

C.F.G.S. DESARROLLO DE APLICACIONES WEB

MÓDULO: Entornos de Desarrollo

Unidad 2

Instalación y uso de Entornos de Desarrollo



Unión Europea
Fondo Social Europeo
"El FSE invierte en tu futuro"

Índice de contenido

1. Funciones de un Entorno de Desarrollo.....	3
1.1. Componentes.....	4
1.2. Soporte del Lenguaje de Programación.....	5
1.3. Actitudes en diferentes plataformas computacionales.....	5
1.4. Desarrollo rápido de aplicaciones (RAD).....	6
2. Componentes de un entorno de desarrollo.....	8
2.1. Editor de texto.....	8
2.2. Traductores.....	15
2.2.1. Compilador.....	15
2.2.2. Intérprete.....	16
2.3. Depurador.....	17
2.4. Refactorización.....	20
2.5. Control de versiones.....	21
2.6. Interfaz gráfica de usuario.....	21
3. Ejemplos de Entornos de desarrollo Integrados.....	23
4. Uso de herramientas CASE en el desarrollo de software.....	25
4.1. Herramienta CASE.....	25
4.2. Historia.....	25
4.3. Objetivos.....	25
4.4. Clasificación.....	26
4.5. Lenguaje Unificado de Modelado.....	28

1. Funciones de un Entorno de Desarrollo.

Un entorno de desarrollo integrado (*Integrated Development Environment, IDE*) es un programa compuesto por una serie de herramientas que utilizan los programadores para desarrollar código, es decir, para crear otros programas.

Esta herramienta puede estar pensada para su utilización con un único lenguaje de programación o bien puede dar cabida a varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, pensemos por ejemplo en la aplicación para oficina LibreOffice que consta de Writer como procesador de textos, Calc como hoja de cálculo, etc. En el caso de un IDE, suele constar de un editor de texto, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

El límite entre un entorno de desarrollo integrado y otras partes del entorno de desarrollo de software más amplio, no está bien definido. A veces se incluye un sistema de control de versiones y varias herramientas integradas para simplificar la construcción de la interfaz gráfica de usuario. Muchos entornos de desarrollo modernos también incluyen un navegador de clases, un inspector de objetos, y una jerarquía de clases diagrama, normalmente todo ello para su uso con el desarrollo de software orientado a objetos.

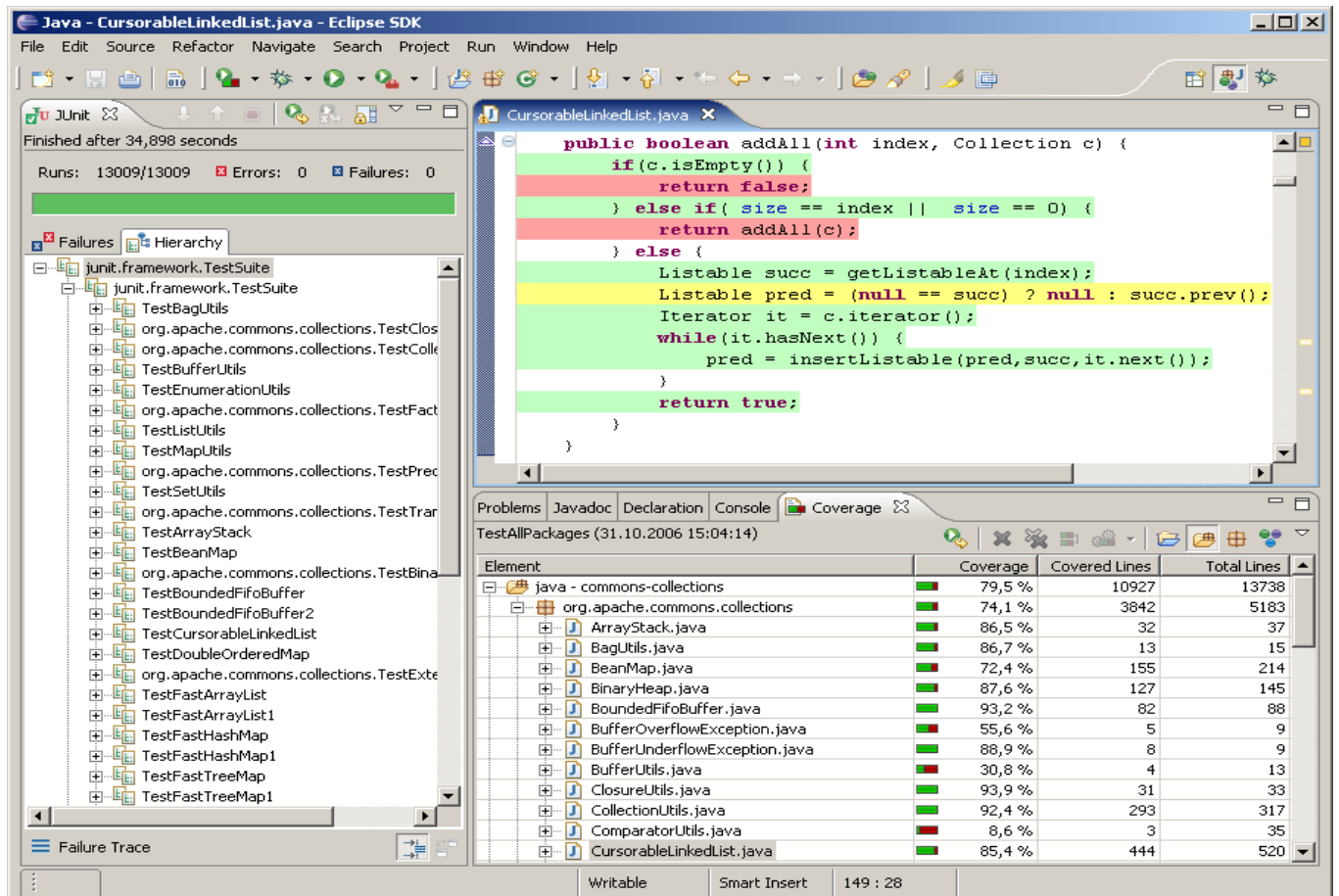
Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. El lenguaje Visual Basic, por ejemplo, puede ser usado dentro de las aplicaciones de Microsoft Office, lo que hace posible escribir sentencias Visual Basic en forma de macros para Microsoft Word desde el mismo Office.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, etc.

En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución (software que provee servicios para un programa en ejecución pero no es considerado en sí mismo como parte del sistema operativo) en donde se permite utilizar el lenguaje de programación en forma interactiva.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, al que mediante plugins se le puede añadir soporte para otros lenguajes (originalmente Java).

Al final, como programas que son, es cuestión de gustos escoger un IDE, la decisión suele recaer en el grupo de desarrollo, el cual tendrá en cuenta los lenguajes de programación soportados, el conjunto de herramientas disponibles y el tipo de software que van a programar. No es lo mismo hacer un programa para la web, que un driver para una impresora. Si es verdad, que cuantos más lenguajes y herramientas soporte el IDE, más recursos hardware se necesitarán. También la curva de aprendizaje, puede variar mucho cuanto más opciones tenga el IDE.



Para hacer un programa no es necesario utilizar un IDE, se puede dar el caso de programadores que no lo utilicen, escogiendo personalmente herramientas independientes para cada tarea (editor, compilador, etc); el inconveniente es que las herramientas por separado no están tan integradas como en un IDE.

1.1. Componentes.

Un IDE, dependiendo del fabricante, puede incluir más o menos herramientas, muchas de ellas pueden ser incluidas posteriormente mediante plugins.

Las herramientas que normalmente componen un entorno de desarrollo integrado son las siguientes:

- **Un editor de texto:** Permite escribir el programa. No confundir un editor con un procesador de textos como el Word. El bloc de notas es un editor de textos.
- **Un traductor (compilador o intérprete):** Permite obtener el código máquina, el ejecutable.
- **Un depurador:** Ayuda a encontrar y resolver errores en los programas.
- **Sistema de control de versiones:** Permite controlar los cambios que se van realizando en el software, muy útil cuando trabajan muchos programadores sobre el mismo código.
- **Constructor de interfaces gráficas de usuario (GUI):** Permite crear la parte visual del programa, los botones, las ventanas, etc.

- **Creación de la documentación.**

1.2. Soporte del Lenguaje de Programación.

Algunos IDE's, famosos como Eclipse o NetBeans, soportan múltiples lenguajes, aunque son más usados con Java. Otros como MonoDevelop, están más especializados en C# y otros lenguajes “.NET”. Por tanto, en principio, un IDE puede soportar uno o varios lenguajes de programación.

Hoy en día muchos de los IDE proporcionan un marco de trabajo adecuado para la mayoría de los lenguajes de programación existentes en el mercado, como C, C++, C#, Java, Python y Visual Basic entre otros.

El soporte para lenguajes alternativos es a menudo proporcionado mediante plugins. Por ejemplo, Eclipse y NetBeans tiene plugins para C, C++ , Ada , Perl , Python , Ruby y PHP , entre otros lenguajes.

1.3. Actitudes en diferentes plataformas computacionales.

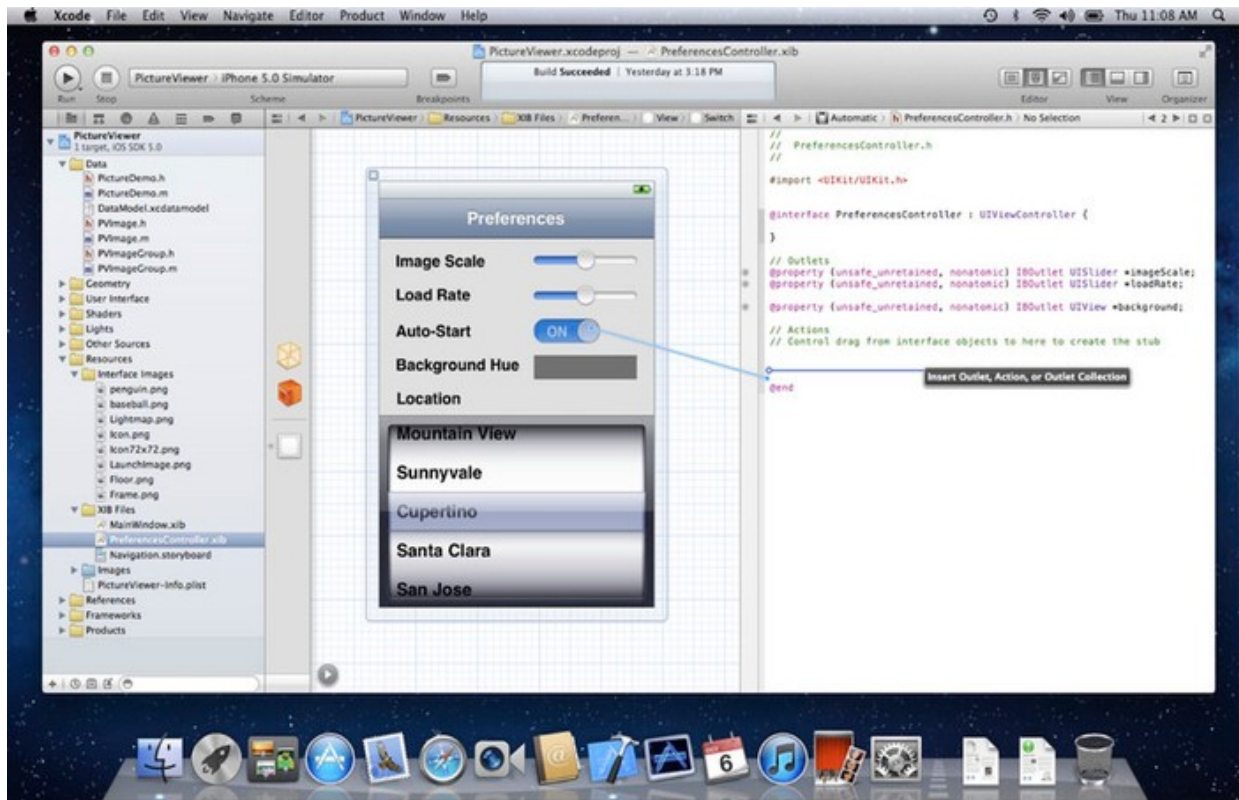
Muchos de los programadores de entornos Unix y derivados como Linux, argumentan que las herramientas de la línea de comandos POSIX (Portable Operating System Interface, X de UNIX) constituyen también un IDE, aunque con un estilo diferente de la interfaz. Por ejemplo, muchos programadores de Unix/Linux utilizan Emacs o Vim como entorno de desarrollo. Vim dispone de un modo "editar, compilar, corregir". De la misma forma que los entornos de desarrollo integrados, pueden editar el código fuente además llamar a un compilador externo, e interpretar sus resultados.

[illegible]

En las distintas plataformas de Microsoft Windows, las herramientas de línea de comando para el desarrollo de aplicaciones están cada vez más en desuso. La realidad es que hay muchas soluciones comerciales y no comerciales para los IDE, sin embargo cada uno tiene un diseño diferente que comúnmente crean incompatibilidades. La mayoría de los principales proveedores de compiladores para Windows siguen ofreciendo copias gratuitas de sus herramientas para la línea de comandos.

Por otro lado, las herramientas del software libre GNU, como el compilador gcc, o el debugger gdb están disponibles en muchas plataformas, incluyendo Windows.

Actualmente los programadores de Mac OS también tienen varios IDE's para programar, incluyendo IDE's nativos como Xcode y herramientas de código abierto, tales como Eclipse y NetBeans.



Algunos IDE de código abierto, tales como IDE Code::Blocks, Eclipse, Lazarus, KDevelop y Netbeans, que a su vez se desarrollan utilizando un lenguaje multiplataforma (por ejemplo, Free Pascal o Java), se ejecutan en múltiples plataformas incluyendo **Windows, GNU / Linux, y Mac OS**.

1.4. Desarrollo rápido de aplicaciones (RAD).

El **desarrollo rápido de aplicaciones** o **RAD** (*Rapid Application Development*) es un proceso de desarrollo de software que permite construir aplicaciones utilizables en poco tiempo. El método comprende el desarrollo interactivo, la construcción de prototipos y el uso de herramientas CASE (Computer Aided Software Engineering).

Tradicionalmente, el desarrollo rápido de aplicaciones tiende a englobar también la usabilidad, utilidad y la rapidez de ejecución.

Hoy en día se suele utilizar para referirnos al desarrollo rápido de interfaces gráficas de usuario tales como Glade, o entornos de desarrollo integrado completos. Algunas de las plataformas más conocidas son Visual Studio, Lazarus, Delphi, Foxpro, Anjuta, Game Maker, Velneo o Clarion. En el área de la autoría multimedia, software como Neosoft Neobook proveen plataformas de desarrollo rápido de aplicaciones, dentro de ciertos límites.

El nombre RAD también puede ser utilizado como estrategia comercial para vender un producto. Así describe RAD la empresa Microsoft:

“En los primeros días de la programación, finalizar un programa sencillo llevaba días o incluso semanas. Cuando se presentó por primera vez Visual Basic en 1991, revolucionó la programación; ya no era necesario escribir código para crear una interfaz de usuario ni había que preocuparse por la administración de memoria. Esta nueva manera de programar se denominó desarrollo rápido de aplicaciones o RAD (Rapid Application Development)”.

Si tan bueno es un RAD... ¡todo el mundo debería utilizarlo! Pero la realidad de la programación actual es muy compleja y no es tan fácil aplicar esta tecnología a cualquier tipo de programa. Un paso esencial es llegar a un acuerdo desde el principio entre clientes y desarrolladores. Esta tecnología exige además equipos de programación disciplinados a la hora de cumplir tiempos y costos. Usada debidamente, se obtienen buenos resultados, ahorrando costes y tiempo.

Otro ejemplo de RAD es Lazarus, aplicación basada en el lenguaje de programación Object Pascal, disponible para los sistemas operativos Windows, GNU/Linux y Mac OS X. Se trata de una alternativa libre y gratuita a Delphi, desarrollada como proyecto de software libre a partir de Free Pascal. La web y la mayoría de la documentación están en inglés, pero el entorno de desarrollo (IDE) está traducido al español en gran parte.



2. Componentes de un entorno de desarrollo.

2.1. Editor de texto.

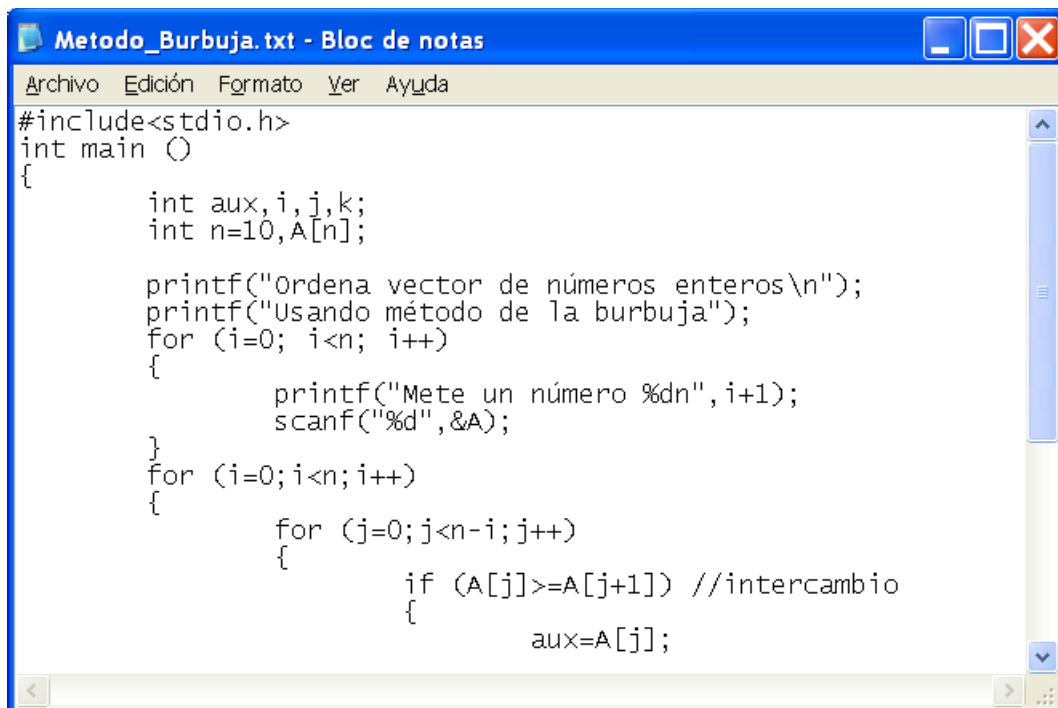
Un **editor de texto** es un programa que permite crear y modificar archivos compuestos únicamente por texto sin formato, conocidos comúnmente como archivos de texto plano o sin formato.

El programa lee el archivo e interpreta los Bytes leídos según el código de caracteres (ASCII o UTF-8) que usa el editor.

El editor de texto más popular es el “Bloc de notas” de Windows, aunque en este caso concreto, no se le puede considerar una buena herramienta para escribir programas, de hecho no está pensado para esta misión. Pero si se podría utilizar. Hay editores de texto que están pensados para programar, mientras que otros son más genéricos.

Un editor de texto suele venir incluido con el sistema operativo. Se usa cuando se desea crear algún tipo de nota, modificar archivos de configuración, scripts o el código fuente de algún programa sencillo.

A pesar de que un procesador de texto (Word, LibreOffice Writer, etc) también permite escribir textos, estos además del propio texto añaden otra información como propiedades de la página, tipos de letra, tamaño, color, etc. por lo que no son adecuados para escribir programas. Es verdad, que suelen tener una opción para guardar el documento en texto plano, pero aún así no son una herramienta especializada para el desarrollo de software y consumen más recursos de los necesarios para la tarea encomendada.



```
#include<stdio.h>
int main ()
{
    int aux,i,j,k;
    int n=10,A[n];

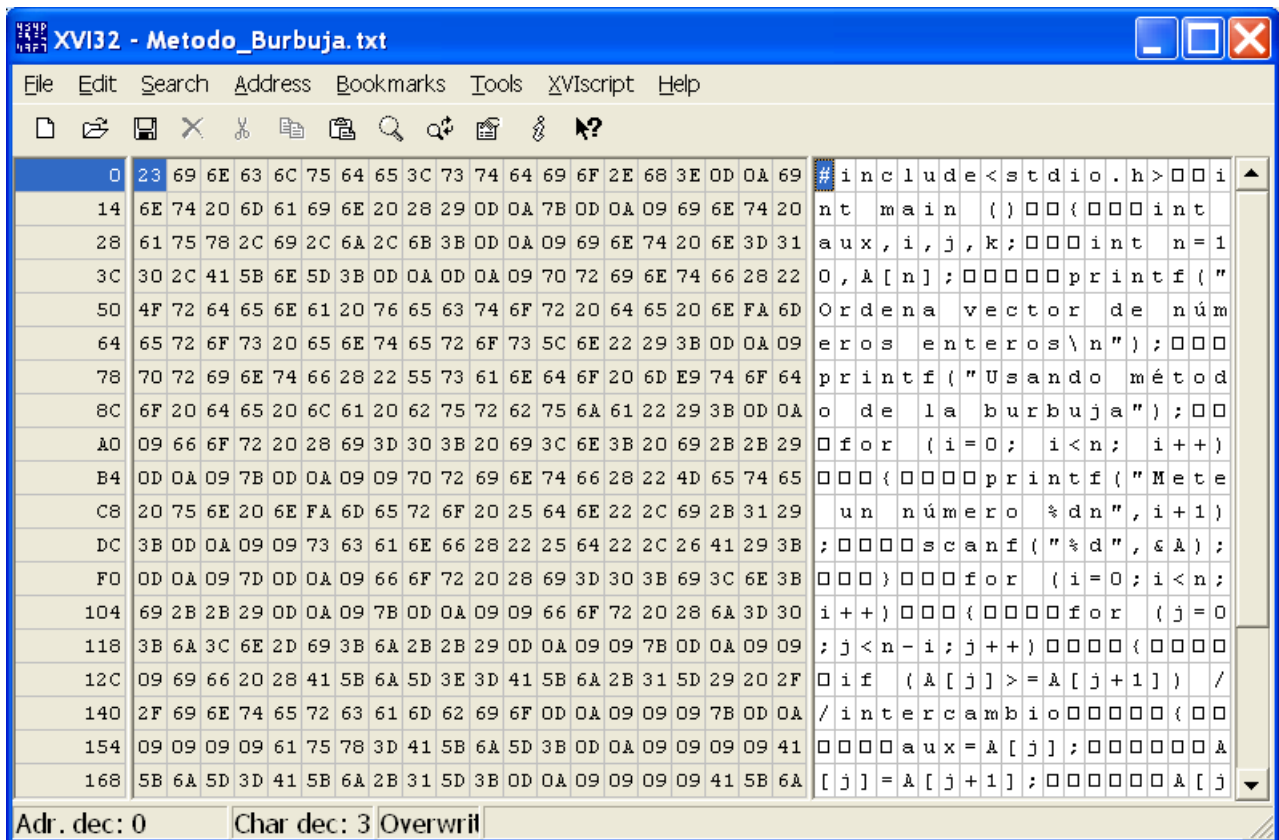
    printf("Ordena vector de números enteros\n");
    printf("Usando método de la burbuja");
    for (i=0; i<n; i++)
    {
        printf("Mete un número %dn",i+1);
        scanf("%d",&A);
    }
    for (i=0;i<n;i++)
    {
        for (j=0;j<n-i;j++)
        {
            if (A[j]>=A[j+1]) //intercambio
            {
                aux=A[j];
```

Si analizamos el contenido en formato numérico de un archivo de texto plano encontraremos únicamente los códigos ASCII correspondientes a cada una de las letras escritas y nada más.

El archivo creado por un editor de texto incluye por convención en Microsoft Windows la extensión ".txt", aunque pueda ser cambiada a cualquier otra con posterioridad. Tanto Unix como Linux dan al usuario total libertad en la denominación de sus archivos. Aunque realmente la extensión del nombre de un fichero no implica nada, suele dar una pista sobre el tipo del contenido del fichero, por tanto, un programa en C, suele tener por extensión ".c", uno en Pascal, ".pas", uno en C++, ".cpp", uno en java, ".java" etc.

A pesar de la sencillez de los archivos de texto, puede aparecer alguna sorpresa al trasladar archivos de texto de un sistema operativo a otro, ya que se debe considerar que existen al menos dos convenciones diferentes para señalar el fin de cada línea escrita. En Unix y Linux usan solo "salto de línea" (LF) (un carácter, código ASCII 10 o 0A en hexadecimal), en cambio Microsoft Windows usa "retorno de carro" (CR, ASCII 13 o 0D en hexadecimal) más "salto de línea" LF (carriage return + line feed) (dos caracteres).

En la siguiente imagen, utilizando un programa editor hexadecimal, podemos observar el contenido de un fichero que contiene un programa escrito en C. En la parte derecha de la ventana, al comienzo vemos que aparecen los caracteres "#include<stdio.h>", en la parte izquierda, el código ASCII de cada carácter en hexadecimal: 23, 69, 6E, 63, 6C, 75, 64, 65, Los símbolos un poco raros (parecen cuadrados) se corresponden con caracteres no imprimibles, en este caso, son los códigos 0D, 0A, que indican "retorno de carro" y "salto de línea".

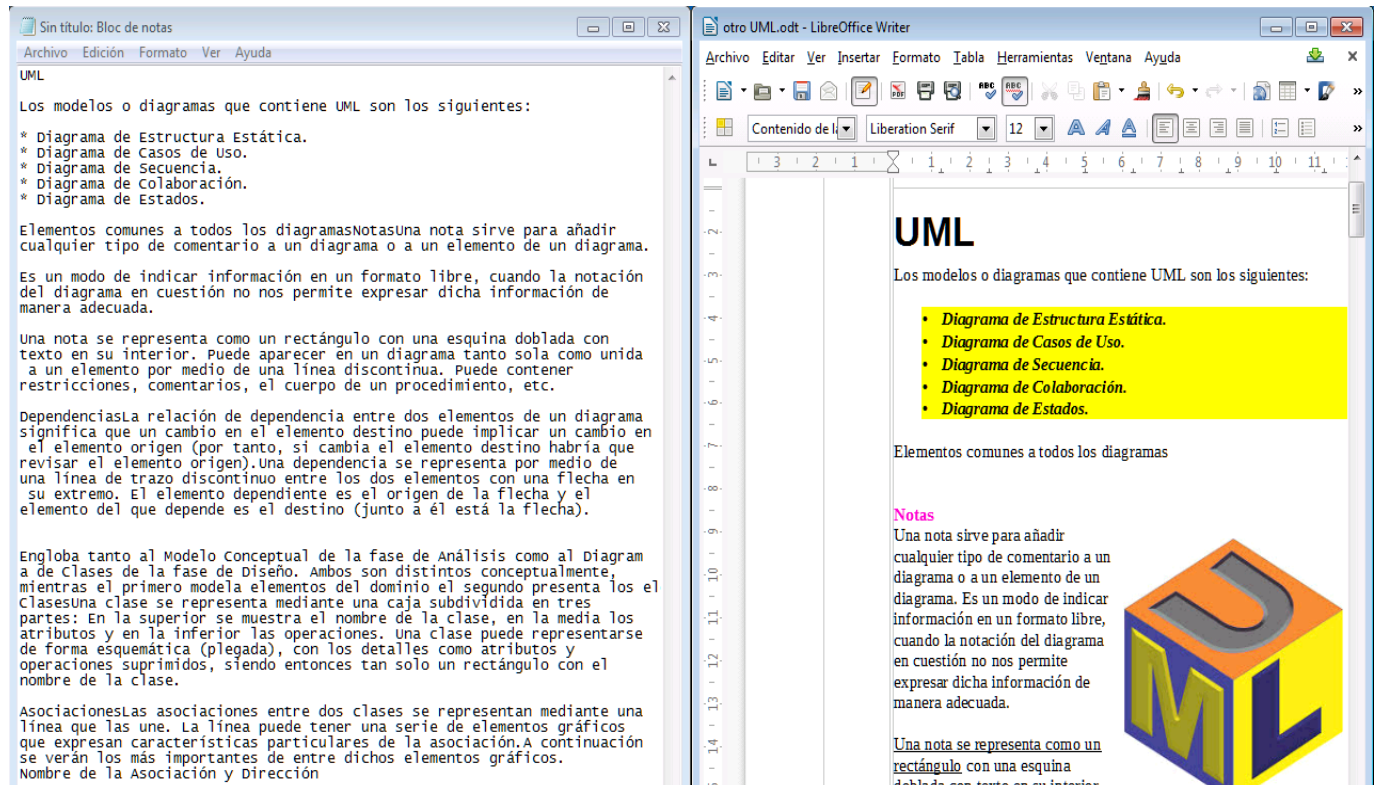


Los editores de textos se distinguen de los procesadores de texto en que se usan para escribir solo texto, sin formato y sin imágenes, es decir **sin diagramación**.

- El texto plano es representado en un editor mostrando todos los caracteres presentes en el archivo. No confundir con que algunos editores de texto para programación muestren coloreado de sintaxis, o que se pueda ver la letra en distintos tamaños o fuentes, ya que en este caso los colores y tamaños los pone el propio editor, mientras que en el fichero donde está almacenado el texto solo están los caracteres sin más. Los únicos caracteres de formateo son: salto de línea, tabulación horizontal y retorno de carro. Los códigos de caracteres más conocido son ASCII y UTF.
- Los documentos creados con procesador de texto contienen información extra (tamaño de página, gráficos, colores, ...) y caracteres de control para darle al texto un formato o diagramación personalizada. En estos casos cada carácter podría tener un aspecto diferente (tamaño, color, fuente,...). Hoy en día un procesador de textos es capaz de manejar incluso gráficos y otros tipos de objetos, convirtiéndose prácticamente en programas de autoedición como los manejados en el campo del periodismo. Los procesadores de texto pueden en la mayoría de los casos tener una opción que les permita almacenar el texto sin formato en un archivo de texto plano, pero se le debe ordenar explícitamente que se desea esa opción. Por tanto, se puede escribir un programa con un procesador de textos, pero no es la herramienta adecuada, sería "como matar moscas a cañonazos".

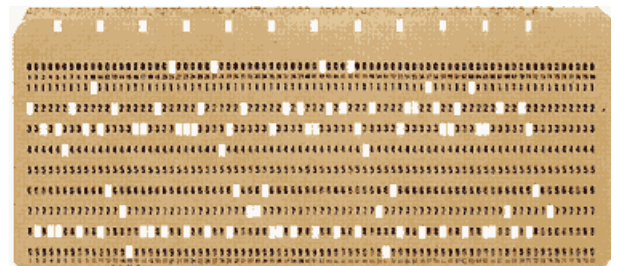
En la siguiente imagen podemos ver la gran diferencia entre un texto en el editor de textos "bloc de notas"

y en el procesador de textos “LibreOffice Writer”. El procesador tiene muchas opciones de diagramación, en el editor sólo se ven caracteres y nada más.



■ Historia de la edición de programas

Antes que existieran los teclados, ratones y monitores actuales, los datos y el código fuente de los programas debía ser entregado a la máquina por medio de tarjetas o cintas perforadas. El programa, un fajo de tarjetas, era leído por un lector de tarjetas perforadas. Cada carácter era representado por una o más perforaciones en una columna de 8 o 10 posibles perforaciones. No más de 80 caracteres podían ser representados en una tarjeta.



Con los teletipos aparecieron los primeros editores de texto que en un comienzo solo mostraban una línea del texto. Mandatos especiales movían el "cursor" en las cuatro direcciones.



El siguiente paso se dio con la aparición del monitor de tubo de rayos catódicos que permitió la edición a "pantalla completa" es decir mostrando varias líneas a la vez, cada línea tenía 80 columnas. El aumento de productividad fue enorme, por la rapidez y simplicidad del mecanismo.

Uno de los primeros editores de texto fue O26, escrito para la consola del operador de la serie de computadoras CDC 6000 a mediados de la década de 1960.

Otro precursor fue el Vi o Vim, escrito en la década de 1970, que aún es el editor estándar para muchos de los sistemas operativos basados en Unix y Linux.



■ Tipos de editores de texto

Hay una gran variedad de editores de texto. Algunos son de uso general, mientras que otros están diseñados para programar en uno o varios lenguajes. Algunos son muy sencillos, mientras que otros tienen implementadas gran cantidad de funciones. En general, los editores de texto son programas que necesitan menos recursos que un procesador de textos.

El editor de texto debe ser considerado como una herramienta de trabajo del programador o del administrador de la máquina. Como herramienta, permite realizar ciertos trabajos, pero también requiere de aprendizaje para que el usuario conozca y obtenga destreza en su uso. La llamada *curva de aprendizaje* es una representación de la destreza adquirida a lo largo del tiempo de aprendizaje.

Un editor puede ofrecer más o menos funciones. Cuantas más funciones tenga, lo normal es que sea más largo el tiempo que se tarda en manejar. Si su curva de aprendizaje es muy inclinada (es difícil de aprender), puede desanimar al programador que terminará dejándolo. Puede que un editor tenga una curva de aprendizaje muy empinada y corta, pero si no ofrece muchas funciones el usuario le reemplazará por otro más productivo. Es decir la elección del editor más apropiado depende de varios factores, alguno de ellos muy subjetivos. Esta coyuntura de intereses ha dado lugar a largas discusiones sobre la respuesta a la pregunta "¿cual es el mejor editor de texto?".

Otro factor a tener en cuenta es la plataforma en la que desarrollamos. Hoy en día muchos editores originalmente salidos de Unix o Linux han sido portados a otros sistemas operativos, lo que permite trabajar en otros sistemas sin tener que aprender el uso de otro editor.

Los editores para programadores profesionales deben ser capaces de leer archivos de gran extensión, mayor que la capacidad de la memoria RAM de la máquina y también arrancar rápidamente, ya que el tiempo de espera disminuye la concentración y disminuye de por sí la productividad.

Algunos editores de texto más profesionales son capaces de reconocer algún lenguaje de programación, lo que permite automatizar engorrosos o repetidos procedimientos a realizar en el texto. Por ejemplo, el editor Emacs



puede ser adaptado a las necesidades del usuario, incluso las combinaciones de teclas para ejecutar funciones pueden ser adaptadas. Además es programable en Lisp.

Otra función muy interesante de los editores de texto actuales especializados en un lenguaje de programación, es la inclusión de coloreado de sintaxis y funciones que ofrecen al usuario auto completar palabras a medida que se escribe, por ejemplo, suele ser habitual, empezar a escribir el nombre de una función y que aparezca a modo de ayuda la sintaxis de la misma.

Algunas funciones especiales incluidas frecuentemente en editores de texto especializados en programación son las siguientes:

- Reconocimiento de uno o varios lenguajes de programación, lo que permite el coloreado de sintaxis, completación de palabras, ayuda propia del lenguaje, etc.
- Regiones plegables. A veces no todo el texto es relevante para el usuario. Con este tipo de editores ciertas regiones con texto irrelevante pueden ser plegadas, escondidas, mostrando al usuario solo lo importante del texto.
- Editar al unísono varios documentos, mostrar diferencias entre textos, macros.

■ Funciones típicas de un editor de texto

- Marcar región

Es la función que marca, visualmente o no, una parte del texto para ser elaborada con otras funciones. La región puede contener varias líneas del texto (región horizontal) o bien varias columnas adyacentes del texto (región vertical).

- Búsqueda y reemplazo.

El proceso de búsqueda de una palabra o una cadena de caracteres, en un texto plano y su reemplazo por otra. Existen diferentes métodos: global, por región, reemplazo automático, reemplazo con confirmación, búsqueda de texto o búsqueda de una Expresión regular.

- Copiar, cortar y pegar.

Sirve para copiar, trasladar o borrar una región marcada.

- Formatear.

Los editores de texto permiten automatizar las únicas funciones de formateo que utilizan: quebrar la línea, indentar, formatear comentarios o formatear listas.

- Deshacer y rehacer.

Consiste en que el programa editor va almacenando cada una de las operaciones hechas por el usuario hasta un número configurable. Si el usuario se arrepiente de algún cambio, por muy anterior que sea, el editor le permite revertir todos los cambios hechos hasta el número configurado. Rehacer es por consiguiente, revertir algo revertido.

- Importar

Agregar o insertar el contenido de un archivo en el archivo que se está editando. Algunos editores permiten insertar la salida o respuesta a un programa cualquiera ejecutado en la [Línea de comandos](#) al archivo que se está editando.

- Filtros

Algunos editores de texto permiten hacer pasar las líneas del texto o de una región por algún programa para modificarlas u ordenarlas. Por ejemplo, para ordenar alfabéticamente una lista de nombres o sacar un promedio de una lista de números.

- Acceso remoto

Un editor para trabajar en la administración de una red de computadoras debe ofrecer la funcionalidad de editar archivos en máquinas remotas, ya sea por medio de [ftp](#), [ssh](#) o algún otro [Protocolo de red](#). [Emacs](#) lo puede hacer mediante el [Plugin tramp](#) (ampliamente configurable con ssh, ftp, scp, sftp, etc), Ultraedit, del ambiente Windows, lo hace mediante ftp.

■ Ejemplos de editores de texto

- **[Bloc de notas](#)**, editor integrado en Windows, también conocido como Notepad (en inglés).
- **[Emacs](#)**, otro editor muy común en [Unix](#).
- **[Gedit](#)**, editor [libre](#) que se distribuye junto con [GNOME](#) para sistemas tipo Unix. También disponible para Windows.
- **[jEdit](#)**, editor popular multiplataforma.
- **[Kate](#)**, un moderno editor para Unix.
- **[Notepad++](#)**, editor de código fuente para Windows.
- **[Vi](#)**, editor muy común en Unix.
- Sublime Text. Para código, markup y prosa.
- En la nube: <https://write-box.appspot.com/>, <http://notepad.cc>, <http://writer.bighugelabs.com/>

```

fileX.cpp (~) - GVIM
File Edit Tools Syntax Buffers Window Help

}

// Execute shell script

in = popen(sCommandToExecute.c_str(), "r"); // execute script

if (in == (FILE *) NULL) /* Check pipe failure */
{
    cout << "[popen failure] Pipe creation failed: Trigger script not ex
ecuted.";
    return 0;
}
pclose(in);
}

if( !SysConfig.getPageType().compare( "HtmlExtra" ) )
{
    cout << h2("Trigger Complete") << endl;
}
else if( !SysConfig.getPageType().compare( "HtmlMinimal" ) )
{
    cout << "Trigger Complete" << endl;
}
    
```

775,1 28%

En la siguiente dirección hay un análisis de los editores más usados:

<http://bitelia.com/2013/10/mejores-editores-de-texto-para-desarrolladores>



Tira de Linux Hispano

by danigm

2.2. Traductores

A la hora de transformar el código escrito en un lenguaje de programación al código máquina (el único realmente ejecutable por la máquina) se necesita un software denominado traductor.

La forma de traducir se puede realizar de 2 formas diferentes, lo que lleva a dos tipos de software de traducción: compiladores e interpretes.

Un lenguaje de programación en si no tiene por que ser compilado o interpretado, aunque es verdad que muchos de ellos solo tienen un tipo de traductor disponible, pero nada tiene que ver con el propio lenguaje.

2.2.1. Compilador.

Un **compilador** es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser un código intermedio (bytecode), o simplemente texto. Este proceso de traducción se conoce como **compilación**.

De esta manera un programador puede diseñar un programa en un lenguaje mucho más cercano a como piensa un ser humano, para luego *compilarlo* a un programa manejable por una computadora.

El nombre de “compilador” se utiliza principalmente para programas que traducen el código fuente de un lenguaje de programación de alto nivel a un lenguaje de bajo nivel (por ejemplo, en lenguaje ensamblador o código de máquina).

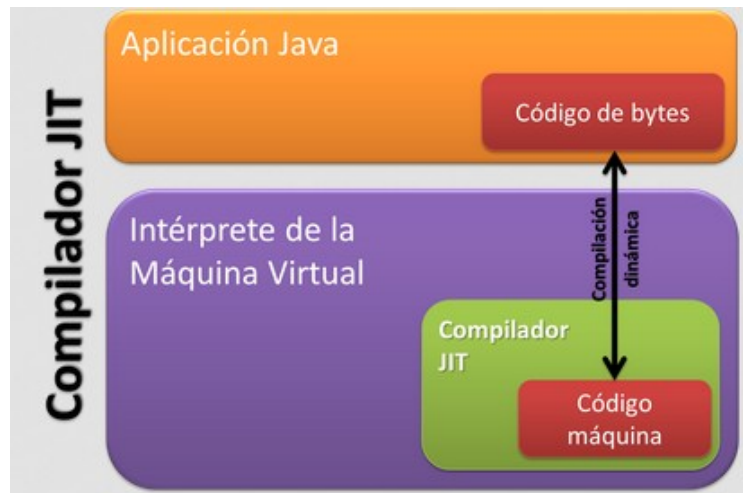
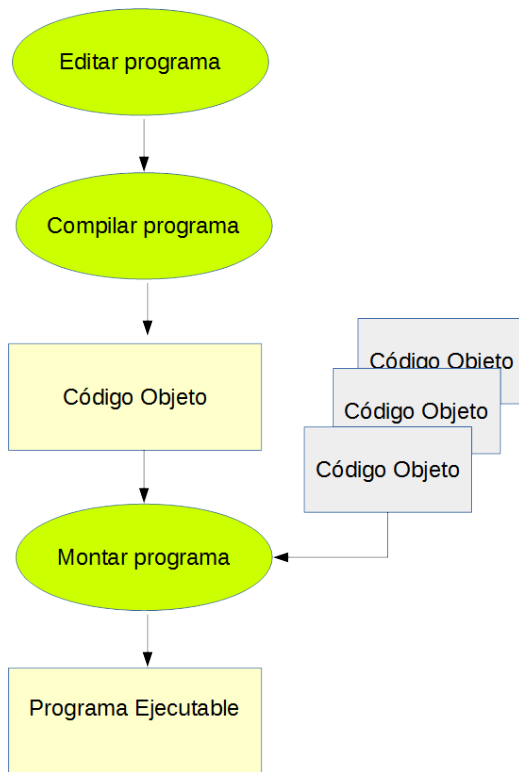
Un compilador es probable que realice muchas o todas de las siguientes operaciones: análisis léxico, pre-procesamiento, el análisis semántico (dirigida por la sintaxis de traducción), la generación de código y la optimización del código.

■ Compilador cruzado

Un **compilador cruzado** es un compilador capaz de crear código ejecutable para otra plataforma distinta a aquella en la que él se ejecuta. Esta herramienta es útil cuando quiere compilarse código para una plataforma a la que no se tiene acceso, o cuando es incómodo o imposible compilar en dicha plataforma como en el caso de las consolas de videojuegos, la programación para móviles o los sistemas embebidos.



Un ejemplo de un compilador con estas posibilidades es el NASM, que puede ensamblar, entre otros formatos, ELF (para sistemas UNIX) y COM (para DOS).



2.2.2. Intérprete.

Es un programa informático capaz de analizar y **ejecutar** otros programas escritos normalmente en un lenguaje de programación de alto nivel. Los intérpretes se diferencian de los compiladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción. Un intérprete traduce y ejecuta línea por línea. El compilador traduce de una pasada a código máquina y ya no se necesita más ni el compilador ni el código fuente. El usuario ya es libre de ejecutar el programa cuantas veces quiera. El intérprete siempre es necesario junto al código fuente para poder ejecutar el programa.

Usando un intérprete, un mismo archivo fuente produce resultados iguales en sistemas sumamente diferentes (ej. un PC y una videoconsola). Usando un compilador, un archivo fuente puede producir resultados iguales solo si es compilado a distintos ejecutables específicos para cada sistema, ya que no hay que olvidar que el resultado de la compilación es código máquina, y este es diferente en cada sistema.



Los programas interpretados suelen ser más lentos que los compilados debido a la necesidad de traducir el programa línea por línea mientras se ejecuta (se está ejecutando por un lado el intérprete y por otro el propio programa), pero a cambio son más flexibles como entornos de programación y depuración (lo que se traduce, por ejemplo, en una mayor facilidad para reemplazar partes enteras del programa o añadir

módulos completamente nuevos). Además permiten ofrecer al programa interpretado un entorno no dependiente de la máquina donde se ejecuta, aunque si dependemos del propio intérprete.

Hay software que permite interpretar o compilar el código fuente original a una forma intermedia más compacta y después traducir eso al código de máquina (ej. Perl, Python, MATLAB, y Ruby). Algunos aceptan los archivos fuente guardados en esta representación intermedia (ej. Python, UCSD Pascal y Java).

En java, el código fuente del programa se compila a un código intermedio diferente al código máquina de la máquina real denominado bytecode, el cual es ejecutado en la “máquina virtual java (JVM)”. El bytecode Java es el tipo de instrucciones que la máquina virtual Java ejecuta. Usualmente es el resultado de utilizar un compilador del lenguaje de programación Java (como javac), pero puede ser generado desde otros lenguajes. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto.

Comparando su actuación con la de un ser humano, un compilador equivale a un traductor profesional que, a partir de un texto, prepara otro independiente traducido a otra lengua, mientras que un intérprete corresponde al intérprete humano, que traduce de viva voz las palabras que oye, sin dejar constancia por escrito.

En la actualidad, uno de los entornos más comunes de uso de los intérpretes informáticos es Internet, debido a la posibilidad que estos tienen de ejecutarse independientemente de la plataforma.

2.3. Depurador.

Un **depurador (debugger)**, es un programa usado para probar y depurar (eliminar) los errores de otros programas. No estamos hablando de errores de compilación debidos a escritura incorrecta del programa en el lenguaje elegido, estamos hablando de errores de lógica, es decir, cuando el programa se ejecuta pero los resultados no son los esperados.

Otra definición, un depurador es una aplicación que permite correr otros programas, permitiendo al usuario ejercer cierto control sobre los mismos a medida que estos se ejecutan. Además permiten examinar el estado del sistema (variables, registros, banderas, etc.) a medida que se van ejecutando las instrucciones. El propósito final de un depurador consiste en permitir al usuario observar y comprender lo que ocurre "dentro" de un programa mientras el mismo es ejecutado.

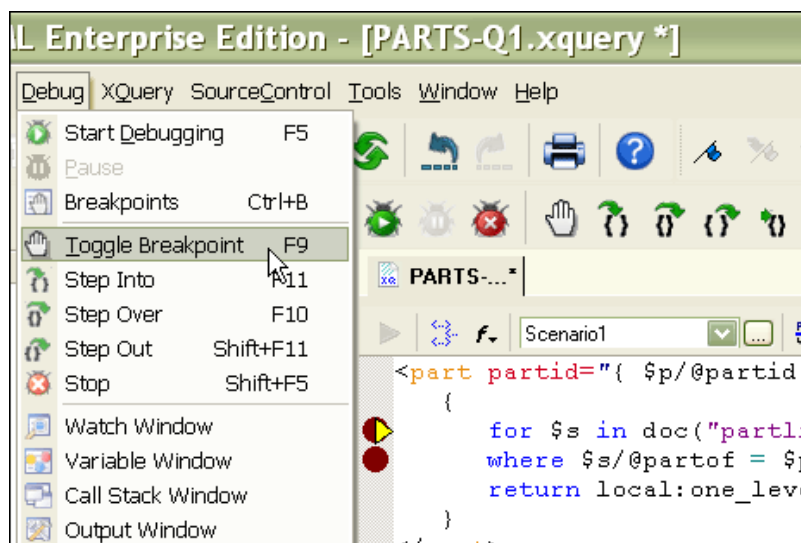
El código a ser examinado puede alternativamente estar corriendo en un simulador de conjunto de instrucciones (ISS), una técnica que permite gran potencia en su capacidad de detenerse cuando son encontradas condiciones específicas pero será típicamente algo más lento que ejecutando el código directamente en el apropiado (o el mismo) procesador. Algunos depuradores ofrecen dos modos de operación - la simulación parcial o completa,- para limitar este impacto.

Los programas depuradores pueden ser de 2 tipos:

- **Depurador de nivel de fuente o depurador simbólico**, común en entornos de desarrollo integrados. Contamos con el código fuente y al mandar depurar el programa se ejecuta pero podremos observar su comportamiento, por tanto muestra la posición en el código original. Si el programa "rompe" o alcanza una condición predefinida, el depurador nos puede mostrar la posición en el código original. Además podemos poner puntos de parada, modificar o consultar variables, ejecutar línea a línea, saltarse instrucciones, etc, estas y otras funciones dependen de la calidad del depurador.
- **Depurador de bajo nivel o un depurador de lenguaje de máquina**, se usan cuando no se tiene el código fuente, por lo tanto sólo contamos con el ejecutable. En estos casos el depurador muestra la línea en el fuente desensamblado (a menos que también tenga acceso en línea al código fuente original y pueda exhibir la sección apropiada del código del ensamblador o del compilador). Al trabajar en ensamblador, es más difícil encontrar los fallos o las modificaciones que se quieran realizar. Este tipo de depuradores que son más difíciles de manejar, son utilizados por programadores de antivirus para analizar las muestras y por programadores que quieren quitar protecciones a juegos o cambiar el comportamiento de los mismos (por ejemplo, poner vidas infinitas).

Un programa "rompe" o "se estrella" o "se cuelga" cuando no puede continuar, normalmente debido a un error de programación. Por ejemplo, el programa pudo haber intentado tener acceso a memoria protegida, o ha intentado realizar una división entre 0, etc.

En general los programas depuradores ofrecen funciones sofisticadas tales como correr un programa paso a paso (instrucción a instrucción) o ejecutar hasta cierto punto y parar. De esta forma se puede comprobar el estado de las variables en memoria o si se produjo algún evento. Los puntos de parada también se les conoce como un "breakpoint" o punto de ruptura.



Algunos depuradores también tienen la capacidad de modificar el estado del programa, por ejemplo, cambiar valores en variables o saltarse instrucciones mientras que está corriendo.

La importancia de un buen depurador no es exagerada. De hecho, la existencia y la calidad de tal herramienta para un lenguaje y una plataforma dada a menudo puede ser el factor de decisión en su uso, incluso si otro lenguaje/plataforma es más adecuado para la tarea.

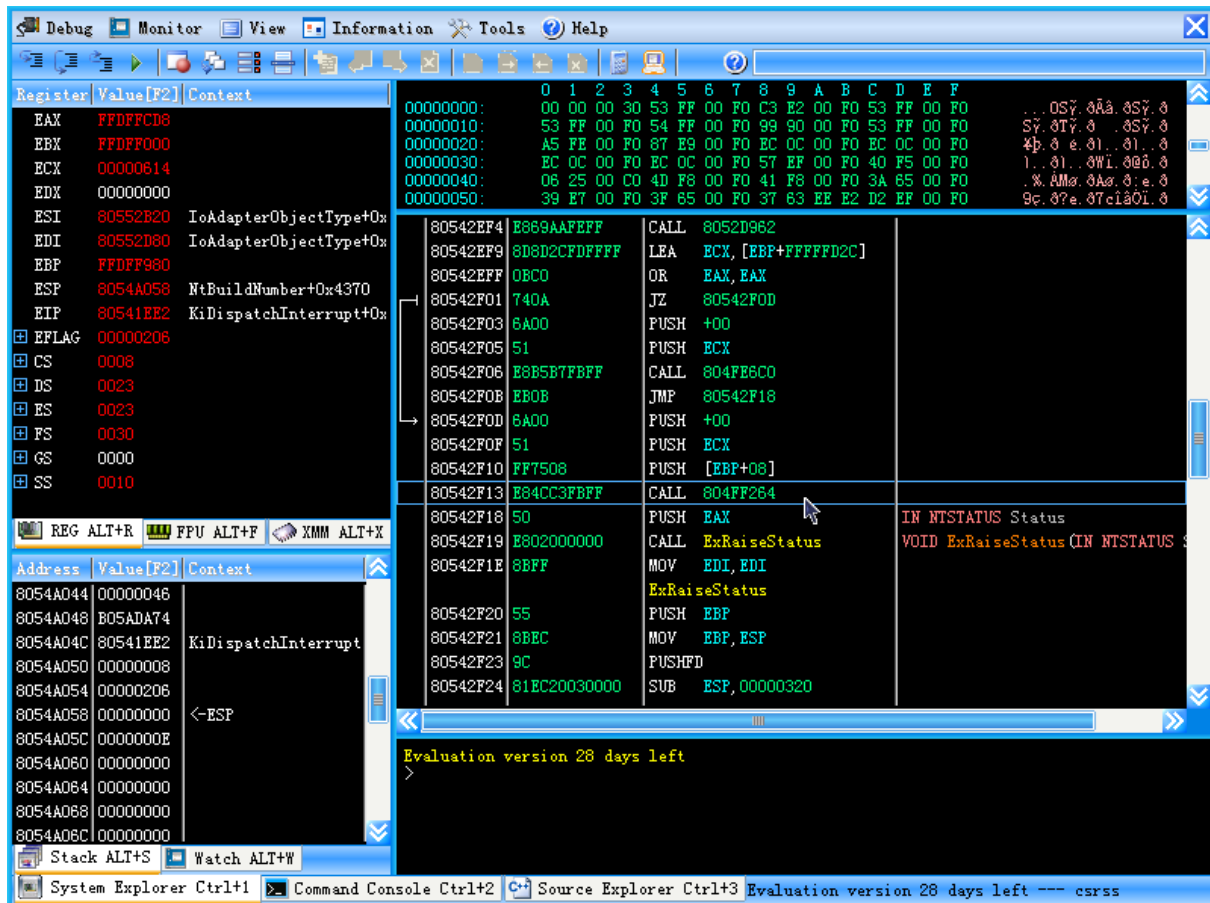
Sin embargo, el software puede comportarse diferentemente corriendo bajo un depurador debido a los cambios inevitables que la presencia de un depurador hará a la temporalización interna del programa. Como resultado, incluso con una buena herramienta de depuración, es a menudo muy difícil rastrear problemas de tiempo de ejecución en complejos sistemas distribuidos con múltiples hilos.

La misma funcionalidad que hace a un depurador útil para eliminar errores permite ser usado como herramienta de craqueo de software para evadir la protección anticopia, poner “vidas infinitas a un juego”, modificar la gestión de derechos digitales, y otras características de protección del software.

Otra utilidad importante es para los programadores de antivirus, que al no tener el código fuente de los mismos, necesitan de buenos depuradores que les permita ver el comportamiento del código infectado y así poder crear después las vacunas.

La mayoría de los motores de depuración actuales, tales como gdb y dbx proporcionan interfaces basadas en línea de comandos. Para facilitar su uso desde los IDE, se utilizan frontales de depuración, extensiones (pluggins) que proporcionan integración con el IDE.

En la siguiente imagen se puede observar el funcionamiento de un depurador a nivel de código máquina y ensamblador, como los utilizados por los programadores de antivirus para localizar el código infeccioso y ver la forma de funcionamiento de un programa.



2.4. Refactorización.

La **refactorización** (*Refactoring*) es una técnica de la ingeniería de software para reestructurar el código fuente de un programa, alterando su estructura interna sin cambiar su comportamiento externo. Se podría llamar también rediseño.

Quitar errores o mejorar un algoritmo no se considera refactorizar. La refactorización es como una limpieza del código donde no se cambia la forma de resolver el problema. Se realiza una vez se ha realizado la codificación y se sabe que funciona.

// Antes de refactoriza:

```
void imprimeFactura() { imprimeEncabezado();
    System.out.println ("Nombre: " + nombre ); System.out.println ("Cantidad: " +
    getCantidad()); }
```

// Después de refactorizar::

```
void imprimeFactura() {
    imprimeEncabezado();
    imprimeDetalles(getCantidad());
}
void imprimeDetalles (double cantidad) {
    // Imprime los detalles de la factura partiendo de una cantidad
```

```
System.out.println ("Nombre: " + nombre );  
System.out.println ("Cantidad: " + cantidad );  
}
```

Antes de refactorizar:

class Program

```
{  
    static double Z;  
    static void Main(string[] args)  
    {  
        Console.WriteLine("Bienvenido");  
        Console.WriteLine("Ingrese base del triangulo: ");  
        int X= int.Parse(Console.ReadLine());  
        Console.WriteLine("Ingrese altura del triangulo: ");  
        int Y = int.Parse(Console.ReadLine());  
        calcula(X, Y);  
        Console.WriteLine("El area de el triangulo es: " + Z);  
        Console.ReadKey();  
    }  
    static void calcula(int x, int y)  
    {  
        Z = (x * y) / 2;  
    }  
}
```

Después de refactorizar:

class Program

```
{  
    static double Area; // área del triángulo  
    static void Main(string[] args)  
    {  
        //Programa que calcula el área de un triangulo  
        Console.WriteLine("Bienvenido");  
        Console.WriteLine("Ingrese base del triangulo: ");  
        int Base = int.Parse(Console.ReadLine());  
        Console.WriteLine("Ingrese altura del triangulo: ");  
        int Altura = int.Parse(Console.ReadLine());  
        calcularAreaTriangulo(Base, Altura);  
        Console.WriteLine("El area de el triangulo es: " + Area);  
        Console.ReadKey();  
    }  
    static void calcularAreaTriangulo(int b, int a)  
    //calcula el área de un triángulo partiendo de la base (b) y la altura (a)  
    {  
        Area = (b * a) / 2;  
        return;  
    }  
}
```

2.5. Control de versiones.

Una versión, revisión o edición de un producto, es el estado en el que se encuentra dicho producto en un momento dado de su desarrollo o modificación.



Se llama **control de versiones** a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, así como las posibles especializaciones realizadas (por ejemplo, para algún cliente específico).

El control de versiones se realiza principalmente en la industria informática para controlar las distintas versiones del código fuente. Sin embargo, los mismos conceptos son aplicables a otros ámbitos como documentos, imágenes, sitios web, etcétera.

Aunque un sistema de control de versiones puede realizarse de forma manual, es muy aconsejable disponer de herramientas que faciliten esta gestión (Git, CVS, Subversion, SourceSafe, ClearCase, Darcs, Bazaar, Plastic SCM, Mercurial, etc.).

2.6. Interfaz gráfica de usuario.

En el contexto de interacción persona-ordenador, la interfaz gráfica de usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

La **interfaz gráfica de usuario**, conocida también como **GUI** (*Graphical User Interface*) es un programa informático que actúa como medio de comunicación entre usuario y equipo, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles. Su principal uso consiste en proporcionar un entorno visual sencillo para permitir la comunicación del usuario con el sistema.

En el caso de los Sistemas Operativos, surge como evolución de las interfaces de línea de comandos (cli) que se usaban para operar los primeros sistemas operativos. Como ejemplos de interfaz gráfica de usuario, cabe citar los entornos de escritorio Windows, el X-Window de GNU/Linux o el de Mac OS X, Aqua.



3. Ejemplos de Entornos de desarrollo Integrados

■ Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent o Azureus.



Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Eclipse fue liberado originalmente bajo la Common Public License, pero después fue re-licenciado bajo la Eclipse Public License. La Free Software Foundation ha dicho que ambas licencias son licencias de software libre, pero son incompatibles con la Licencia pública general de GNU (GNU GPL).

■ NetBeans.

NetBeans es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.



NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo. Sun Microsystems fundó el proyecto de código abierto NetBeans en junio de 2000 y continúa siendo el patrocinador principal de los proyectos.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados *módulos*. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de NetBeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las

aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

■ MonoDevelop

MonoDevelop es un entorno de desarrollo integrado libre y gratuito, diseñado primordialmente para C# y otros lenguajes .NET como Nemerle, Boo, Java (vía IKVM.NET) y en su versión 2.2 Python. MonoDevelop originalmente fue una adaptación de SharpDevelop para Gtk#, pero desde entonces se ha desarrollado para las necesidades de los desarrolladores del Proyecto Mono. El IDE incluye manejo de clases, ayuda incorporada, completamiento de código, Stetic (diseñador de GUI) integrado, soporte para proyectos, y un depurador integrado desde la versión 2.2.



MonoDevelop puede ejecutarse en las distintas distribuciones de Linux y en Mac. Desde la versión 2.2, MonoDevelop ya cuenta con soporte completo para GNU/Linux, Windows y Mac, completando así un hito para ser un verdadero IDE Multiplataforma.

■ Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.



Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se intercomunican entre estaciones de trabajo, páginas web y dispositivos móviles.

■ Anjuta DevStudio

Es un entorno de desarrollo de software con grandes utilidades para la programación, como por ejemplo Interfaz gráfica, editor de código para distintos lenguajes, gestor de proyectos, asistentes para la creación de aplicaciones, depurador de errores, control de versiones. Es software libre y de código abierto, disponible bajo la Licencia Pública General de GNU.



■ Aptana Studio

Es un entorno de desarrollo integrado de software libre basado en eclipse y desarrollado por Aptana, Inc., que puede funcionar bajo Windows, Mac y Linux y provee soporte para lenguajes como: Php, Python, Ruby, CSS, Ajax, HTML y Adobe AIR. Tiene la posibilidad de incluir complementos para nuevos lenguajes y funcionalidades.



■ IntelliJ IDEA



Entorno de desarrollo Java creado por Jet Brains. Existen dos distribuciones: Community Edition (gratuita) y Ultimate (comercial). A pesar de ser de pago la versión más completa, tiene muchos seguidores ya que lo consideran el IDE más inteligente a la hora de trabajar.

4. Uso de herramientas CASE en el desarrollo de software

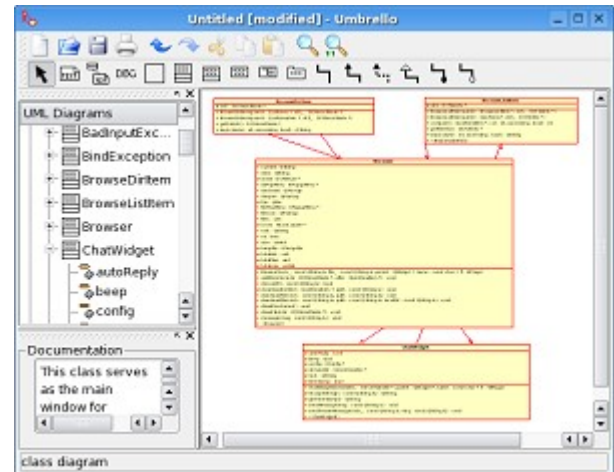
4.1. Herramienta CASE.

Las **herramientas CASE** (*Computer Aided Software Engineering*, Ingeniería de Software Asistida por Computadora) son aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

Sistema de software que intenta proporcionar ayuda automatizada a las actividades del proceso de software.

Los sistemas CASE a menudo se utilizan como apoyo al método.



4.2. Historia.

Ya en los años 70 un proyecto llamado ISDOS diseñó un lenguaje, y por lo tanto un producto, que analizaba la relación existente entre los requisitos de un problema y las necesidades que éstos generaban, el lenguaje en cuestión se denominaba PSL (Problem Statement Language) y la aplicación que ayudaba a buscar las necesidades de los diseñadores PSA (Problem Statement Analyzer).

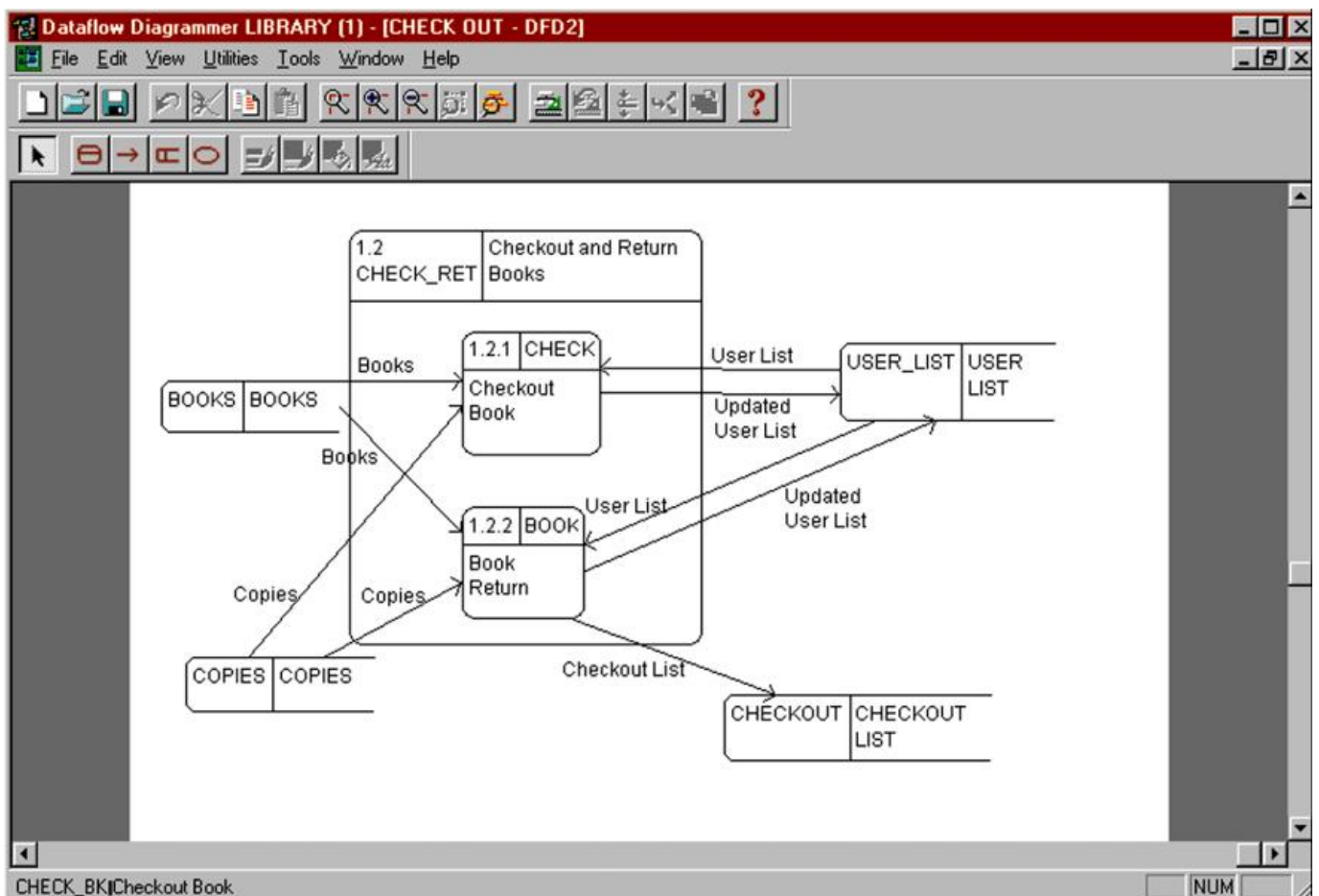
Aunque éstos son los inicios de las herramientas informáticas que ayudan a crear nuevos proyectos informáticos, la primera herramienta CASE fue Excelerator que salió a la luz en el año 1984 y trabajaba bajo una plataforma PC.

Las herramientas CASE alcanzaron su techo a principios de los años 90. En la época en la que IBM había conseguido una alianza con la empresa de software AD/Cycle para trabajar con sus mainframes, estos dos gigantes trabajaban con herramientas CASE que abarcaban todo el ciclo de vida del software. Pero poco a poco los mainframes han ido siendo menos utilizados y actualmente el mercado de las “Big CASE” ha muerto completamente abriendo el mercado de diversas herramientas más específicas para cada fase del ciclo de vida del software.

4.3. Objetivos.

- Mejorar la productividad en el desarrollo y mantenimiento del software.
- Aumentar la calidad del software.
- Reducir el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.

- Mejorar la planificación de un proyecto
- Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
- Automatizar el desarrollo del software, la documentación, la generación de código, las pruebas de errores y la gestión del proyecto.
- Ayuda a la reutilización del software, portabilidad y estandarización de la documentación.
- Gestión global en todas las fases de desarrollo de software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.



4.4. Clasificación.

Aunque no es fácil y no existe una forma única de clasificarlas, las herramientas CASE se pueden clasificar teniendo en cuenta los siguientes parámetros:

1. Las plataformas que soportan.
2. Las fases del ciclo de vida del desarrollo de sistemas que cubren.

3. La arquitectura de las aplicaciones que producen.
4. Su funcionalidad.

La siguiente clasificación es la más habitual basada en las fases del ciclo de desarrollo que cubren:

- *Upper CASE (U-CASE)*, herramientas que ayudan en las fases de planificación, análisis de requisitos y estrategia del desarrollo, usando, entre otros diagramas UML.
- *Middle CASE (M-CASE)*, herramientas para automatizar tareas en el análisis y diseño de la aplicación.
- *Lower CASE (L-CASE)*, herramientas que semi-automatizan la generación de código, crean programas de detección de errores, soportan la depuración de programas y pruebas. Además automatizan la documentación completa de la aplicación. Aquí pueden incluirse las herramientas de [Desarrollo rápido de aplicaciones](#).

Existen otros nombres que se le dan a este tipo de herramientas, y que no es una clasificación excluyente entre sí, ni con la anterior:

- *Integrated CASE (I-CASE)*, herramientas que engloban todo el proceso de desarrollo software, desde análisis hasta implementación.
- *MetaCASE*, herramientas que permiten la definición de nuestra propia técnica de modelado, los elementos permitidos del metamodelo generado se guardan en un repositorio y pueden ser usados por otros analistas, es decir, es como si definiéramos nuestro propio UML, con nuestros elementos, restricciones y relaciones posibles.
- *CAST (Computer-Aided Software Testing)*, herramientas de soporte a la prueba de software.
- *IPSE (Integrated Programming Support Environment)*, herramientas que soportan todo el ciclo de vida, incluyen componentes para la gestión de proyectos y gestión de la configuración.

Por funcionalidad podríamos diferenciar algunas como:

- Herramientas de generación semiautomática de código.
- Editores [UML](#).
- Herramientas de [Refactorización](#) de código.
- Herramientas de mantenimiento como los [sistemas de control de versiones](#).

Ejemplos de herramientas CASE libres son: ArgoUML, Use Case Maker, ObjectBuilder...

Como toda herramienta, también pueden tener sus puntos en contra si se hace un uso inadecuado o abuso de las mismas.

Beneficios pobres: Un desarrollador hace uso de herramientas CASE para obtener un beneficio, normalmente económico, ya que se espera una reducción de costes, por tanto, hay que tener en cuenta la

cantidad de dinero que vamos a invertir en la compra de licencias de uso de software de este tipo y la rentabilidad esperada en los proyectos que se realicen.

Bajo rendimiento: A veces, las herramientas CASE que apoyan un proyecto podrían no funcionar como se esperaba. Esto dará como resultado el retraso del proyecto, que también aumentará el coste de desarrollo. Los Requisitos de hardware y software de ciertas herramientas pueden ser superiores a la equipación con la que se cuenta, por lo que puede que las herramientas no produzcan un rendimiento óptimo. Esto puede obligar a la adquisición de nuevos equipos y software adicional.

Requieren una Introducción de datos precisa. El analista de sistemas estudia los requisitos del nuevo software a producir, para ello debe recoger información precisa sobre qué necesidades y requerimientos necesita el usuario para su programa. De no hacerlo bien, el resultado será un mal modelo que se introducirá en las herramientas CASE. Esto desemboca en una salida que es diferente de las necesidades del usuario. En este caso, la organización se puede ver obligada a rehacer todo el sistema de nuevo. Por tanto, el uso de herramientas CASE no aseguran de ningún modo que se realice un buen análisis, de ahí que es importante que estas incorporen utilidades que permitan el buen entendimiento entre analista y cliente.

Enfoque de desarrollo fijo. Las herramientas CASE fijan un método predefinido y los pasos que se deben seguir para el desarrollo de una aplicación. En general esto es bueno ya que obliga a seguir unas directrices que se suponen adecuadas, pero en ocasiones, coarta la creatividad de los desarrolladores. Por ejemplo, el desarrollador puede saber una solución más simple a la lógica de negocio de la aplicación que se va a desarrollar, pero no puede ponerlas en práctica al tener que trabajar con las herramientas CASE de forma estricta. Hay que saber ser un poco flexibles.

Para saber más

En el siguiente enlace se presenta una ampliación de los tipos y ayudas concretas de las herramientas CASE.

[Ayudas concretas de CASE.](#)

4.5. Lenguaje Unificado de Modelado.

Lenguaje Unificado de Modelado -UML (*Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group).



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

UML ofrece un estándar para describir un "plano" del sistema (un modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

Es importante resaltar que UML es un "lenguaje de modelado", no de programación, diseñado para especificar o para describir métodos o procesos. Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo.

Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo de software (tal como el Proceso Unificado Racional o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

UML no puede compararse con la programación estructurada, pues UML significa Lenguaje Unificado de Modelado, no es programación, solo se diagrama la realidad de una utilización en un requerimiento. Mientras que, programación estructurada, es una forma de programar como lo es la orientación a objetos, sin embargo, la programación orientada a objetos viene siendo un complemento perfecto de UML, pero no por eso se toma UML sólo para lenguajes orientados a objetos.

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

