada de elementos. Los arreglos pueden ser de los siguientes tipos: de una, dos, tres o más dimensiones.

Atributo: Es una especificación que define una propiedad de un Objeto, elemento o archivo. También puede referirse o establecer el valor específico para una instancia determinada de los mismos.

Ciclo de vida: Describe el desarrollo de software, desde la fase inicial hasta la fase final.

Clase: Las clases son declaraciones de objetos, también se podrían definir como abstracciones de objetos. Esto quiere decir que la definición de un objeto es la clase. Cuando programamos un objeto y definimos sus características y funcionalidades en realidad lo que estamos haciendo es programar una clase.

Código Fuente: Es un conjunto de líneas de texto que son las instrucciones que debe seguir la computadora para ejecutar dicho programa. Por tanto, en el código fuente de un programa está descrito por completo su funcionamiento.

Comentarios generales: Están destinados a integrar información adicional dentro del código fuente. En el caso del comentario general, es útil para control de versiones, y detalles muy generales.

Comentarios privados: Están destinados a integrar información adicional dentro del código fuente. En el caso del comentario privado, es útil para la mejora a futuro de procedimientos o funciones en específico.

Desarrollo: Puede involucrar numerosas y variadas tareas, desde lo administrativo, pasando por lo técnico y hasta la gestión y el gerenciamiento. Pero casi rigurosamente siempre se cumplen ciertas etapas mínimas: Análisis, Diseño, Codificación, Pruebas, Instalación y Mantenimiento.

Diagrama de secuencias: Muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada caso de uso.

Diagramas de casos de uso: Los Casos de Uso son qué hace el sistema desde el punto de vista del usuario. Es decir, describen un uso del sistema y cómo este interactúa con el usuario.

Diseño: Se define como el proceso previo de configuración mental, "pre-figuración", en la búsqueda de una solución en cualquier campo.

Documentación: Son los elementos constitutivos que integran y forman parte de un sistema de información, como son todos los aspectos que produjeron el análisis, diseño, programación, evaluación y pruebas, implantación y operación y mantenimiento, incluyendo los manuales de operación y del usuario.

Evolución de la informática: Es el crecimiento y desarrollo integral de la tecnología de la información y de las comunicaciones, en las que a través del tiempo se van manifestando sus estados del arte.

Forma: Es una ventana de Windows la cual usaremos para interactuar con el usuario, ya que en dicha ventana o formulario, estarán los controles y demás objetos gráficos que mostraremos al usuario de nuestra aplicación. Los formularios también son llamados "formas" o Forms en su nombre en inglés.

Función: Realizan una tarea, al igual que un procedimiento, pero siempre suelen devolver un valor, resultado del código que se ha ejecutado en su interior. A las funciones no se les puede asignar valores, a diferencia de las Propiedades.

Herramienta de desarrollo: Son aquellos programas o aplicaciones que tengan cierta importancia en el desarrollo de un programa (programación). Pueden ser de importancia vital (como un ensamblador, un compilador o un editor) o de importancia secundaria, como una IDE (Integrated Development Environment - Entorno de Desarrollo Integrado).

Interfaz: Hace referencia al conjunto de métodos para lograr interactividad entre un usuario y la computadora.

Lenguaje de programación: es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo.

Llena Grid: Cuadrícula para presentar datos en forma de tabla.

Mantenimiento productivo: Esta etapa se caracteriza por la progresiva mentalización por la calidad y el consiguiente desarrollo de técnicas para el control y aseguramiento de la calidad. En esta etapa, se produce un gran desarrollo tecnológico en los medios de producción, impulsado por la necesidad

de diseñar equipos que puedan producir bienes de la calidad exigida por el mercado.

Mapa de conceptos: Es una Técnica de Estudio dentro del constructivismo que produce aprendizajes significativos al relacionar los conceptos. Se caracteriza por su simplificación, jerarquización e impacto visual y convencional.

Modelamiento de clases: A través de ella podemos diseñar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

Modelar sistemas: Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

Modelo conceptual: Modelo visual de un sistema que ilustra las interconexiones de los componentes del modelo.

Módulo: Al igual que las clases, son "espacios" en los cuales se incluyen declaraciones de variables, procedimientos, funciones, etc. Pero a diferencia de las clases, el código contenido en un módulo siempre está disponible de forma directa, sin necesidad de crear una "instancia" de dicho módulo.

Operación: Para el desarrollo y crecimiento que las empresas u organizaciones pretenden lograr, es necesario que cuente con herramientas de consulta que integren la información operativa a través de manuales funcionales.

Palabras reservadas: Es una palabra que tiene un significado gramatical especial para un lenguaje de programación y no puede ser utilizada como un identificador en ese lenguaje.

Plantilla: Es el procedimiento que permite guiar, portar o construir un diseño o esquema predefinido.

Plataforma (informática): Es determinado software y/o hardware con el cual una aplicación es compatible y permite ejecutarla.

Procedimiento: Es una secuencia de pasos repetible y determinista; es decir, una en que siempre se irán obteniendo los mismos conjuntos de valores de salida, para los mismos conjuntos de valores de entrada.

Programación estructurada: Es una técnica para escribir programas de manera clara. Para ello se utilizan únicamente tres estructuras: secuencia, selección e iteración; siendo innecesario el uso de la instrucción o instrucciones de transferencia incondicional (GOTO, EXIT FUNCTION, EXIT SUB o múltiples RETURN).

Programación orientada a objetos: Es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento.

Proyecto informático: Es un sistema de cursos de acción simultáneos y/o secuenciales que incluye personas, equipamientos de hardware, software y comunicaciones, enfocados en obtener uno o más resultados deseables sobre un sistema de información.

Reutilización de código fuente: Es la capacidad de trasladar características de un objeto a otro, lo que se logra con alguna forma de herencia. Existen principalmente dos formas de reutilización del código: binario y fuente.

UML: Es un conjunto de herramientas, que permite modelar (analizar y diseñar) sistemas orientados a objetos.

B I B L I O G R A F I A

- Diccionario Porrúa de la Lengua Española, 2009.

- Ingeniería del software un enfoque práctico, Roger S. Pressman. Ed McGraw-Hill/Interamericana de España S.A. España 1993.

- The UML specification documents, Booch, G., Jacobson, I. Y Rumbaugh, J. Ed Santa Clara CA, 1997.

- Aprendiendo UML en 24 horas, Joseph Schmuller, Ed Prentice Hall, México 2000.

- Fundamentos de programación, Ernesto Peñalosa Romero, 3ª Edición, UNAM Campus Aragón, México 2001.

- Programación orientada a objetos con C++, Francisco Javier Cevallos Sierra, Ed AlfaOmega Grupo Editor, México 1998.

- Guide to good programming, Meek, B., and P. Heath (eds.), Ed Halsted Press (Wiley),1980.

- Diseño y Gestión de Sistemas de Base de Datos, Angel Lucas Gómez, Ed Paraninfo, España 1993.

- Encyclopedia of Computer Science, Third Edition, Anthony Ralston, Edwin D. Reilly, Ed IEEE PRESS, USA 1993.

- Dictionary of Computer Terms, Third Edition, Douglas Downing, Michael Covington, USA 1992.

- Microsoft ® Encarta ® Biblioteca de Consulta 2002. © 1993 – 2001 Microsoft Corporation.

- Introducción a las Computadoras, Robledo Sosa C. Ed. Tlamatini S.A. México 1986.

- UML y Patrones Introducción al análisis y diseño orientado a objetos, Craig Larman, Ed Prentice Hall, México 1999.

- Discrete structures and autómata theory, Rakesh Dube, Adesh Pandey, Ritu Gupta, Ed. Alpha Science International Ltd, India 2006.

- Introducción a la teoría de autómatas, lenguajes y computación, John E. Hopcroft, Jeffre D. Ullman, Ed. CECSA, México 1993.

- Autómatas y lenguajes, un enfoque de diseño, Ramón Brena, Ed. ITESM, México 2003.

- Teoría de autómatas, lenguajes formales y gramática, David Castro Esteban, Ed. Universidad de Alcalá, España 2004.

- Teoría de autómatas y lenguajes formales, Dean Kelley, Ed. Prentice Hall, España 2001.

- Teoría de autómatas y lenguajes formales, Navarrete Sánchez Isabel, Martínez Béjar Rodrigo, Ed. Diego Marin Librero Editor, S.L. España 2003.

- Teoría de la computación, Elisa Viso G., Facultad de Ciencias UNAM, México 2002.

REFERENCIAS DE INTERNET

http://www.rae.es

http://es.wikipedia.org

http://es.wikibooks.org/wiki/Fundamentos_de_programaci%C3%B3n/Herramientas_de_desarrollo

http://www.diccionarios.com

http://andresorellana.tripod.com/tema2.htm (año de consulta 2009)

http://lucas.simplenet.com/trabajos/objetos/objetos.html (año de consulta 2009)

http://www.acatlan.unam.mx/Revistas/11/Tecnologia/ingsof.txt (en la actualidad el enlace no funciona, año de consulta 2009)

http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html (año de consulta 2009)

http://orbita.starmedia.com/~boymerang/queleng.htm (en la actualidad la referencia electrónica no funciona)

http://www.alegsa.com.ar/Dic/plataforma.php (año de consulta 2009)

http://es.kioskea.net/contents/genie-logiciel/cycle-de-vie.php3 (año de consulta 2009)

http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c1/c1.htm (año de consulta 2009)

http://www.ingenierosoftware.com/analisisydiseno/uml.php (año de consulta 2009)

http://www.gestialba.com/public/utilidadescast05.htm (año de consulta 2009)

http://www.desarrolloweb.com/articulos/499.php (año de consulta 2009)

APÉNDICES

APÉNDICE A

CICLO DE VIDA DE UN PROYECTO[i]

Todos los sistemas informáticos comprenden el ciclo de vida de un proyecto. Cada proyecto atraviesa por algún tipo de análisis, diseño e implantación, aunque no se haga exactamente como se muestra por ejemplo en el siguiente diagrama.

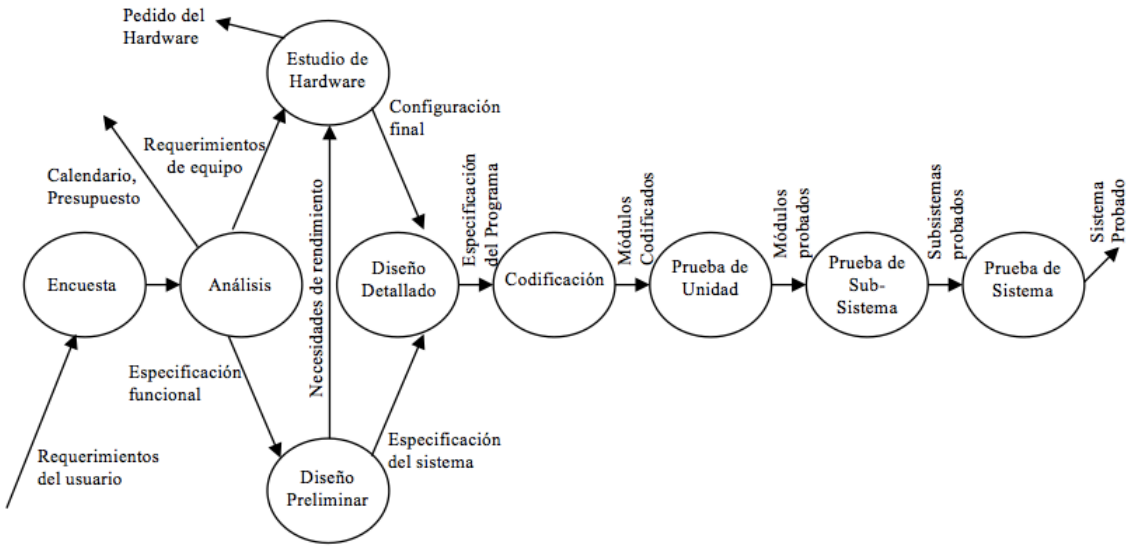


Figura 1. Diagrama de ciclo de vida de un proyecto

El ciclo de vida de un proyecto utilizado, puede diferir de la Figura 1 en una o en todas las formas siguientes:

- Las fases de encuesta y análisis pudieran juntarse en una sola.

- Puede no haber fase de estudio de hardware si se cree que cualquier sistema nuevo pudiera instalarse con las computadoras existentes sin causar mayor problema operacional. (Sin embargo, es necesario para sistemas de software muy grande)

- La fase de diseño preliminar y el diseño de detalles pudieran juntarse en una sola llamada simplemente de diseño. (Sin embargo, es importante aclarar que en algunas empresas lo establecen en sus lineamientos de construcción)

- Diversas fases de pruebas pueden juntarse en una sola; de hecho, podrían incluirse con la codificación, siempre y cuando los sistemas sean pequeños.

El uso de la implantación ascendente es una de las grandes debilidades del ciclo de vida de los proyectos clásicos. Tal como se muestra en la Figura 1, se espera que los programadores lleven a cabo primero sus pruebas modulares, luego las pruebas del subsistema, y finalmente las pruebas del sistema mismo.

Muchas organizaciones que desarrollan sistemas únicos, el enfoque ascendente presenta un gran número de dificultades serias:

- Nada esta hecho hasta que todo este terminado.

- Las fallas más triviales se encuentran al comienzo del periodo de prueba y las más graves al final.

- La eliminación de fallas suele ser extremadamente difícil durante las últimas etapas de prueba del sistema.

- La necesidad de prueba con la computadora aumenta exponencialmente durante las etapas finales de prueba.

La segunda debilidad más importante del ciclo de vida de un proyecto clásico es su insistencia en que las fases sucedan secuencialmente. Querer esto es una tendencia natural humana: deseamos decir que hemos terminado la fase de análisis del sistema y que nunca tendremos que volver a preocuparnos por ella. El único problema del progreso ordenado es que no es nada realista. Por

ejemplo, durante el periodo que transcurre para desarrollar el sistema pueden cambiar ciertos aspectos del ambiente del usuario (la economía, la competencia, los reglamentos gubernamentales que afectan a las actividades del usuario).

DOCUMENTACION[ii]

Para artículos de materias relacionadas ver: Aplicaciones administrativas, tabla de decisión, copia de cartas, especificación de programas, ingeniería de software patrones y programación estructurada.

La documentación es una parte vital para desarrollar y usar un sistema basado en computo y es una parte integral de lo que ahora es llamado ingeniería de software. En algunas organizaciones comerciales 20% o hasta más de total de esfuerzos en el desarrollo va dentro de la documentación del nuevo sistema, grabando como es que trabaja y como es desarrollado. La documentación de un proyecto se divide en dos categorías: desarrollo de documentación y control de documentación. El desarrollo de documentación se graba como un sistema basado en computo, está estructurado y que es lo que esta supuesto a hacer, proporciona el historial de información acerca de cual es el diseño encontrado. Control de documentación, por otro lado realiza funciones administrativas y de servidor. Este graba los recursos usados en el desarrollo e implementación del sistema, e incluye aquellos documentos como: planes de proyecto, horarios, detalles de recurso de colocación y reportes de progreso.

FUNCIONES DE LA DOCUMENTACION, la documentación tiene 4 funciones principales:

1. Comunicación de ínter tareas / interfase
2. Referencia histórica para corrección y modificación
3. Control de calidad y cantidad

4. Referencia de instrucción

La relativa importancia de cada uno de estos depende de muchos factores, por ejemplo, uno de los más importantes es la intención y el tipo de proyecto, puede ser un sistema comercial a gran-escala o un programa científico de problemas-resueltos usado por uno o 2 técnicos en una cantidad limitada de datos. Durante cada categoría, hay variaciones en el tamaño del proyecto, complejidad del problema, organización del personal y el tiempo a escala para su uso y desarrollo. Cada función de la documentación es descrita más adelante.

Comunicación de ínter tareas / interfase: esta operación graba que ha sido hecho en cada etapa del proyecto, por lo que estas instrucciones pueden ser el resultado de la siguiente fase del trabajo, o puede que toda la gente involucrada en el proyecto este de acuerdo en que ha sido hecho antes de los procedimientos de trabajo para dar el siguiente paso. La cantidad de tiempo y esfuerzo debe ser dedicada a la documentación, por esta razón es una función del indicador del sistema y el número de personas involucradas.

En el desarrollo de un mejor sistema comercial el cual requiere procedimientos como: facturación, control de inventario, nómina de sueldos o control de producción, muchas personas estarán involucradas. En un sistema de control de producción, por ejemplo: las funciones de negocios involucradas pueden incluir, entre otros:

1. Venta de proyectos (unido a las cuentas de ventas).
2. Partes de la explosión y producción recibido/ en la red (unido al diseño de ingeniería).
3. Plantear recursos de colocación y horarios.
4. Materiales de ordenación/ colocación y herramientas.
5. Monitorear el progreso del trabajo.
6. Reportes de desechos y dividendos.

7. Costos de trabajo (unidos a todos los sistemas).

Muchas de estas funciones están interrelacionadas. Algunas 20 o 30 funciones de trabajo separadas o unidades de organización pueden estar involucradas en el desarrollo, implementación y manejo del sistema de computo. En adición a las funciones de trabajo como las descritas, diferentes niveles para el personal involucraran dirección ejecutiva o mayor, dirección en línea, supervisores y operadores. Similarmente, un número de funciones de trabajo serán ejecutadas para el personal en el procesamiento de datos o en el departamento de servicios de dirección, por ejemplo:

1. Analistas de negocios, consultores de negocios internos quienes aconsejan la dirección en los métodos para negocios e identifican áreas para el mejoramiento.

2. Analistas de sistemas (q.v.) son quienes investigan, analizan y especifican un nuevo sistema.

3. Diseñadores de sistema, son quienes diseñan el sistema nuevo (computador y manuales de procedimiento) con detalles.

4. Programadores, diseñan, codifican y prueban los programas del computador para el sistema.

5. Operadores, son los responsables por e l día a día del manejo del sistema.

Debe haber también soporte general o servicio de personal durante el procesamiento de datos como: programadores de mantenimiento, personal para soporte de software, planificadores a futuro, y analistas estándar. En una instalación pequeña, muchas de las funciones de trabajo antes mencionadas

pueden ser ejecutadas por una persona o un grupo pequeño. En una instalación grande, cada función de trabajo puede ser ejecutada por un grupo especial.

Mantener a la gente informada pasando información e ideas para aprobación y así mismo dar instrucciones involucra una compleja comunicación de redes, en la cual la documentación formal juega un papel vital.

Un fracaso de documentación por medio de información pobre (o la carencia de la misma) puede resultar verdaderamente muy caro. La documentación también ayudará continuamente a la seguridad del proyecto ya que pueden ocurrir cambios en el personal.

El uso de la documentación para la comunicación de ínter tareas / interfase es igualmente importante en grandes proyectos técnicos o científicos. Donde el desarrollo de un programa o grupo de programas puede ser hecho por sólo un número limitado de técnicos quienes son a menudo problemas de ambos programadores, de solución y proposición, la importancia de la documentación durante el proyecto disminuye. Sin embargo, la documentación de lo que ha sido hecho y como trabajan los programas será importante para las referencias históricas o de instrucción como están descritas más adelante.

REFERENCIA HISTORICA

La función de referencia es relevante para ambos trabajos, científico y comercial. Esta es la documentación de cómo el sistema trabaja facilita el cambio después de que es utilizado. Todos los sistemas son materia a cambiar. El mantenimiento de los sistemas de negocios y programas será requerido por causa de la naturaleza del negocio y sus cambios de métodos, o porque la organización es reestructurada, nuevos tipos de productos son desarrollados, cambian los requerimientos de reportes de dirección, etc.

En el trabajo científico, los programas deben ser alterados porque la naturaleza del problema para ser resuelto cambia, posiblemente como resultado de nuevas investigaciones. Un sistema debe ser cambiado por causa del nuevo software o hardware. Puede ser deseable cambiar los métodos de procesamiento para que las nuevas técnicas se vuelvan disponibles. La razón para el cambio puede mentir en conjunto fuera de la organización, como es el caso con los requerimientos y cambios legales.

Un sistema puede mantenerse eficiente sólo si la operación existente de todos los procedimientos y programas es claramente conocida y entendida. La documentación del sistema provee de este conocimiento. Por ejemplo un programa escrito hace un año es cambiado hoy, el programa consiste en 2,000 instrucciones, con ramas y recovecos anidados. El programador que originalmente lo escribió no esta disponible. Las modificaciones requieren la lógica del programa comprendido, el nuevo programador debe asegurarse que los errores no sean introducidos por no ver más allá del impacto en algunos de los cambios.

La documentación de un sistema puede ser también revisada con propósitos de ejecución. Muchas instalaciones desarrollan Standard de ejecución basados en grabaciones de tiempo y presupuestos de recursos usados en el desarrollo del sistema, así como en tipos de sistemas, alcance y complejidad. El control de la documentación es usado para los detalles en los recursos y el desarrollo de la documentación para la descripción del sistema. Para la captura formal de los detalles de todos los proyectos, los recursos estimados para los proyectos futuros pueden ser mejorados.

CONTROL DE CALIDAD / CANTIDAD

Como un sistema desarrolla varios elementos de la documentación hasta que cada paso es terminado, la dirección puede usar esta documentación para evaluar el progreso del proyecto y la ejecución individual.

REFERENCIA DE INSTRUCCIÓN

El desarrollo de la documentación puede ser revisado durante y después del desarrollo para muchos propósitos en general. Por ejemplo, la documentación habilitará a estudiar el sistema por técnicos experimentados. Este es particularmente importante en la referencia de instrucción para generalizar sistemas o propósitos- generales de software. Otro beneficio de la documentación fuera de partido puede evaluar el sistema y su método de operación para determinar si el paquete es apropiado para usarlo en otro ambiente. En este caso debe darse información suficiente para habilitar al usuario a aplicar el software a otros requerimientos y problemas.

La referencia de instrucción ahora incluye toda la literatura que da un proveedor de software, como la referencia de manuales para todos los lenguajes, utilidades, sistemas de operación subrutinas y aplicación de paquetes, también incluye la documentación y facilidades de librería en una gran organización que produce su propio software.

TIPOS DE DOCUMENTACIÓN

En el desarrollo de un sistema, si es un sistema comercial a gran- escala o un grupo de programas científicos para analizar datos, ciertas categorías de datos deben ser consideradas. Estas son:

1. Documentación analítica.
2. Documentación de sistema
3. Documentación de programa
4. Documentación de operaciones
5. Ayuda usuario / dirección

Cada una de estas categorías es descrita más adelante, a través de los diferentes factores que tienen influencia en la forma de la documentación en cualquier organización particular.

DOCUMENTACION ANALITICA

Consiste en todos los reportes y grabaciones producidas cuando un proyecto es iniciado. Para todos los proyectos excepto aquellos que requieren un sencillo, primer- tiempo, un programa problema-solución, algunas formas breves para iniciar son requeridas. En muchas organizaciones los técnicos que diseñan, programan y prueban un sistema están agrupados en un computador o departamento de proceso de datos y los usuarios que trabajan en el departamento de proceso de datos, deben definir la naturaleza y los objetivos del proyecto. En algunos ambientes técnicos y científicos, el usuario es capaz de especificar en términos exactos que es requerido para el camino de procesar salidas. Generalmente, para cualquier tipo de proyecto, el inicio corto debe consistir en un requerimiento del usuario aterrizando el problema(i.e. que necesita el usuario llevar a cabo); un estudio viable que evalué las posibles soluciones (fuera de línea) y un plan de proyecto que estime el tiempo y los recursos requeridos para desarrollar e implementar el sistema. Fracasar para producir y estar de acuerdo con esas tres declaraciones en breve resultara después un esfuerzo desperdiciado en el proyecto. Son vitales cuando sea que un usuario es comisionado para el trabajo por los técnicos del computador, y debe proveerse antes de que el dinero sea destinado a impuestos por tiempo de consumo del sistema de diseño y programación.

DOCUMENTACION DE SISTEMAS

Encuadrar toda la información necesaria para definir el propósito del sistema base de computo a un nivel donde puede ser programado, probado e

implementado. El mayor documento es alguna forma de especificación del sistema, el cual actúa como una grabadora permanente de la estructura, este funciona y fluye trabajando, y los controles en el sistema. Este es el significado básico de la comunicación entre los sistemas de diseño, programación y funciones de usuario. Una muestra del contorno de la especificación de documentación para un mayor proyecto es mostrado en la fig. 1, si el proyecto resultará en el desarrollo de solamente uno o dos programas para uso restringido, entonces sólo el programa (procesado) habrá de ser producido.

DOCUMENTACION DE PROGRAMA

Incluye las grabaciones de los detalles lógicos y los códigos de los componentes de programas del sistema. Estas grabaciones, preparadas por el programador ayudan a la aceptación y desarrollo del programa. averías de ingeniería, mantenimiento, conversión de datos de software y cambios de programación.

La documentación de programas cubre dos especificas aplicaciones de programas y propósitos –generales o el desarrollo de software en casa. En adición a la documentación *como trabaja un programa,* las instrucciones para usar el programa deben ser escritas por el paquete de software.

DOCUMENTACION DE OPERACIONES

Estos procedimientos requieren para manejar el sistema de personal de operación. Este da la secuencia general para ejecutar los eventos de trabajo y define los procedimientos precisos para el control y seguridad de datos, preparación de los mismos, ejecución de programas, salidas dispersas y de operaciones.

AYUDA USUARIO/ DIRECCIÓN

Consiste en todo el material de instrucción y descripción necesario para el usuario en la participación del manejo operacional del sistema, incluye notas de los resultados de interpretación de salidas. Donde el paquete de software es producido, esta categoría incluye todo el material necesario para evaluar los programas y las instrucciones para su uso.

Toda instalación debería establecer documentación estándar (i.e. para la realización de ciertos documentos a cierto tiempo) eso define el contenido, formato y distribución de documentos. Muchos factores influyen en que documentos serán producidos, como cuando y por quien. Por ejemplo, la extensión del comité directivo es indicada en cuanto a la dirección de la instalación, es preparada para colocar tiempo y recursos, no sólo para desarrollar un sistema, también para su documentación. Otro factor de control puede ser *las características del proyecto*, que consiste en el número de proyectos, su extensión, complejidad y duración. Es crucial para cualquier lugar estándar de la estructura de la organización para ambos es como un todo, en el desarrollo y departamento de operaciones en particular, este en turno, es afectado por el ambiente técnico, las técnicas usadas de hardware/software, como el nivel de lenguaje de programación, la calidad de la documentación producida por el software, y el de propósitos-especiales de programas de documentación (flowcharters, etc.)

Del total de la documentación, seleccionamos un tipo para revisarlo en detalle nos enfocamos en este para que los limites de las tareas de programación puedan ser claramente definidos, y porque esta función en programación es similar en muchas organizaciones.

DOCUMENTACION DE PROGRAMAS

La figura 1 muestra el flujo de la documentación en diseño, código y prueba de cada programa respectivamente. El punto de comienzo es un programa de especificación. Típicamente, este es un informe de lo que el programa debe hacer, la tarea del programador es determinar como va a realizar el programa. Cuantos formatos de datos son predefinidos y cuanto se deja a la discreción del programador depende de la política de instalación y el proyecto. Otras salidas para la fase de programación incluyen librerías -la cual describe si el software esta disponible para el proyecto (también de las salidas suministradas o de una librería interna)- los programas estándar, los cuales dan las reglas y técnicas para la programación en esa instalación.

Las salidas (outputs) incluyen un manual de programación el cual describe programas en detalle (construcción, código y pruebas) instrucciones de uso (para un programa general), e instrucciones de operación de computo para el manejo de día a día. En muchos casos la tarea de documentar un programa es adicionar a la especificación inicial del programa para construir el manual del programa. Los distintos elementos de la documentación del programa son discutidos más adelante.

ESPECIFICACION DEL PROGRAMA

Este es un informe de los datos disponibles para procesar, las salidas requeridas, y los detalles necesarios del proceso. La especificación puede ser preparada por el ponente del problema, un especialista en sistemas analista/ diseñador, o el programador. Debe estar completo, exacto, y sin ambigüedades, cambiar a la especificación después de que el programa comience puede ser muy caro. La especificación usualmente contiene:

1. Entradas (Input)

2. Salidas (Output)
3. Ejecución de funciones mayores
4. El significado de la comunicación entre estos programas, los anteriores y los que siguen.
5. Reglas lógicas y decisiones para ser seguidas, incluyendo informes de cómo las entradas son alteradas, examinadas y usadas.
6. Criterio de edición y validación.
7. Acciones a tomar en errores o condiciones excepcionales.
8. Tablas especiales, formulas y algoritmos.

La descripción de las reglas de procesamiento (número 5 en la lista), pueden ser dados en forma narrativa, tabla-decisión y flowchart.

MANUAL DEL PROGRAMA

Del programa de especificación, el programador diseña, codifica y prueba el programa. El rendimiento de este ejercicio es el manual de programa. La forma lógica del diseño y el listado de la fuente del programa dependerá del tipo de aplicación y el software usado. Por ejemplo si un pre-procesador de la mesa de decisión de alto-nivel (q.v.) se usa el programa tabular junto con la descripción de los datos y la lista de la fuente, al final estará bastante completa sin la preparación del organigrama. Similarmente algunas instalaciones usan software para la documentación final; e.g. flowcharters que el producto detalló información por información en los organigramas del programa fuente.

Una de las ventajas de los lenguajes de alto nivel como el Cobol es que la fuente que se lista forma la parte mayor de la documentación final, el programador debe asegurar que el programa de la fuente no sólo sea lógicamente correcto y siga las reglas del lenguaje, sino también que el programa codificado sea ordenado y fácil de entender. Usando datos significativos empotrando nombres y comentarios en el programa, es posible hacer que una fuente programe casi

auto explicativamente. Un organigrama puede ser acostumbrado a un general "mapa de la ruta" al codificar y detallar en la inscripción de la fuente en este caso. Al alterar el programa operacional, muchos programadores se refieren directamente a la fuente que lista y después al organigrama sólo si el cambio requerido no es inmediatamente obvio en el listado.

Las listas de organigramas y referencias (producido por el recopilador u otro software) puede usarse para verificar que una alteración no ha perturbado otro código erróneamente. La ventaja de usar comentarios en la fuente de lista es que minimiza referencias a otros documentos. Los comentarios no se compilan como parte del programa, sino que meramente aparece en la inscripción de la fuente. Deben guardarse comentarios breves de la fuente del programa mientras que al mismo tiempo se describen y son significativos.

Note que la documentación final no sólo mostrará cómo el programa trabaja, sino también cómo fue probado (para el control de calidad y otra prueba después de los cambios que se hicieron), operando instrucciones para ejecutarlo, y cualquiera de los parámetros especiales serán dados al sistema operativo.

Aunque el manual del programa es el mayor rendimiento de la especificación del programa, el programador usará (y puede producir) otros tipos de documentación. Si el programa desarrollado es para el uso general, o estuviera dentro o fuera de la organización, las instrucciones adicionales necesarias para el usuario serán agregadas. Ellos deben permitirle al usuario que conteste las siguientes preguntas:

1. ¿Qué es lo que hace?
2. ¿Queremos usarlo?
3. ¿Podemos usarlo?
4. ¿Cómo lo usamos?

5. ¿Qué hacemos si cambia?
6. ¿Cuáles son sus limitaciones básicas?

Para el software internamente producido, un acercamiento es realizar un programa abstracto que pueda mantenerse en la biblioteca de la documentación central. Si, por primera inspección, el usuario siente que cumple las necesidades, la documentación detallada puede consultarse.

La forma en que la documentación se detalló depende del alcance y complejidad del software. Por ejemplo, la guía del usuario puede producirse para dar una descripción general del programa, las facilidades disponibles, el ambiente que requiere el hardware, ejemplos de los usos del software. La guía del usuario puede contener instrucciones para usar el software, o puede proporcionarse un manual de referencia de un programador también puede darse información detallada. La mayoría de los software se construyen para que el usuario programador proporcione parámetros para un trabajo particular. Los parámetros pueden ser tan simples como especificar una dirección o datos para ser procesados o ser tan comprensivos como una lista completa de requisitos del proceso. El manual de referencia del programador describirá la construcción del programa, junto con todos los parámetros requeridos (su formato, interdependencia, usos, operando instrucciones y condiciones de error y diagnóstico).

Los programas se preparan para el uso operacional que será ejecutado repetidamente, se asigna a la biblioteca del programa automatizado, normalmente guardado en disco magnético. La documentación para estos programas normalmente se sostiene en alguna forma de la biblioteca central de los archivos, junto con las copias de referencia de todas las descripciones del software.

DOCUMENTACION, MATENIMIENTO Y CONTROL

Una vez que un programa se ha llevado a cabo, la documentación debe retenerse para la referencia subsiguiente. Cuando el sistema cambia, la documentación se consultará y se alterará conjuntamente. Es vital revisar la documentación para que refleje completamente y con precisión en todo momento el funcionamiento del sistema. Si la documentación no se revisa, entonces el mantenimiento extenso será muy difícil. Después de cualquier enmendadura, todos los programas afectados tendrán que ser probados para demostrar que los cambios se han hecho correctamente y que no rompen o invalidan otro proceso.

Por consiguiente, es necesario asegurar que se usan los procedimientos de mando apropiados. Todos los cambios deben grabarse propiamente y todas las copias de la documentación se actualizan. Para minimizar el riesgo de copias fuera de fecha que se usan por equivocación, se deben versionar los ajustes de la documentación, para reducir el tiempo revisando archivos. Todas las copias deben cumplir los requisitos del parámetro actual para el Sistema Operativo, reglas del idioma, limitaciones y parámetros para los programas que utilizan mensajes de error producidos por todo el software. Una instalación robusta no sólo creará una biblioteca de archivos central, sino también fija a un bibliotecario con jornada completa para cubrir y enmendar la distribución. Esto a veces es manejado por un departamento de "apoyo del software”, que asegurarán que ambos programas y departamentos de operaciones estén informados de cambios en la disponibilidad del software y operación, así como la introducción de la nueva versión del Sistema Operativo.

Aunque algunas formas de practicar la documentación ha sido necesaria desde el advenimiento de la programación, hay una reciente tendencia hacia una metodología más disciplinada, donde el sistema y su documentación es producto de un derivado íntegro del sistema y de las actividades de diseño. Como el

método HIPO (Hierarchy plus Input-Process-Output). (Jerarquía más Entrada-Proceso-Rendimiento)

El impulso de esta nueva filosofía consta de un sistema de programación que contiene subsistemas, disminuye la dificultad obtenida en el diseño de arriba hacia abajo ya que los componentes se pueden manejar por separado. En 1976 Informática Inc. Usando la metodología de HIPO para el análisis, diseño de sistema y diseño del programa para su producto de control de piso de compra/80. Al final de la aplicación, se concluyó que sólo los funcionamientos y la necesidad de manuales de usuario se producían. La documentación de HIPO desarrollada continúa sirviendo a su propósito intencional, aunque HIPO (y otra estructura similar) la documentación es, desgraciadamente, no más fácil de mantener que las formas tradicionales, y retiene la utilidad más tiempo y más confiablemente que muchas otras formas de documentación. Esto es verdad porque la metodología tiende a dar fuerza a la similitud estructural entre la documentación del diseño y el propio programa. Si usted entiende lo que el sistema hace, es más fácil de encontrar la función y su implementación.

DOCUMENTACIÓN[iii]

La descripción escrita de un programa de computadora es conocida como documentación. La documentación entra en varias categorías:

1. La documentación interna, consiste en comentarios dentro del programa (vea comentario), es principalmente dirigida a futuros programadores que tengan que hacer correcciones u otras modificaciones.

2. Documentación en-línea, es la información que se despliega como las corridas del programa o puede llamársele ayuda. El usuario debe controlar la cantidad de información desplegada (más para los principiantes, y menos

conforme la experiencia del usuario aumenta). También las órdenes de AYUDA deben ser sensibles al contexto, en el cual se invocan; por ejemplo: tecleando AYUDA dentro de un editor deben llamar información sobre el editor, no el sistema operativo entero.

3 Tarjetas de referencia, contiene detalles fácilmente olvidados para la referencia rápida. Una tarjeta de la referencia asume que el usuario ya está familiarizado con los principios generales del programa.

4. Manuales de referencia, partiendo de instrucciones completas para el programa de una manera sistemática. La información relacionada debe agruparse, y debe proporcionarse un buen índice.

5. Guías didácticas, sirven como introducciones para los nuevos usuarios. Al contrario de un manual de referencia, una guía didáctica da la información en el orden en el que el usuario quiera aprenderlo; los artículos son agrupados a través de su importancia en lugar de por función o categoría lógica.

APÉNDICE B

PROPUESTA DE UN FORMATO PARA DOCUMENTAR CÓDIGO FUENTE

La figura 1 muestra una ventana desarrollada en Visual Basic la cual nos servirá para darle seguimiento a la documentación de procedimientos y funciones, proporcionar la versión de Visual Basic, la fecha de codificación y fecha de la actualización, al final aparece una caja de textos donde se puede agregar un comentario general.

A detalle, el primer cuadro de textos llamado “Versión VB” solo se podrá poner un 5 o un 6 ya que son las únicas versiones que puede leer esta herramienta. en el cuadro que dice “Parámetros de Entrada”, si el procedimiento los tiene aparecerán dentro de esa caja de textos, así como para el caso de “Parámetros de Salida”, delante de estos, está una caja de textos, en está se tendrá la libertad de asignarle comentarios al procedimiento o a la función, según sea el caso. En las fechas que aparecen, se pueden modificar, ya que no es necesario que sea la fecha del sistema. Para el caso del comentario general, este es utilizado para darle un panorama general al modulo que se esta analizando en ese momento.

Todos y cada uno de los comentarios que se han sugerido se introducirán en donde les corresponde, por ejemplo: El comentario general y las fechas, se agregarán al inicio del modulo, y los comentarios de los procedimientos y funciones, se añadirán donde les corresponde, al inicio de cada procedimiento u función.

Cabe señalar que este es un formato propuesto, no es un formato que siga ningún estándar o disciplina.

Figura 1. Propuesta de un formato para documentar código fuente.

APÉNDICE C

CRISIS DE SOFTWARE[iv]

La Ingeniería del Software es la disciplina relacionada con el desarrollo de productos de soporte lógico o software. Un producto de software es el conjunto completo de programas informáticos, procedimientos, documentación y datos especificados para su suministro a un cliente; el desarrollo se ocupa de todas las actividades técnicas y de gestión necesarias para crear el producto, y realizar el desarrollo eficazmente significa cumplir las necesidades del cliente ajustándose a unos límites de tiempo, coste y calidad.

El concepto de Ingeniería de Software surgió tras reuniones de trabajo emprendidas por la Organización del Tratado del Atlántico Norte (OTAN) en 1968 y 1969 para estudiar lo que entonces se describía como “Crisis del Software”. Había demasiados proyectos de desarrollo de soporte lógico que experimentaban fallos, los cuales se atribuían al rápido aumento en la escala y complejidad del Software en cuestión. Se reconoció que era necesario un planteamiento más sistemático en el desarrollo del Software, que debía basarse en principios de ingeniería ya establecidos.

El Software evoluciona a través de muchas versiones, a medida que se corrigen errores, se mejora el funcionamiento y se responde a las modificaciones que surgen en los requisitos. Cada nueva versión se crea a través de un proceso de desarrollo de Software. Típicamente, el proceso se divide en cuatro fases principales: (1) el análisis y especificación de requisitos, donde se establece qué debe de lograr el producto de Software; (2) el diseño, que determina cómo cumplirá el Software esos requisitos; (3) la puesta en práctica, que crea el producto de Software que se ha diseñado (esto combina el desarrollo de nuevos componentes con la reutilización o modificación de componentes anteriores); (4) la prueba, que garantiza que el producto de Software funciona como se

pretende. Los productos intermedios, como las especificaciones de requisitos y los diseños de Software, también se revisan en profundidad antes de pasar a la siguiente fase de desarrollo.

El Software no siempre se ha desarrollado de forma controlada, y en la actualidad hay algunos sistemas que presentan grandes dificultades para su mantenimiento. El organismo de normalización ISO (International Standards Organization) ha definido los requisitos de un sistema de gestión de calidad de carácter general que cubre el desarrollo de cualquier producto (ISO 9001) y ha publicado directrices específicas para aplicar esa norma al desarrollo de Software (ISO 9000-3). Una organización que ponga en práctica un sistema de gestión de calidad, según esa norma, puede ser auditada y recibir una certificación formal de su proceso de desarrollo.

Los ingenieros informáticos están implicados en un gran número de áreas de aplicación, cada vez son más. Algunos ejemplos son: la realización de transacciones rápidas de valores en el mercado bursátil, el almacenamiento, intercambio y presentación de información en Internet, los videojuegos, la mejora de imágenes obtenidas por telescopios y el control de marcapasos cardiacos. En todos los casos, los principios de la Ingeniería del Software ayudan a garantizar que los sistemas resultantes sean fiables y funcionen del modo requerido.

APÉNDICE D

ANTECEDENTES DE LA CRISIS DEL SOFTWARE

Al inicio de la era informática cuando el número de computadoras era muy reducido, y los programadores que existían no explotaban al máximo dicho equipo de cómputo, esto era gracias a que los lenguajes de programación que existían eran limitados.

La mayoría del software que se desarrollaba era utilizado por la misma persona la cual escribía, ejecutaba y si fallaba, corregía dicho desarrollo, esto fue debido a que no existía tanta competencia en el ambiente computacional, entonces los ejecutivos estaban seguros que esa persona estaría allí cuando existiese un error.

Con el paso del tiempo se fue desarrollando a gran medida el hardware, principalmente la velocidad y la capacidad de espacio para guardar información, sin embargo, el software no ha tenido esa importancia y por lo tanto no ha evolucionado a la misma velocidad que el hardware.

En la Tabla 1[v] se muestra como fueron evolucionando tanto el hardware como el software.

Generación y aplicaciones	**Tecnologías y arquitectura**	**Unidades Periféricas**	**Lenguajes de Programación y alfabeto**	**Sistema Operativo y administración**	**Aspectos cuantitativos y facilidades**	**Modelos**
Primera 1946 – 1959 Instrumento de cálculo	Tubos de vacío	Lectoras y perforadoras de tarjetas y cintas de papel, etc.	Ensambladores Primitivos Numérico	Varios básicos y rudimentarios	Memoria central 1000 a 8000 palabras Proceso: 10^4 OPS/SEG Precio de 1 a 2.5 Millones de DLS.	IBM – 850 BENDIX 6 – 15 UNIVAC 5590 BULL – PT IBM – 709

Segunda 1959 – 1964	Transistores Ferritas	Lectoras y perforadoras de tarjetas, impresoras y cintas magnéticas	Ensambladores Primarios Compiladores (FORTRAN y ALGOL) Números, letras y algunos caracteres especiales	Rudimentario control de periféricos inicia y termina tareas Primitiva planeación de producción con procesos masivos	Memoria central 8000 a 32000 palabras Proceso: 10^5 OPS/SEG Precio de 0.1 a 100 millones DLS. Existencia de bibliotecas	CDC – 160 IBM – 7090 IBM – 1401 BORROUGHS 5500 BCA 305 BENDIX 6-20 CDC 3600 CEC 6600
Tercera 1964 – 1970 Sistema de Información	Circuitos Integrados y memoria de película magnética Arquitectura Multiprogramación Multiproceso Sistemas de interrupción Optimización de códigos	Cintas y discos magnéticos Terminales de video y teletipos	Lenguajes de alto nivel COBOL, PL/1 Base de datos (DMS) Números, letras y caracteres especiales	Manejo de discos Multiproceso, memoria dinámica y virtual, etc. Compleja y especializada	Memoria central 64 a 256K palabras Proceso: 10^6 OPS/SEG Memoria secundaria 10^4 caracteres Precio de 5*10^ a 10^8 DLS. Edición y prueba interactiva de programas	IBM 360 BURROUGS 6700 PDP 10 PDP 11 UNIVAC 1106 CYBER 170
Cuarta 1970 – 1980 Sistemas de comunicación de información para negocios y uso personal	Microelectrónica Memorias MOS (Metal Oxide Sylicates) Arquitectura Proceso distribuido Uso de microprocesadores	Terminales inteligentes Discos y cintas magnéticas Equipo de graficación Lectores ópticos y Digitalizadores	Bases de datos distribuidas, Lenguajes interactivos, descriptivos y gráficos procedurables (cogen - link) Irrestricto Mayúsculas, Minúsculas Símbolos matemáticos Alfabeto árabe, japonés, etc.	Proceso sin interrupciones Comunicación entre máquinas Rutinas de recuperación, etc. Simple para equipos personales. Compleja para redes de proceso distribuido	Memoria central 64 a 10^7 caracteres Proceso: 10^7 OPS/SEG Memoria secundaria 10 ^ 10 ^ 8 DLS. Metaprocesadores, correo electrónico, manejadores de texto, etc.	IBM 4330 UNIVAC 1100 BURROUGS UGHS 6900, 7900 PRIME 650 MP 3100 VAX APPLE TR 80 IBM – PC
Quinta 1990 - ?	Estas máquinas se caracterizan por la utilización de enjambres procesadores microscópicos operando simultáneamente para recibir y clasificar información, por su capacidad básica de inferencia y generación de conocimientos y esquemas generales a partir de información particular así como su estrecha relación con el hombre. Su sistema de control sigue y utiliza los principios de las máquinas LISP y del flujo de datos.					

Tabla 1. Evolución del Hardware y Software

En la Tabla 1, se muestra la evolución del Hardware y Software. Para el caso del Software, en la actualidad existe un retraso, ya que no se le dio la importancia que se requería. Sobre todo por que el principal motivo de la superación del Software, es el cliente.

APÉNDICE E

IMPORTANCIA DE LA CRISIS DEL SOFTWARE[vi]

En 1968, en una conferencia patrocinada por el Comité de Ciencia de la OTAN se estableció, con el propósito de resolver la crisis del software, una nueva disciplina: la Ingeniería de Software. 25 años después, esta rama emergente de la ingeniería - aplicable a grandes y pequeños proyectos de software - esta evolucionando de una actividad intensiva en trabajo a una industria intensiva en capital, administrativa y tecnológicamente soportada.

Desde los inicios de la utilización comercial de las computadoras en la década de 1950, las espectaculares mejoras en la tecnología de la producción del hardware han conducido a la fabricación de maquinas cada vez más poderosas, disponibles a precios cada vez más bajos. En cambio, la demanda social de software cada día más grande y complejo ha sufrido una atención deficitaria y los costos para obtener este software ha crecido aun más y rápidamente. Mientras se han logrado algunas mejoras modestas en la calidad y productividad del software, el patrón general de actuación industrial en esta materia sigue siendo el de los calendarios incumplidos, los sobregiros presupuestales y los problemas de calidad. Esta condición de la industria es lo que generalmente se conoce como la crisis del software.

Con relación a los problemas de productividad, hacia 1970 se estimaba que la demanda de software (que de 1975 a 1985 crecería entre 21 y 23% anual) no podría ser atendida por la capacidad de producción, que apenas aumentaría entre 11.5 y 17% anual. En la década de 1970, el Departamento de Defensa de Estados Unidos (DoD, el mayor usuario de computadoras del mundo) informo que la mayor parte de los proyectos de software no fueron terminados (aun cuando pago por ellos), o los sistemas resultantes no se utilizaron o fueron entregados con errores importantes, de modo que tuvieron que abandonarlos o

reelaborarlos o modificarlos. Diversos autores han coincidido en señalar que, en promedio, las instalaciones gastan el 70% de su presupuesto para desarrollo y mantenimiento de software existente, el cual, para continuar en operación, requiere que el tiempo medio para efectuar una modificación o corregir una falla sea menor que el tiempo medio entre dos solicitudes de cambio o entre dos fallas. Y es que la probabilidad de que un programador logre corregir una falla al primer intento no es muy alta: si modifica solamente de cinco a diez instrucciones, la probabilidad máxima es de 50%; si modifica entre 40 y 50 instrucciones, la probabilidad de acertar baja a 20%.

En lo que respecta a los problemas de costo, de entrada hay que señalar que en 1985, los costos del software a nivel mundial ya excedían los 140 mil millones de dólares. A partir de entonces, han crecido a una tasa de 12% anual, de modo que para 1995 se habrán alcanzado los 435 mil millones de dólares, como se muestra en la figura 1. Particularmente, en lo que concierne al alto costo del software respecto del hardware, es preciso tener presente que en 1955, el costo del primero era menor al 10% del costo del segundo; que 20 años después, el costo del software era ya tres veces mayor que el del hardware y que para 1985, la diferencia se incremento nueve veces, tal como se muestra en la figura 2. Como botón de muestra, vale la pena citar que, en 1973, el DoD aseguro que sus costos de software representaban el 45% de su presupuesto informático, frente a un 38% de costos de operación (sobre los que incidían significativamente en los costos de operación) y solo un 17% de costos de hardware como se muestra en la figura 3.

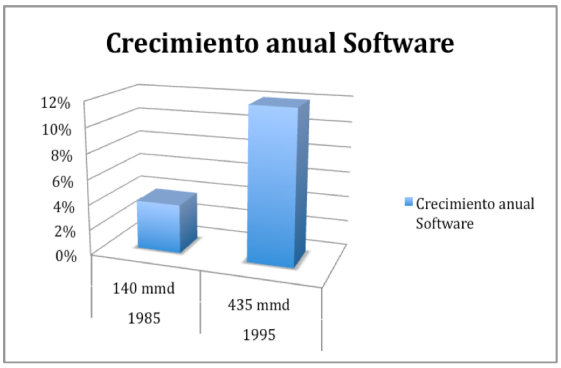


Figura 1. Tasa de crecimiento al 12 % anual

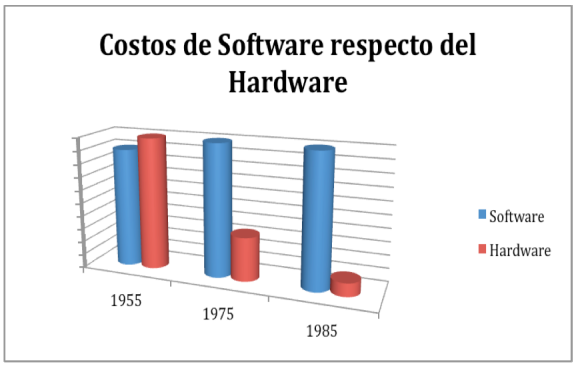


Figura 2. Diferencia de costos de Software vs Hardware

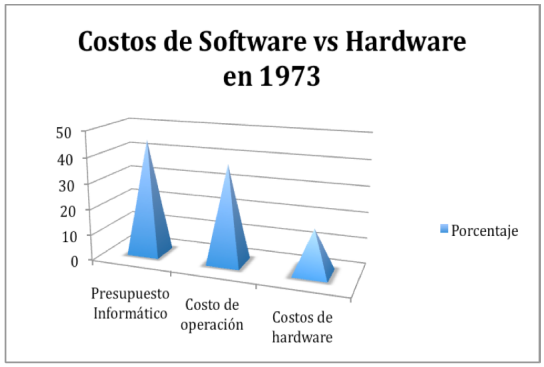


Figura 3. Costos de Hardware al 17%

Los problemas de calidad del software han provocado perdidas cuantiosas. En junio de 1962, el Mariner I[14] se salió de curso y tuvo que ser destruido: (este problema, costó 18.5 millones de dólares) y se debió a un error en uno de los programas que guiaban la nave.

A pesar de que existen avances reales en métodos y técnicas, en herramientas y en las habilidades de los grupos de desarrollo, la demanda crece más rápidamente que los progresos en la productividad del software. Mientras que ahora hay más personas involucradas en el desarrollo del software, los ingenieros siguen repitiendo los mismos errores que en la década de 1970. Se necesitan, entonces, mejores herramientas, técnicas y métodos y, lo que es más importante, mejor educación y entrenamiento (capacitación).

[14] Nave espacial de USA

APÉNDICE F

UML (Unified Modeling Language)[vii]

El UML (Lenguaje Unificado para la construcción de Modelos) se define como un "Lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de Software ..."[15] Es un sistema notacional (que, entre otras cosas, incluye el significado de sus notaciones) destinado a los sistemas de modelado que utilizan conceptos orientados a objetos.

El UML es un estándar incipiente de la industria para construir modelos orientados a objetos. Nació en 1994 por iniciativa de Grady Booch y Jim Rumbaugh para combinar sus dos famosos métodos: el de Booch y el OMT (*Object Modeling Technique*, Técnica de Modelado de Objetos). Más tarde se les unió Ivar Jacobson, creador del método OOSE (*Object-Oriented Software Engineering*, Ingeniería de Software Orientada a Objetos). En respuesta a una petición de OMG (*Object Management Group,* asociación para fijar los estándares de la industria) para definir un lenguaje y una notación estándar del lenguaje de construcción de modelos, en 1997 propusieron el UML como candidato.

Prescindiendo de la aceptación que pueda tener, este lenguaje recibió la aprobación de ipso en la industria, pues sus creadores representan métodos muy difundidos de la primera generación del análisis y diseño orientados a objetos. Muchas organizaciones dedicadas al desarrollo de software y los proveedores de herramientas CASE (Computer Aided Software Engineering) lo adoptaron, y muy probablemente se convertirá en el estándar mundial que utilizarán los desarrolladores, los autores y los proveedores de herramientas CASE.

[15] Booch, G., Jacobson, I. Y Rumbaugh, J. The UML specification documents. Santa Clara CA. 1997.

DIAGRAMAS DEL UML[16]

El UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el UML es un lenguaje, cuenta con reglas para combinar tales elementos. En lugar de indicarle cuales son los elementos y las reglas, se verán directamente los diagramas ya que se utilizarán para hacer el análisis del sistema.

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como *Modelo*. El modelo UML de un sistema es similar a un modelo a escala de un edificio junto con la interpretación del artista del edificio. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice como implementar dicho sistema.

DIAGRAMAS CONTENIDOS EN EL PROYECTO[viii]

Los diagramas que van a verse en este proyecto son:

- Modelamiento de Clases
- Casos de Uso
- Diagramas de Secuencia

MODELO DE CLASES

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de contenimiento.

Un diagrama de clases esta compuesto por los siguientes elementos:

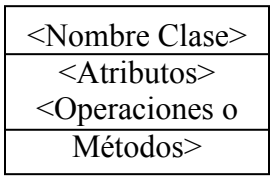
[16] Aprendiendo UML en 24 horas. Joseph Schmuller. Prentice may. México 2000.

- Clase: atributos, métodos y visibilidad.
- Relaciones: Herencia, Composición, Agregación, Asociación y Uso.

Elementos:

Clase:
Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de clase). A través de ella podemos modelar el entorno en estudio (una Casa, un auto, una Cuenta Corriente, etc.).

En UML, una clase es representada por un rectángulo que posee tres divisiones:



- Superior: contiene el nombre de la clase
- Intermedio: contiene los atributos (o variables de instancia) que caracterizan a la clase (pueden ser private, protected o public).
- Inferior: contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

Atributos y Métodos:

Atributos:
Los atributos o características de una clase pueden ser tres tipos, los que definen el grado de comunicación y visibilidad de ellos con el entorno, estos son:

- Public(+, ): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.

- Private(-, ): Indica que el atributo solo será accesible desde dentro de la clase (sólo sus métodos los pueden accesar).

- Protected(#, ): Indica que el atributo no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de las subclases que se deriven.

Métodos:
Los métodos u operaciones de una clase son la forma en como ésta interactúa con su entorno, estos pueden tener las características:

- Public(+, ): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.

- Private(-, ): Indica que el método solo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden accesar).

- Protected(#, ): indica que el método no será accesible desde fuera de la clase, pero si podrá ser accesado por métodos de la clase además de métodos de las subclases que se deriven.

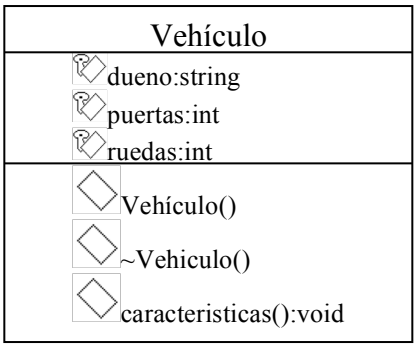
Relaciones entre clases:
Ahora ya definido el concepto de clase, es necesario explicar como se pueden interrelacionar dos o más clases (cada uno con características y objetivos diferentes).

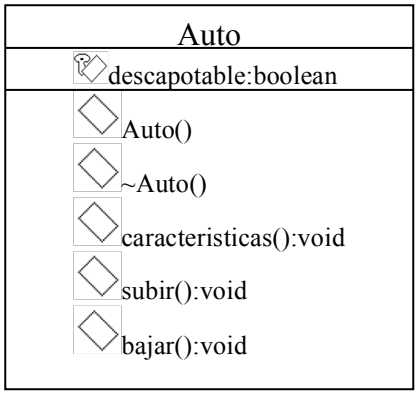
Antes es necesario explicar el concepto de cardinalidad de relaciones: En UML, la cardinalidad de las relaciones indica el grado y nivel de dependencia, se anotan en cada extremo de la relación y éstas pueden ser:

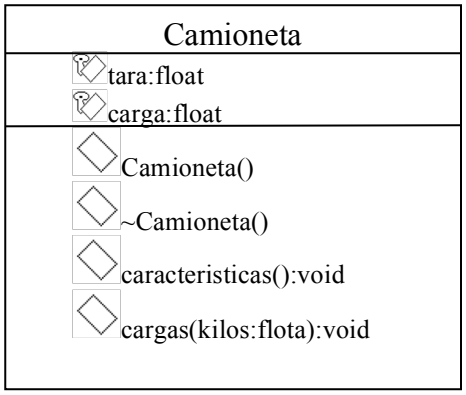
- uno o muchos: 1..* (1,.n)
- 0 o muchos: 0..* (0..n)
- número fijo: m (m denota el número)

Herencia (Especialización/Generalización): →

Indica que una subclase hereda los métodos y atributos especificados por una Super Clase, por ende la subclase además de poseer sus propios métodos y atributos, poseerá las características y atributos visibles de la superclase (public y protected), ejemplo:







En la figura se especifica que Auto y Camión heredan de Vehículo, es decir, Auto posee las Características de Vehículo (Precio, VelMax, etc.) además posee

algo particular que es Descapotable, es cambio Camión también hereda las características de Vehículo (Precio, VelMax, etc.) pero posee como particularidad propia Acoplado, Tara y Carga.

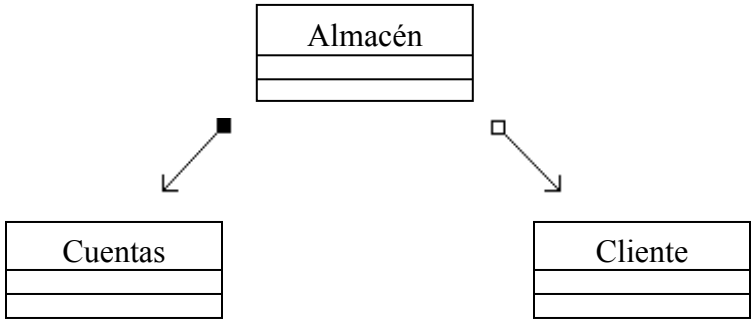
Cabe destacar que fuera de este entorno, lo único "visible" es el método Características aplicable a instancias de Vehículo, Auto y Camión, pues tiene definición pública en cambio atributos como Descapotable no son visibles por ser privados.

Agregación: ◇⟶

Para modelar objetos complejos, n bastan los tipos de datos básicos que proveen los lenguajes: enteros, reales y secuencias de caracteres. Cuando se requiere componer objetos que son instancias de clases definidas por el desarrollador de la aplicación, tenemos dos posibilidades:

- Por Valor: Es un tipo de relación estática en donde el tiempo de vida del objeto incluido esta condicionado por el tiempo de vida del que lo incluye. Este tipo de relación es comúnmente llamada **Composición** (el Objeto base se construye a partir del objeto incluido, es decir, es "parte/todo").
- Por Referencia: Es un tipo de relación dinámica, en donde el tiempo de vida del objeto incluido es independiente del que lo incluye. Este tipo de relación es comúnmente llamada **Agregación** (el Objeto base utiliza al incluido para su funcionamiento).

Un ejemplo es el siguiente:



En donde se destaca que:

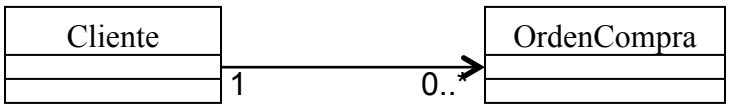
- Un Almacén posee Clientes y cuentas (los rombos van en el objeto que poseen las referencias).
- Cuando se destruye el Objeto Almacén también son destruidos los objetos Cuenta asociados, en cambio no son afectados los objetos Cliente asociados.
- La composición (por Valor) se destaca por un rombo relleno.
- La agregación (por Referencia) se destaca por un rombo transparente.

La flecha en este tipo de relación indica la navegabilidad del objeto referenciado. Cuando no existe tipo de particularidad la flecha se elimina.

Asociación: ⟶

La relación entre las clases conocida como Asociación, permite asociar objetos que colaboran entre si. Cabe destacar que no es una relación fuerte, es decir, el tiempo de vida de un objeto no depende del otro.

Ejemplo:



Un Cliente puede tener asociaciones muchas Ordenes de Compra, en cambio una orden de compra solo puede tener asociado un Cliente.

Dependencia o Instanciación (uso): - - - - - - ->

Representa un tipo de relación muy particular, en la que una clase es instanciada (su instanciación es dependiente de otro objeto/*clase). Se denota por una flecha punteada.

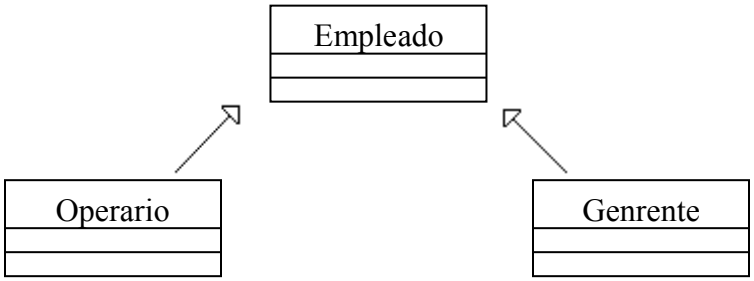
El uso más particular de este tipo de relación es para denotar la dependencia que tiene una clase de otra, como por ejemplo, una aplicación gráfica que instancia una ventana (la creación del Objeto Ventana esta condicionado a la instanciación proveniente desde el objeto Aplicación):

Cabe destacar que el objeto creado (en este caso la Ventana gráfica) no se almacena dentro del objeto que lo crea (en este caso la Aplicación).

Casos Particulares:

- Clase Abstracta:

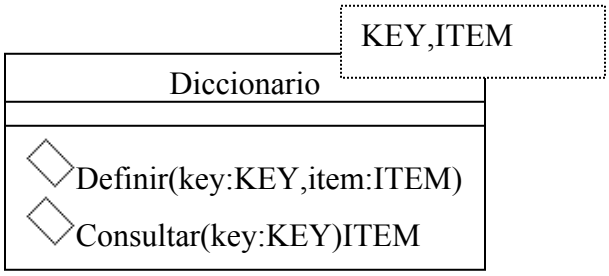
Una Clase Abstracta se denota con el nombre de la clase y de los métodos con letra "*itálica*". Esto indica que la clase definida no puede ser instanciada pues posee métodos abstractos (aún no han sido definidos, es decir, sin implementación). La única forma de utilizarla es definiendo subclases, que implementan los métodos abstractos definidos.



- Clase Parametrizada:

Una Clase Parametrizada se denota con un subcuadro en el extremo superior de la clase, en donde se especifican los parámetros que deben ser pasados a la clase para que esta pueda ser instanciada. El ejemplo más típico es el caso de un Diccionario en donde una llave o palabra tiene asociado un significado, pero en este caso las llaves y elementos pueden ser genéricos. La genericidad puede

venir dada de un Template (como en el caso de C++) o bien de alguna estructura predefinida (especialización a través de clases).



CASOS DE USO

Introducción:

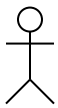
El diagrama de casos de uso representa la forma en como un Cliente (Actor) opera con el sistema en desarrollo, además de la forma, tipo y orden en como los elementos interactúan (operaciones o casos de uso).

Un diagrama de casos de uso consta de los siguientes elementos:

- Actor
- Casos de Uso
- Relaciones de Uso, Herencia y Comunicación

Elementos:

- Actor



Una definición previa, es que un Actor es un rol que un usuario juega con respecto al sistema. Es importante destacar el uso de la palabra rol, pues con

esto se especifica que un Actor no necesariamente representa a una persona en particular, sino más bien la labor que realiza frente al sistema.

Como ejemplo a la definición anterior, tenemos el caso de un sistema de ventas en que el rol de Vendedor con respecto al sistema puede ser realizado por un Vendedor o bien por el Jefe de Local.

- Casos de Uso 

Es una operación/tarea especifica que se realiza tras una orden de algún agente externo, sea desde una petición de actor o bien desde la invocación desde otro caso de uso.

- Relaciones

  - ❖ Asociación ⟶

Es el tipo de relación más básica que indica la invocación desde un actor o caso de uso a otra operación (caso de uso). Dicha relación se denota con una flecha simple.

  - ❖ Dependencia o Instanciación - - - - - - - ›

Es una forma muy particular de relación entre clases, en la cual una clase depende de otra, es decir, se instancia (se crea). Dicha relación se denota con una flecha punteada.

  - ❖ Generalización ⟞

Este tipo de relación es uno de los más utilizados, cumple una doble función dependiendo de su estereotipo, que puede ser de Uso (<<uses>>) o de Herencia (<<extends>>).

Este tipo de relación esta orientado exclusivamente para casos de uso (y no para actores).

extends: Se recomienda utilizar cuando un caso de uso es similar a otro (características).

uses: Se recomienda utilizar cuando se tiene un conjunto de características que son similares en más de un caso de uso y no se desea mantener copiada la descripción de la característica.

De lo anterior cabe mencionar que tiene el mismo paradigma en diseño y modelado de clases, en donde esta la duda clásica de usar o heredar.

DIAGRAMA DE SECUENCIAS

El diagrama de secuencias representa la forma en como un cliente (Actor) u Objetos (Clases) se comunican entre si en petición a un evento. Esto implica recorrer toda la secuencia de llamadas, de donde se obtienen las responsabilidades claramente.

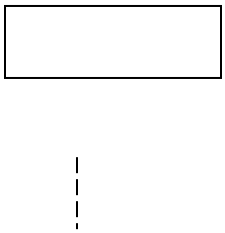
Dicho diagrama puede ser obtenido de dos partes, desde el diagrama estático de clases o el de casos de uso (son diferentes).

Los componentes de un diagrama de secuencias son:

- Un Objeto o Actor.
- Mensaje de un objeto a otro objeto.
- Mensaje de un objeto a si mismo.

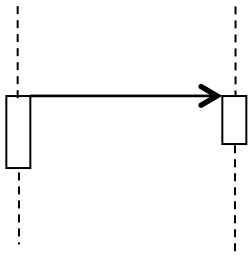
ELEMENTOS

- Objeto/Actor:



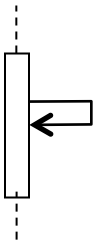
El rectángulo representa una instancia de un objeto en particular, y la línea punteada representa las llamadas a métodos del objeto.

- Mensaje a otro objeto:



Se representa por una flecha entre un objeto y otro, representa la llamada de un método (operación) de un objeto en particular.

- Mensaje el mismo objeto.



No solo llamadas a métodos de objetos externos pueden realizarse, también es posible visualizar llamadas a métodos desde el mismo objeto en estudio.

APÉNDICE G

PROGRAMACIÓN ESTRUCTURADA[17]

La PE[18] es un estilo de programación que se basa en el uso de tres estructuras básicas de control lógico:

1. SECUENCIA.
2. SELECCIÓN.
3. ITERACIÓN.

Un programa estructurado se compone de funciones, segmentos, módulos y/o rutinas, cada una con una sola entrada y una sola salida. Cada uno de estos módulos (aún en el mismo programa completo) se denomina programa apropiado, cuando además de estar compuesto solamente por las tres estructuras básicas tiene sólo una entrada y una salida y en ejecución no tiene partes por las cuales nunca pasa ni tiene ciclos infinitos.

La PE tiene una teoría fundamental, la cual afirma que cualquier programa, no importa el tipo de trabajo que ejecute, puede ser elaborado utilizando únicamente las tres estructuras básicas (secuencia, selección, iteración).

DEFINICIÓN DE LAS ESTRUCTURAS BÁSICAS DE CONTROL LÓGICO

1. SECUENCIA

Indica que las instrucciones de un programa se ejecutan una después de la otra, en el mismo orden en el cual aparecen en el programa. Se representa

[17] Ingeniería del Software. Roger S. Pressman. Edit. Mc Graw - Hill. México 1993.
[18] Programación Estructurada

gráficamente como una caja después de otra, ambas con una sola entrada y una única salida.

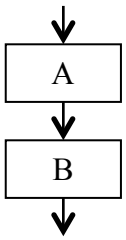


Diagrama de flujo

Caja de estructuras

Las cajas A y B pueden ser definidas para ejecutar desde una simple instrucción hasta un conjunto de varias de ellas (conformando un módulo), siempre y cuando guarden una lógica apropiada.

2. SELECCIÓN

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

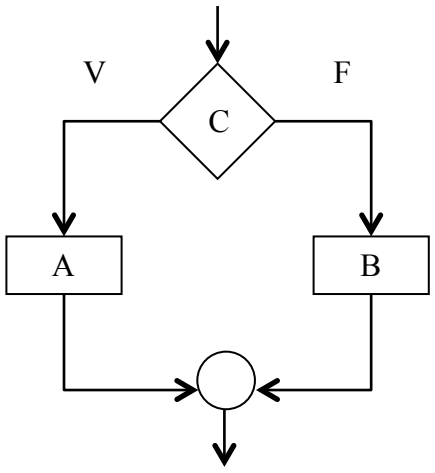
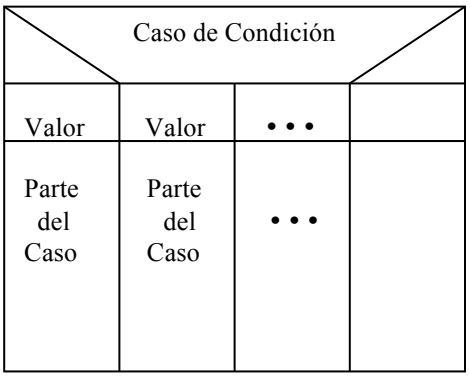


Diagrama de flujo



Caja de estructuras

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

3. ITERACIÓN

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

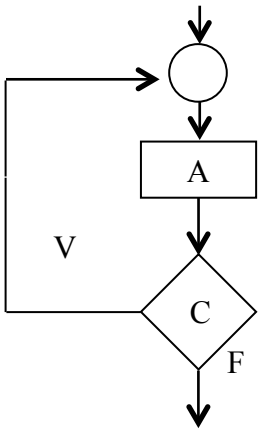
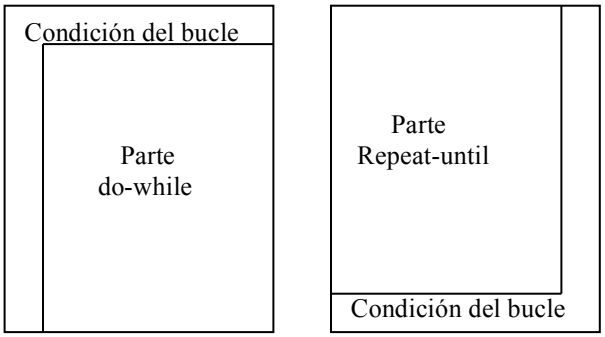


Diagrama de flujo



Caja de estructuras

Aquí el bloque A se ejecuta repetidamente mientras que la condición C se cumpla o sea cierta. También tiene una sola entrada y una sola salida; igualmente A puede ser cualquier estructura básica o conjunto de estructuras.

VENTAJAS DE LA PE

Con la PE, elaborar desarrollos sigue siendo una labor que demanda esfuerzo, creatividad, habilidad y cuidado. Sin embargo, con este nuevo estilo podemos obtener las siguientes ventajas:

1. Los programas son más fáciles de entender. Un programa estructurado puede ser leído en secuencia, de arriba hacia abajo, sin necesidad de estar saltando de un sitio a otro en la lógica. La estructura del programa es más clara puesto que las instrucciones están más ligadas o relacionadas entre si, por lo que es más fácil comprender lo que hace cada función.

2. Reducción del esfuerzo en las pruebas. El programa se puede tener listo para producción normal en un tiempo menor del tradicional; por otro lado, el seguimiento de las fallas o depuración (debugging) se facilita debido a la lógica más visible, de tal forma que los errores se pueden detectar y corregir más fácilmente.

3. Reducción de los costos de mantenimiento.

4. Programas más sencillos y más rápidos.

5. Aumento en la productividad del programador.

6. Se facilita la utilización de las otras técnicas para el mejoramiento de la productividad en programación.

7. Los programas quedan mejor documentados internamente.

DESVENTAJAS DE LA PE

1. Una corrección implica la repetición de todo el diagrama.

2. Desde el punto de vista usuario son complejos, ya que usan una simbología no manejada por los mismos.

APÉNDICE H

PROGRAMACIÓN ORIENTADA A OBJETOS (POO)[19]

Actualmente una de las áreas más interesante en la industria y a nivel académico es la programación orientada a objetos (POO). Esta promete mejoras de un alcance mayor en la forma de diseño, desarrollo y mantenimiento del software, ofreciendo una solución a largo plazo a los problemas y menos preocupaciones que han existido desde el comienzo en el desarrollo de software como:

1. La falta de portabilidad y rehusabilidad del código
2. Código que es difícil de modificar
3. Ciclos de desarrollo largos
4. Técnicas de codificación no entendibles.

Un lenguaje de programación orientado a objetos trata de resolver estos problemas, usando las tres características que lo definen: debe estar basado en objetos, basado en clases y capaz de tener herencia de clases. Algunos lenguajes de programación cumplen uno o dos de estas características; solo unos cuantos cumplen todas ellas. Por lo regular la parte más difícil de soportar es la herencia.

El concepto de programación orientada a objetos (POO) no es nuevo, lenguajes clásicos de los 70's se basan en ella. Dado que la POO se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo.

[19] http://lucas.simplenet.com/trabajos/objetos/objetos.html

El elemento fundamental de la POO es, como su nombre lo indica, el objeto. Se define un objeto como un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización. El objeto no es un dato simple, pero si contiene en su interior ciertos número de componentes bien estructurados, tampoco es una entidad aislada, sino que forma parte de una organización jerárquica.

"La programación orientada a objetos representa una forma totalmente distinta de ver la computación". Podemos decir que sus principios están cimentados en la programación y el análisis estructurado, pero hacen una serie de refinamientos e implantan una serie de características que la colocan como un paradigma evolutivo. Estos dos enfoques no son los único existentes, pero los programadores e inclusive los analistas, se encuentran encerrados en el paradigma con el cual fue creado el lenguaje con el que normalmente trabajan (y generalmente se trabaja uno solo), Bobrow y Stefik listan cinco tipos:"[20]

- Orientado a los procedimientos	Algoritmos
- Orientados a los objetos	Clases y objetos
- Orientados a la lógica	Metas, expresados en cálculo de predicados
- Orientados a reglas	Reglas SI-ENTONCES
- Orientados a constreñimiento	Relaciones invariantes

Actualmente podemos incluir un tipo más:

- Orientado a eventos	Ambientes gráficos

[20] Fundamentos de programación. Ernesto Peñalosa Romero. 3ª Edición. UNAM Campus Aragon. México 2001.

ESTRUCTURA DE UN OBJETO

Un objeto puede considerarse como una especie de cápsula dividida en tres partes, las cuales son:

1. Relaciones
2. Propiedades
3. Métodos

Cada uno de estos desempeña un papel completamente independiente.

Las Relaciones, permiten que el objeto se inserte en la organización y están formadas principalmente por punteros a otros objetos.

Las Propiedades, distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

Los Métodos, son las operaciones que pueden realizarse sobre el objeto, que normalmente estarán incorporados en forma de programas (código) que el objeto es capaz de ejecutar y que también pone a disposición de sus descendientes a través de la herencia.

CLASES

Una clase puede considerar como una plantilla para crear objetos de esa clase o tipo. Una clase describe los métodos y atributos que definen las características comunes a todos los objetos de esa clase. Precisamente la clave de la programación orientada a objetos está en abstraer los métodos y atributos comunes a un conjunto de objetos y encapsularlos en una clase.

Una clase es un tipo de objeto definido por el usuario. En otras palabras, una clase equivale a la generalización de un tipo específico de objeto. Un objeto es la concreción[21] de una clase. Para disponer de un objeto, primero se abstraen las características comunes de ese grupo de objetos, después se delimitan esas características comunes (clase) y por último se pone nombre a uno o más ejemplares de la clase (objetos).

ENCAPSULAMIENTO

El encapsulamiento u ocultación de información se refiere a la práctica de incluir dentro de un objeto todo lo que necesita, de tal forma que ningún otro objeto necesite conocer nunca su estructura interna. Esta característica permite ver un objeto como una caja negra, en la que se han metido de alguna manera toda la información relacionada con dicho objeto. Esto nos permitirá manipular los objetos como unidades básicas, permaneciendo oculta su estructura interna.

HERENCIA

La herencia es el mecanismo para compartir automáticamente métodos y atributos entre clases y subclases. Por ejemplo si declaramos la clase profesor como subclase de empleado, todos los métodos y variables asociadas con la clase empleado son automáticamente heredados por la subclase profesor. Si la clase empleado contiene métodos inapropiados para la subclase profesor, estos pueden eludirse, escribiendo nuevos métodos y almacenándolos como parte de la clase profesor, o también pueden redefinirse para que respondan de forma distinta a como lo hacen en la clase base.

[21] Reunión de varias partículas en una masa sólida.

Esta característica de la POO está fuertemente ligada a la reutilización del código. Esto es, el código de cualquiera de las clases existentes puede ser utilizado sin mas que crear una clase derivada de ella.

El concepto de herencia conduce a una estructura jerarquizada de clases, lo cual no significa que en POO todas las relaciones entre clases deban ajustarse a una estructura jerárquica.

La herencia puede ser también múltiple; esto significa que puede derivarse de dos o mas clases base.

POLIMORFISMO

Ésta característica permite implementar múltiples formas de un mismo método, dependiendo cada una de ellas de la clase sobre la que se realice la implementación. Esto hace que se pueda acceder a una variedad de métodos distintos (todos con el mismo nombre) utilizando exactamente el mismo medio de acceso."[22]

VENTAJAS DE LA POO

1. Facilita la reutilización del software. A través de la herencia se nos permite utilizar en un objeto las operaciones implementadas para otros sin esfuerzo adicional. Por otro lado la encapsulación nos facilita el uso de objetos que no hemos definido ni implementado.

2. Facilita la construcción de programas portables. Es posible diseñar una capa de objetos que se comunique con la máquina o el sistema operativo, y sobre ésta los objetos que dan funcionalidad a la aplicación. Una migración a otra

[22] Programación orientada a objetos con C++. Francisco Javier Cevallos Sierra. AlfaOmega Grupo Editor. México 1998.

arquitectura sólo requerirá cambios en dicha capa, y el encapsulamiento nos garantiza que los cambios se van a limitar a esos objetos.

3. Facilita el mantenimiento. El encapsulamiento nos garantiza que las modificaciones realizadas en un objeto tendrán un efecto limitado. Si un elemento de datos se accede directamente y en un momento dado cambia de tipo o formato, habrá que localizar todos los puntos en los que se accede a dicho elemento y modificarlos en consecuencia. Si este elemento de datos está encapsulado en un objeto y siempre es accedido mediante las operaciones disponibles, un cambio como el indicado se limitará al objeto, del que sólo habrá que cambiar el elemento de datos y las operaciones del objeto que lo utilizan. Lo recomendable es desconocer la implementación interna del objeto, nosotros necesitaremos utilizar un objeto que ofrece una serie de funcionalidades, pero no nos interesa de que forma nos ofrece las mismas.

4. Provoca que las tareas de análisis, diseño e implementación sean más intuitivas, ya que se manejan objetos, concepto con el que todos estamos familiarizados, y estructuradas, ya que podemos asignar las tareas de diseño e implementación a nivel de objetos. Se debe recordar que los objetos en nuestros diseños van a representar objetos presentes en el mundo real.

DESVENTAJAS DE LA POO

1. Curvas de aprendizaje largas, debido principalmente a que implica un cambio de mentalidad, y no sólo el aprender un nuevo lenguaje.

2. Dificultad en determinar las características de un objeto. Debido a que un objeto no se define aisladamente, sino que depende de las relaciones con otros objetos, y el establecimiento de nuevas relaciones puede implicar un cambio en la definición del objeto y viceversa.

Por todo ello es conveniente iterar, esto es, repetir el proceso de definición de objetos y de identificación de relaciones, tantas veces como sean necesarias para alcanzar un modelo estable. En muchos casos resulta complicado llegar a un modelo de objetos definitivo con el que se pueda abordar el problema planteado de forma completa.

APÉNDICE I

LENGUAJES DE PROGRAMACIÓN

Los lenguajes de programación son un vehículo de comunicación entre los humanos y las computadoras. El proceso de codificación – comunicación mediante un lenguaje de programación – es una actividad humana. Es por ello que las características psicológicas del lenguaje afectan directamente a la calidad de la comunicación. Las características de ingeniería de un lenguaje tienen un impacto importante sobre el éxito de un proyecto de desarrollo de software. Finalmente, las características técnicas de un lenguaje pueden influenciar la calidad del diseño ya que van dirigidas hacia un lenguaje de programación específico.

Las características de los lenguajes de programación se centran en las necesidades que puede tener un proyecto específico de desarrollo de software. Aunque algunas de ellas se tornan ocultas como requisitos para el código fuente. Se establece un conjunto general de características de ingeniería y son: 1) facilidad de traducción del diseño al código fuente, 2) eficiencia del compilador, 3) portabilidad del código fuente, 4) disponibilidad de herramientas de desarrollo y 5) facilidad de mantenimiento.

Sin embargo, lo anterior no tendría sentido sin la elección de un lenguaje de programación adecuado para un proyecto específico, o peor aún si solo se dispusiera de un lenguaje o en su defecto el cliente demanda uno en particular.

“...el arte de elegir un lenguaje de programación comienza con el propio problema planteado, al decidir cuales son sus requisitos y su importancia relativa, ya que probablemente será imposible satisfacerlos todos por igual (con

un único lenguaje)... se deben medir los lenguajes disponibles teniendo al lado una lista de requisitos..."[23]

Independientemente de la elección del lenguaje de programación, se deberá tomar en cuenta la planificación, el diseñó, la codificación, la prueba y el mantenimiento de un proyecto. El papel del lenguaje de programación ha de tenerse presente en todo momento.

LENGUAJE VISUAL BASIC (VB)

Es un sistema de desarrollo diseñado especialmente para crear aplicaciones con una interfaz gráfica, en una forma rápida y sencilla. Para soportar este tipo de desarrollos, Visual Basic utiliza fundamentalmente dos herramientas, una que le permite realizar los diseños gráficos y otra programar en un lenguaje de alto nivel.

Visual Basic Está centrado en dos tipos de objetos, Ventanas y Controles, que permiten diseñar sin programar, una interfaz gráfica (mecanismo de comunicación entre el usuario y la aplicación) para una aplicación. Técnicamente tiene características de orientación a objetos pero no es totalmente orientado a objetos.

Ventajas: VB es el lenguaje de programación visual con mayor soporte a nivel mundial. Se puede obtener ayuda e información por diversos medios y es realmente sencillo encontrar desarrolladores especializados. Su facilidad de aprendizaje y su interfaz amigable al usuario.

[23] Meek, B., and P. Heath (eds.) Guide to good programming, Halsted Press (Wiley),1980.

Desventajas: no cuenta con apuntadores estándar, ni con programación orientada a objetos real. En ocasiones, el manejo automático de la memoria (con datos de cadena y Variant) puede producir pequeños y muy ocasionales retrasos en los programas.

APÉNDICE J

MANUAL DE INSTALACIÓN

Este manual muestra paso a paso como instalar ENDER. Se deben seguir los siguientes pasos.

1. Se debe introducir en la unidad de CD/DVD el CD-ROM que contiene el instalador de ENDER.
2. Una vez introducido el CD-ROM de instalación, se debe abrir una carpeta para ver el contenido. Cuando se tengan los archivos a la vista se le da doble clic al archivo ejecutable "setup", tal y como se muestra en la figura 1.

Figura 1. Archivo ejecutable "setup".

3. Cuando se ejecuta el archivo "setup", se despliega una pantalla como la que se muestra en la figura 2, se le da clic en el botón "Aceptar" para continuar con la instalación

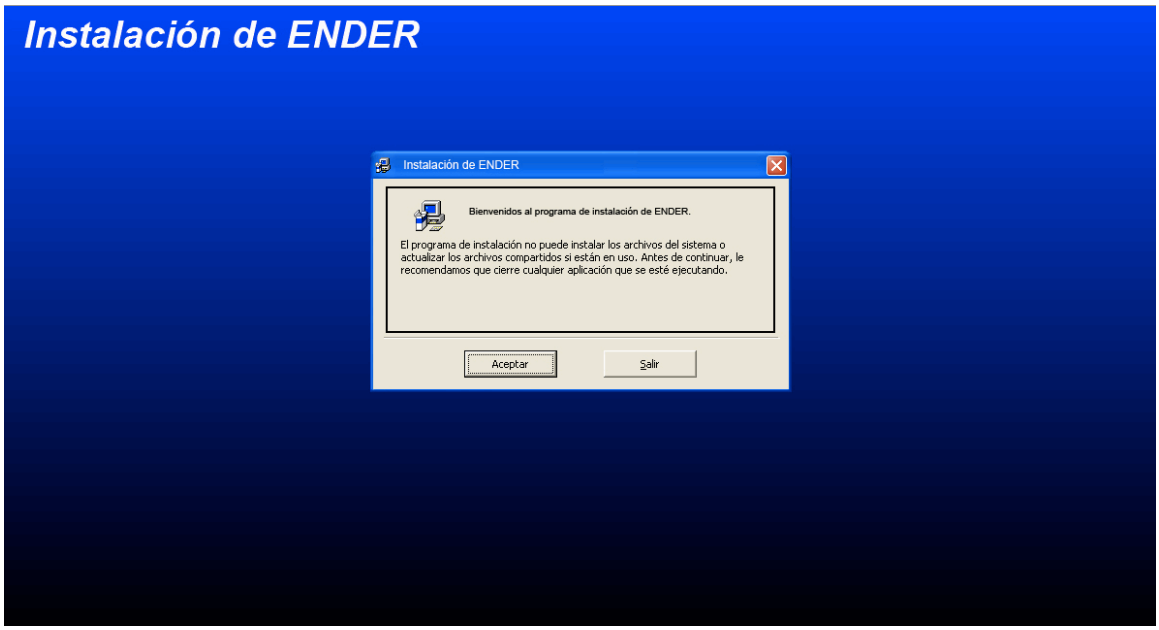


Figura 2. Pantalla de bienvenida a la instalación de ENDER.

4. La pantalla la de figura 3, funciona para elegir la carpeta donde estará instalado ENDER, sólo hay que dar clic en la imagen de la computadora para seguir con la instalación.

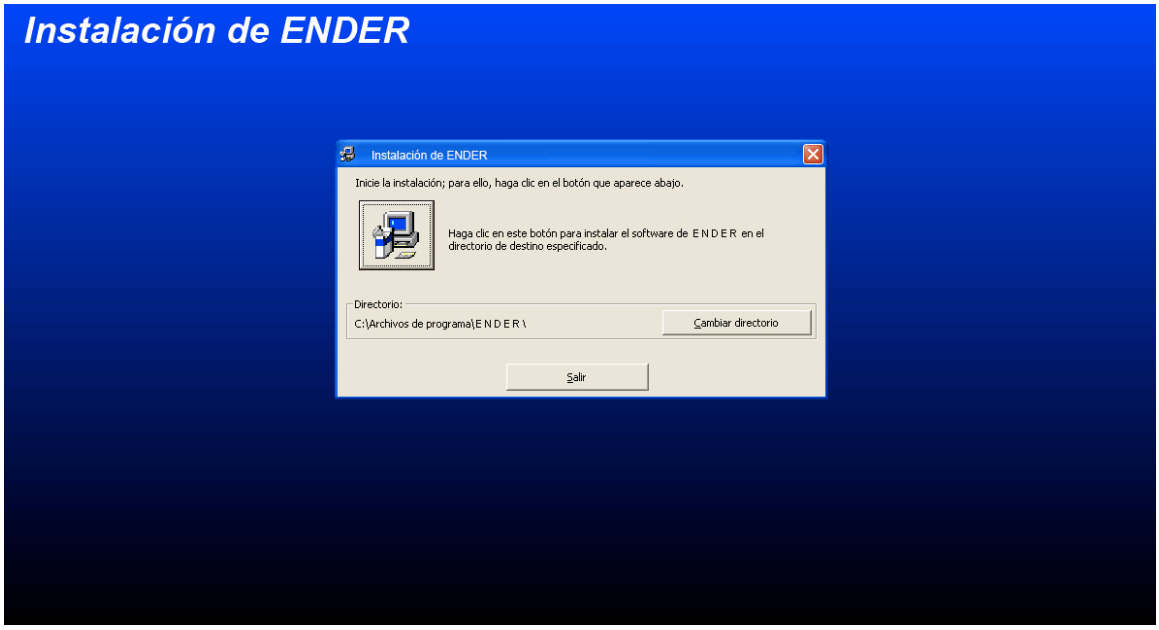


Figura 3. Pantalla para elegir carpeta donde se instalará ENDER.

5. En la figura 4 se muestra el grupo donde se instalará ENDER, sólo hay que darle clic en continuar.

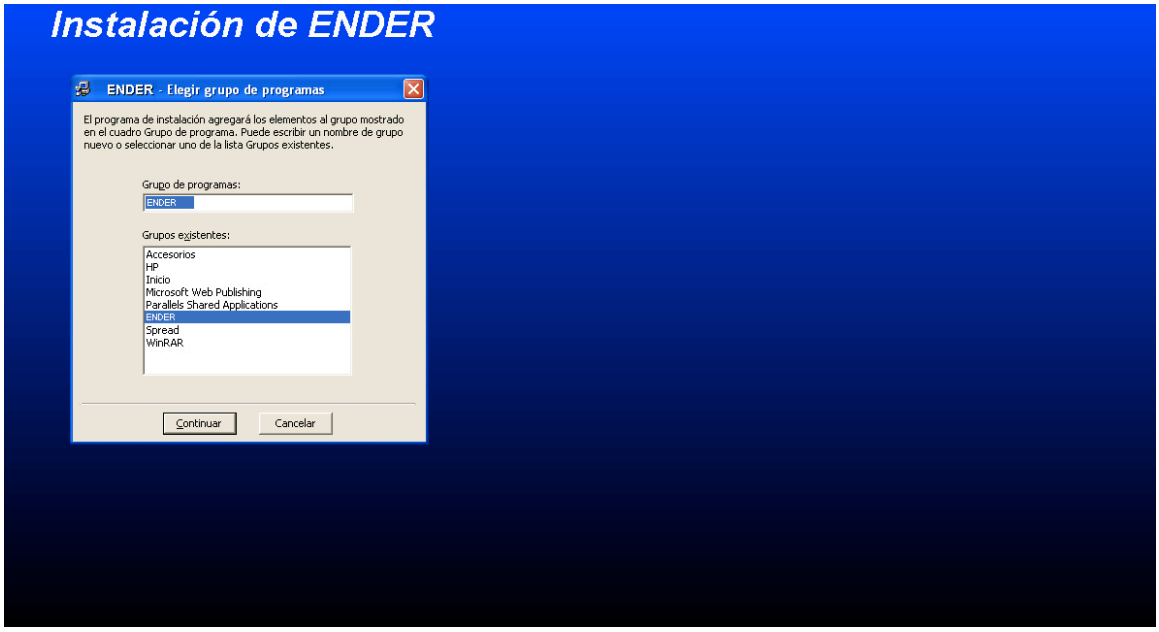


Figura 4. Grupo donde se instalará ENDER.

6. La imagen de la figura 5 muestra un probable error que puede ocurrir si se tienen las versiones actuales de los archivos requeridos, simplemente hay que darle un clic en el botón “No a todo”, para que se queden los archivos de versión reciente.

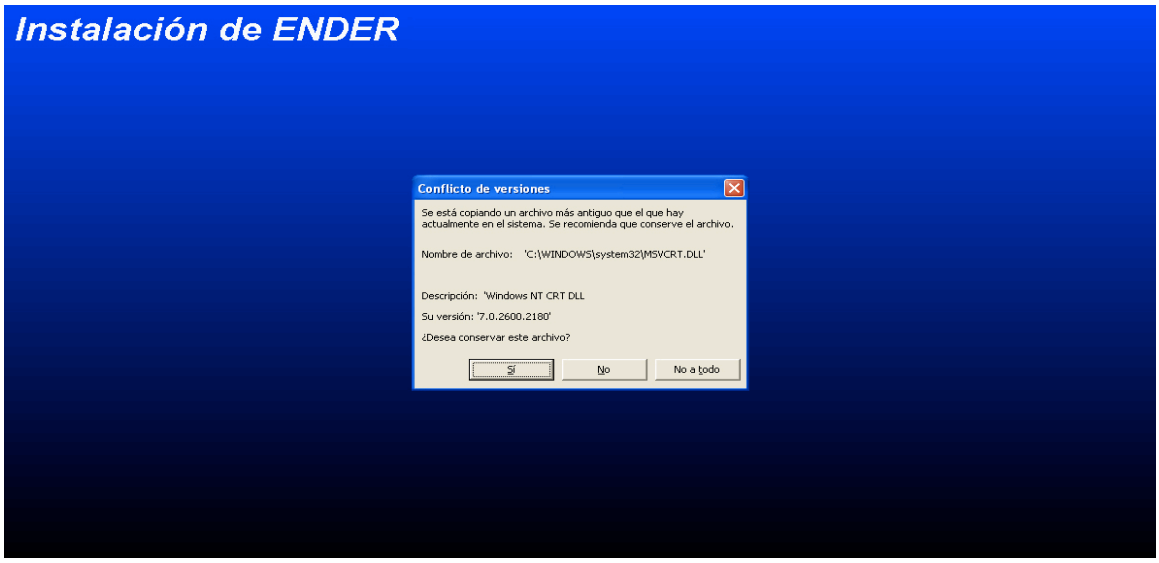


Figura 5. Posible error de instalación de ENDER.

7. En la figura 6, se le da seguimiento a la instalación de ENDER.

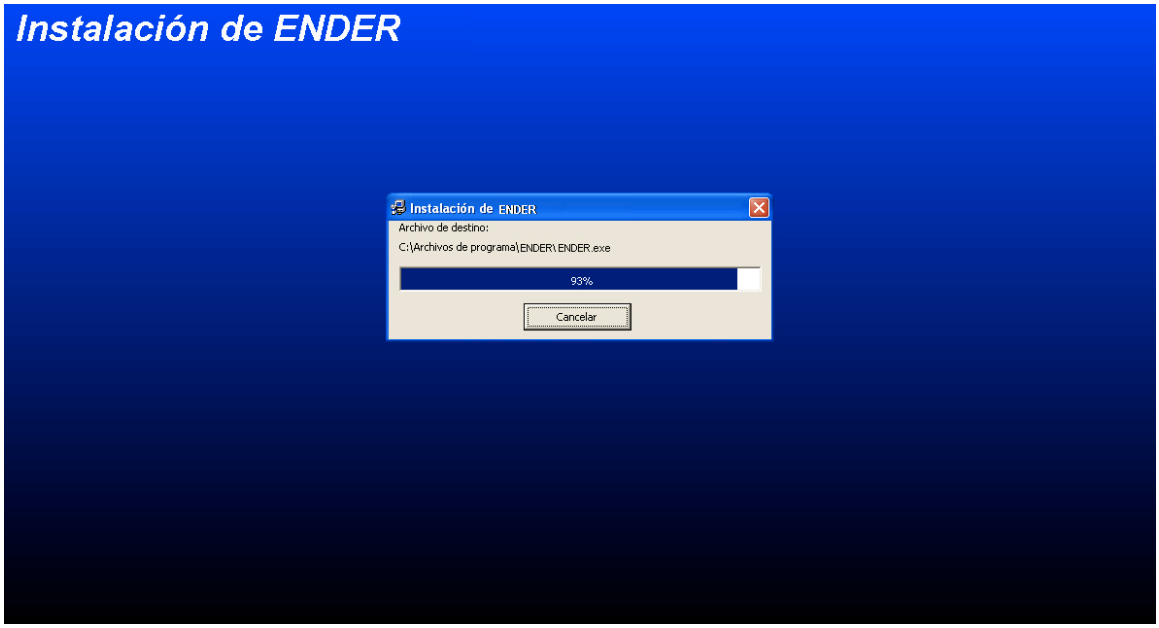


Figura 6. Seguimiento de la instalación de ENDER.

8. Una vez que ha finalizado de instalar ENDER, aparece una pantalla como la que se muestra en la figura 7.

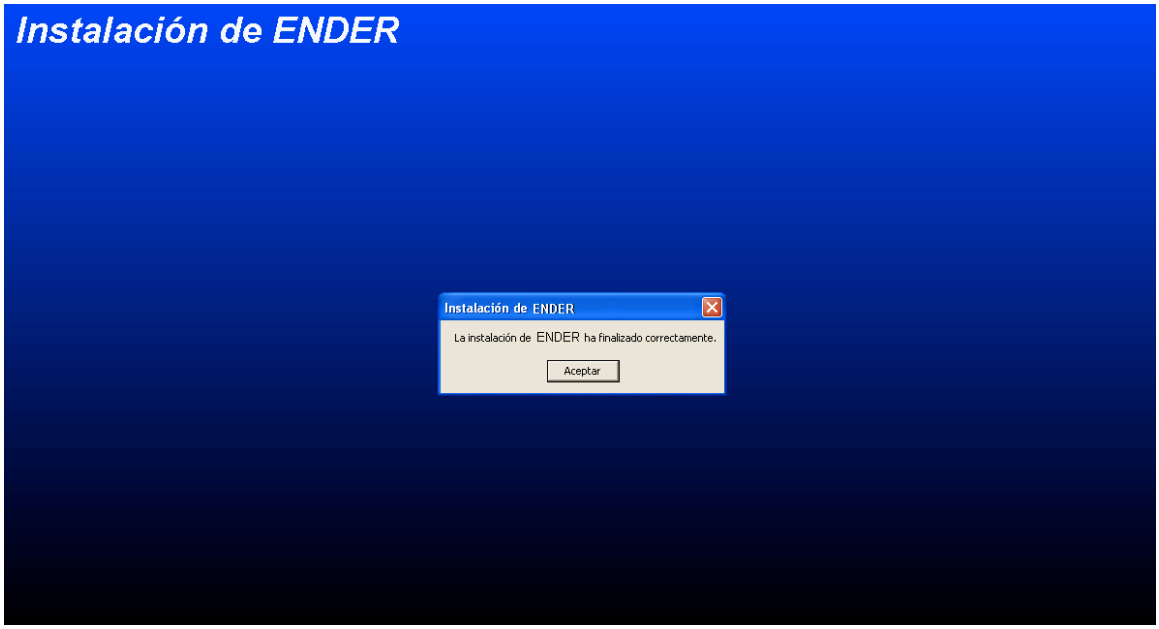


Figura 7. Instalación de ENDER finalizada.

APÉNDICE K

MANUAL DE USO Y MANEJO

El presente manual esta diseñado para mostrarle al usuario final el uso y manejo de ENDER.

Para iniciar ENDER, se debe dar clic en el botón “Inicio” de Windows, después en “Todos los programas” y al final se elige el grupo ENDER para seleccionar el ejecutable ENDER. Lo anterior se muestra en la figura 1.

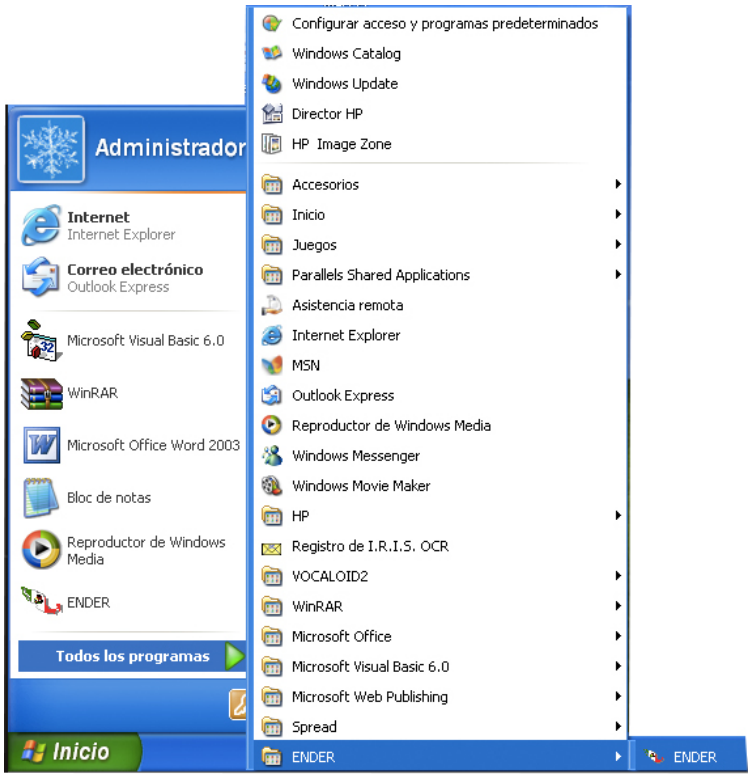


Figura 1. Ejecución de ENDER.

Una vez que se ha ejecutado ENDER, se muestra una ventana como la de la figura 2. Después de esto se debe dar clic en “Ok”, para continuar con la aplicación.

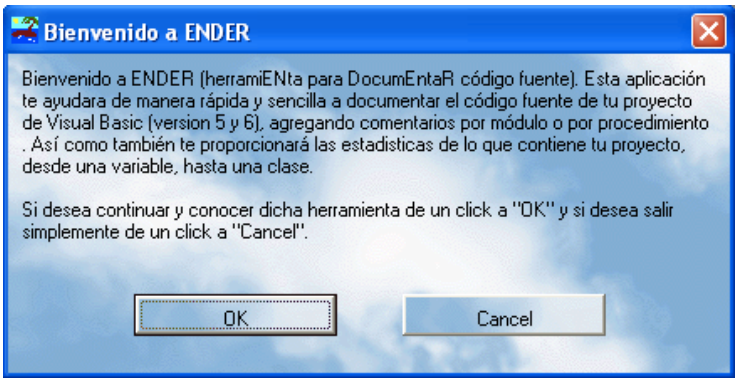


Figura 2. Ventana de bienvenida a ENDER.

Después de la ventana de bienvenida, aparece la aplicación de ENDER como se muestra en la figura 3.

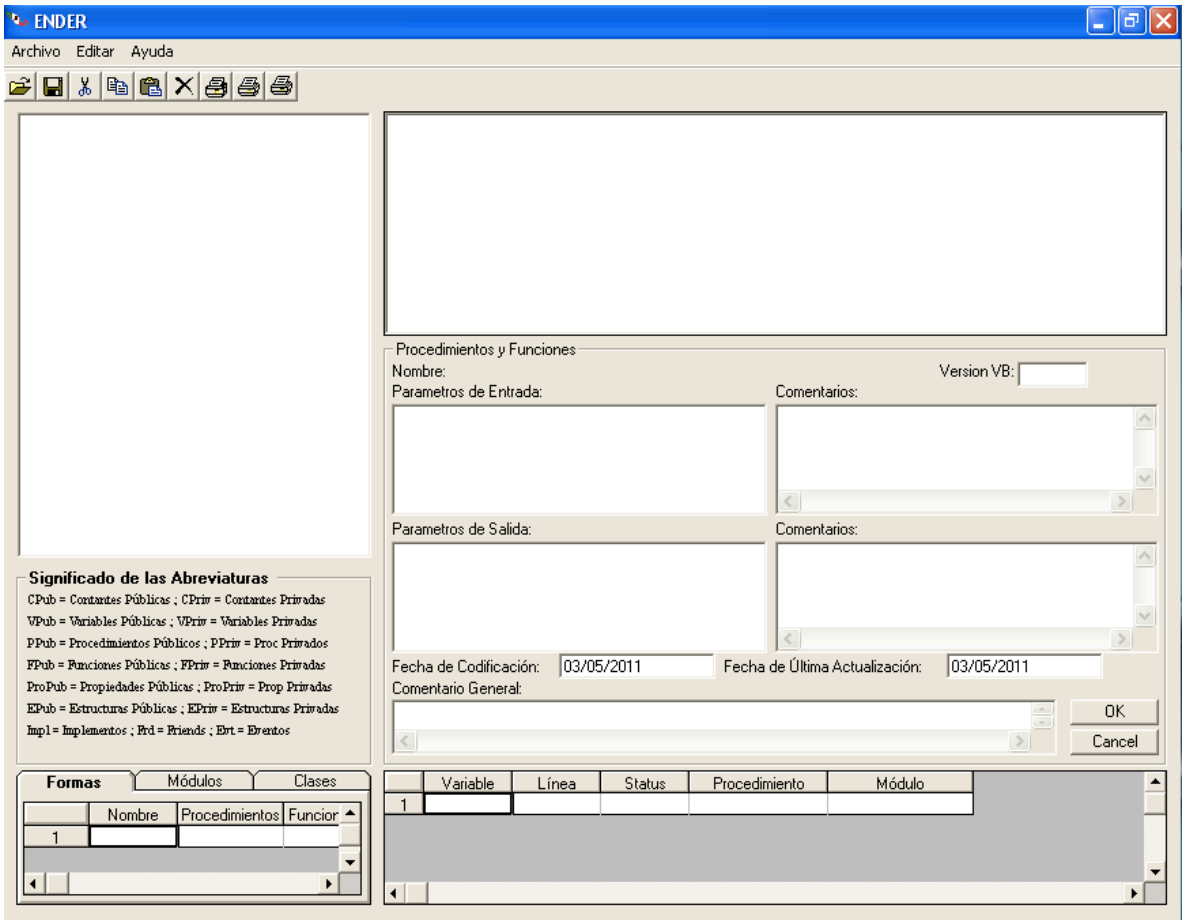


Figura 3. ENDER en ejecución.

Cuando ya este en ejecución ENDER hay que elegir el menú Archivo y seleccionar la opción Abrir, tal como se muestra en la figura 4. Esto con el fin de elegir un proyecto que fue elaborado en Visual Basic 5 o 6.

Figura 4. Menú Archivo

Una vez que se ha elegido el proyecto, el árbol de nodos se complementa de forma similar como la figura 5.

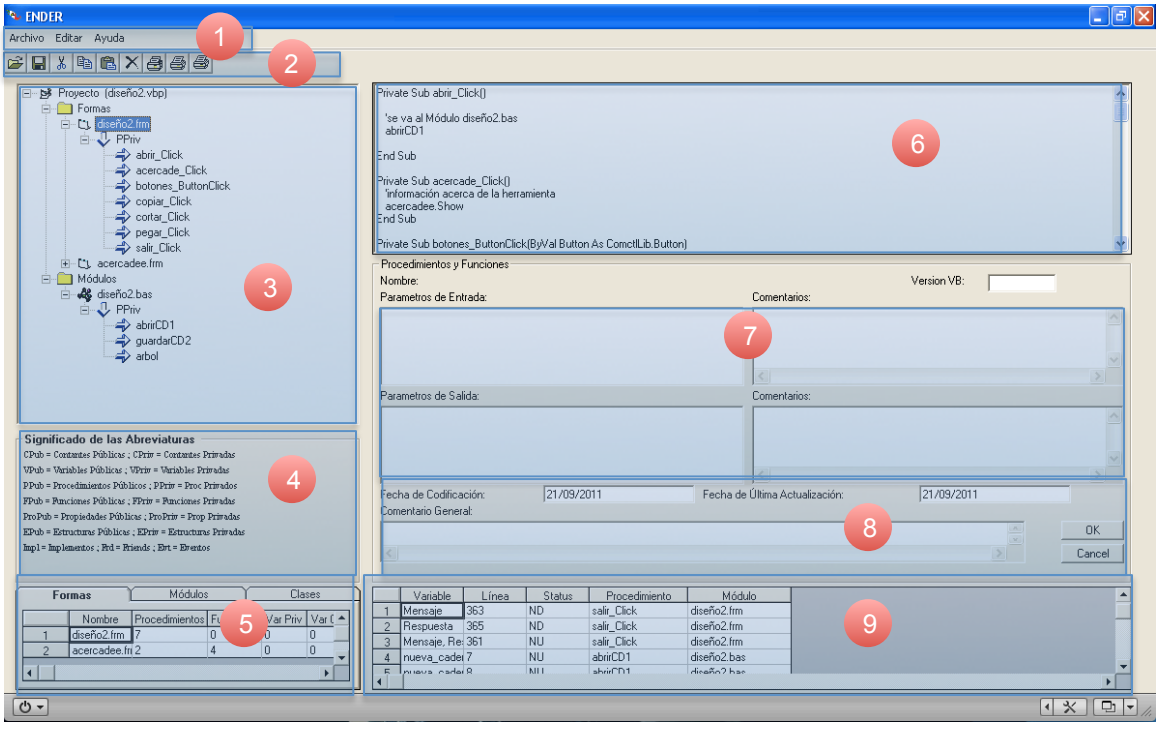


Figura 5. Ejemplo de un proyecto analizado

En la figura 5 se han colocado números por sección, a continuación se explica cada uno de ellos.

1. Barra de Menús. Ésta contiene las herramientas comunes como es Abrir, Guardar, Imprimir, Copiar, Cortar, Pegar y Acerca de.
2. Barra de Herramientas. Contiene los íconos abrir, guardar, cortar, copiar, pegar, eliminar, imprimir código, imprime resumen por módulo e imprime resumen de variables.
3. Árbol de nodos. Se forma cuando se elige abrir un proyecto, dentro de ésta se muestran los formularios, módulos y clases.
4. Significado de las abreviaturas. Dentro de ENDER se manejan abreviaturas, motivo por el cual se creo esta sección para que el usuario conozca los significados.
5. Contenido del proyecto. Una forma de encontrar el número de formularios, módulos y clases contenidos en un proyecto es por medio de esta sección, que comprende el nombre, número de procedimientos y funciones, variables privadas y globales.
6. Código fuente. Esta sección muestra el código fuente del elemento señalado en el árbol de nodos.
7. Parámetros de entrada y salida. Creada para señalar los parámetros de entrada y salida, a un lado una caja de textos para poner un comentario.
8. Fechas y comentario general. La sección de fechas puede modificarse de acuerdo la etiqueta que le corresponda, ya sea de codificación o de actualización. Los comentarios generales son para que aparezcan en el encabezado del código fuente del formulario, módulo o clase.
9. Estatus de variables. En esta sección se encontrarán las variables no declaradas o no utilizadas que se encuentren en el proyecto, así como la línea, el procedimiento o función y el módulo en donde se localiza.

Una vez que se ha analizado el proyecto el usuario puede elegir guardarlo con otro nombre si así lo desea, cuando existan comentarios generales y/o en la sección de parámetros de entrada y salida.

Existe una ventana que muestra la versión y explica el funcionamiento de dicho desarrollo de software. Éste se encuentra en el menú Ayuda, y comando Acerca de …, aparece una ventana como la figura 6.

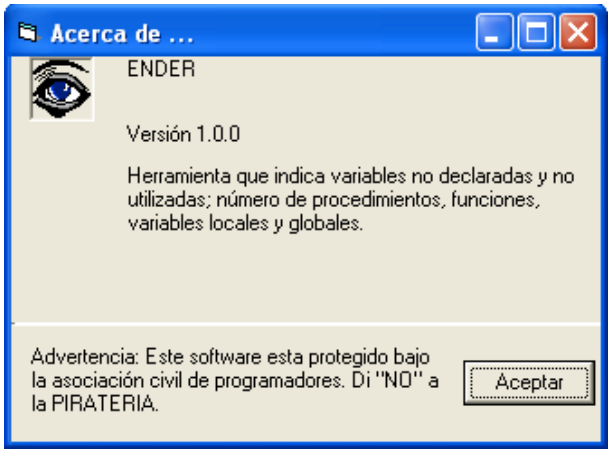


Figura 6. Ventana Acera de …

APÉNDICE L

ESQUEMA PARSEO

El esquema parseo de la figura 1 demuestra la forma en la que se realiza el análisis y se crea el árbol de nodos, así como el uso del algoritmo mencionado en el capítulo 2.

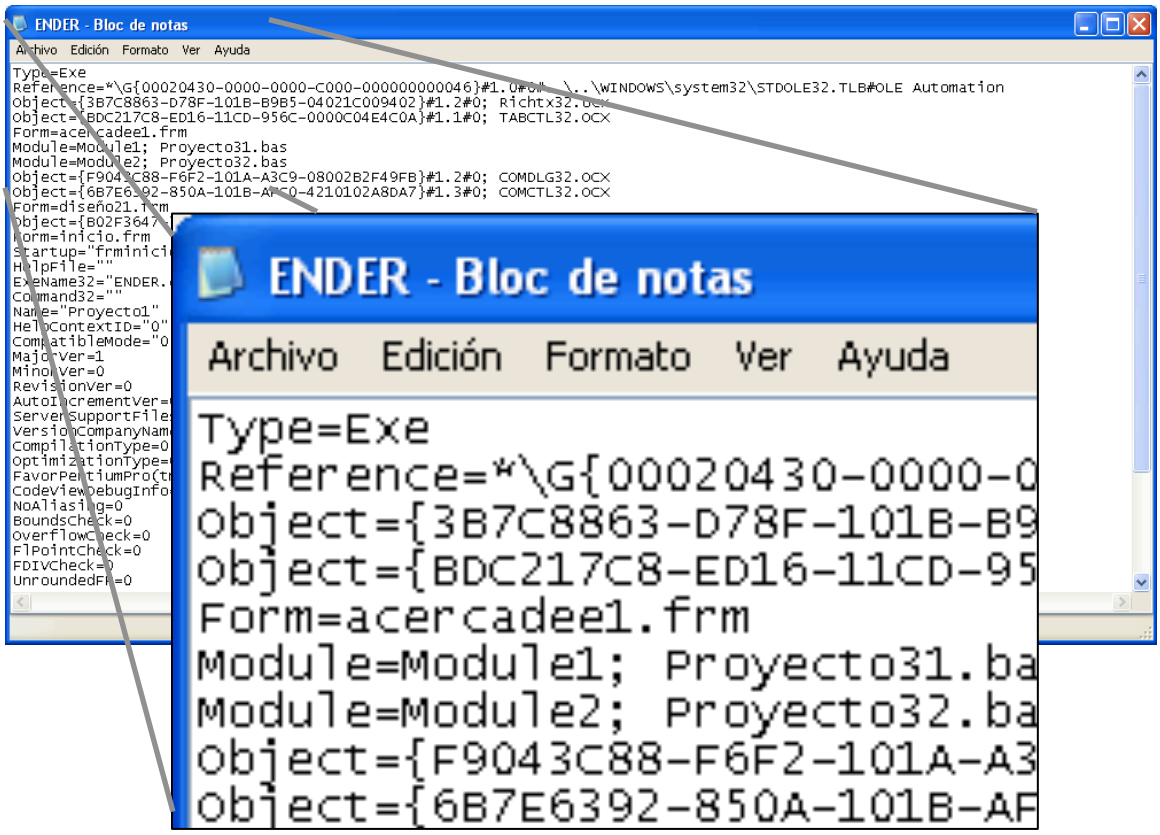


Figura 1. Esquema de parseo

A partir de la figura 1 se explicarán algunas líneas para expresar el funcionamiento del parseo que realiza ENDER.

En la primera línea de código del archivo ENDER.vbp se puede leer:

Type=Exe → Donde: “Type” es palabra reservada,

"=" es un caracter especial

"Exe" es una palabra reservada

En la segunda línea se puede apreciar que:

Reference=*\G{00020430…….} → Donde: "Reference" es una palabra reservada

"=" es un caracter especial

"*\G{00020430…….}" hace referencia a una librería

Para el caso de la tercera y cuarta línea:

Object={……} →Donde: "Object" es una palabra reservada

"=" es un caracter especial

"{}" hace referencia a objetos, de tipo "ocx"

La quinta fila:

Form=acercadee1.frm → Donde: "Form" es una palabra reservada

"=" es un caracter especial

"acercadee1.frm" es el nombre del formulario al que llama en el momento adecuado

Finalmente sexta y séptima fila:

Module=Module1; Proyecto31.bas → Donde: "Module" es palabra reservada

"=" es caracter especial

"Module1" indica que se trata de un archivo de tipo módulo que puede contener procedimientos, funciones, etc…

";"es caracter especial

"Proyecto31.bas" es el nombre del módulo, junto con su extensión

De esta forma se realiza el parseo en todos los programas o sistemas que se analicen en ENDER.

[i] Diseño y Gestión de Sistemas de Base de Datos. Angel Lucas Gómez. Paraninfo. España 1993.
[ii] Encyclopedia of Computer Science. Third Edition. Anthony Ralston, Edwin D. Reilly. IEEE PRESS. USA 1993. (Este Texto fue traducido por la Lic. Blanca Gabriela Mauries Ramírez.)
[iii] Dictionary of Computer Terms. Third Edition. Douglas Downing, Michael Covington. USA 1992. (Este Texto fue traducido por la Lic. Blanca Gabriela Mauries Ramírez.)
[iv] Microsoft ® Encarta ® Biblioteca de Consulta 2002. © 1993 – 2001 Microsoft Corporation.
[v] Introducción a las Computadoras. Robledo Sosa C. Ed. Tlamatini S.A. México 1986.
[vi] www.acatlan.unam.mx/Revistas/11/Tecnologia/ingsof.txt (en la actualidad la referencia electrónica no funciona)
[vii] UML y Patrones Introducción al análisis y diseño orientado a objetos. Craig Larman. Prentice Hall. México 1999.
[viii] www.dcc.uchile.cl/~psalinas/uml/introduccion.html