

The background features a dynamic, abstract graphic on the left side. It consists of several curved bands in shades of red, from deep maroon to bright coral, set against a white background. These bands curve from the top left towards the bottom right, creating a sense of motion and depth.

Juan Carlos Pérez González

Desarrollo de Interfaces

DESARROLLO DE INTERFACES

Juan Carlos Pérez González

Prólogo

El siguiente libro no pretende ser un manual de referencia sobre Desarrollo de Interfaces sino una herramienta de ayuda a aquellos docentes que imparten el módulo profesional Desarrollo de Interfaces encuadrado en el 2º curso del Ciclo Formativo Desarrollo de Aplicaciones Multiplataforma.

Se ha dividido en ocho capítulos que coinciden con los bloques de contenidos contemplados en el currículo del módulo tal como se establece en el Decreto 114/2010 que desarrolla las enseñanzas mínimas del mencionado ciclo.

Se ha intentado que los distintos puntos de los bloques de contenidos coincidiesen como apartados en cada unidad. No siempre ha sido posible.

Los contenidos han sido elaborados por el autor buscando y recogiendo diferentes fuentes en la Red. En la mayoría de los casos adaptándolos. De todas formas aquellos textos o imágenes cuyos autores consideren que no deben ser usados aquí y tras demostración de su propiedad serán retirados o modificados.

Al final de cada tema se propone un ejemplo con indicaciones para su resolución y ejercicios propuestos.

Cualquier duda, sugerencia o queja será bien recibida. Espero que sea de útil. Esa es la intención.

A mi hija Rosalía

Indice

| | |
|---|-----|
| Confección de Interfaces de Usuario..... | 1 |
| Generación de Interfaces de Usuario utilizando XML..... | 33 |
| Creación de componentes visuales..... | 56 |
| Usabilidad..... | 67 |
| Confección de informes..... | 91 |
| Documentación de las aplicaciones..... | 107 |
| Distribución de la aplicaciones..... | 123 |
| Realización de pruebas..... | 144 |
| Anexo..... | 161 |

UD1. CONFECCIÓN DE INTERFACES DE USUARIO

Objetivos

RA1. Generar interfaces gráficas de usuario mediante editores visuales utilizando la funcionalidad del editor y adaptando el código generado.

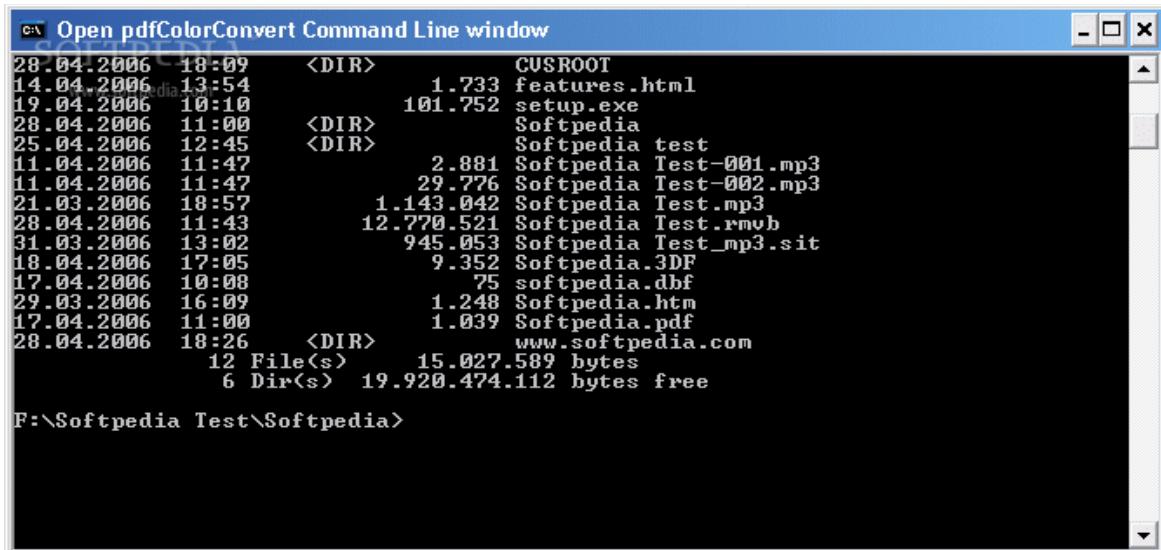
- CA1.1. Se crea una interfaz gráfica utilizando los asistentes de un editor visual.
- CA1.2. Se utilizan las funciones del editor para localizar los componentes de la interfaz.
- CA1.3. Se modifican las propiedades de los componentes para adecuar a las necesidades de la aplicación.
- CA1.4. Se analiza código generado por el editor visual.
- CA1.5. Se modifica el código generado por el editor visual.
- CA1.6. Se asocian a los eventos las acciones correspondientes.
- CA1.7. Se desarrolla una aplicación sencilla para comprobar la funcionalidad de la interfaz gráfica obtenida.

Confección de interfaces de usuario

- Librerías de componentes disponibles para diferentes sistemas operativos y diversas lenguajes de programación: características.
- Herramientas propietarias y libres de edición de interfaces.
- Componentes de la interfaz visual: características y campo de aplicación. Localización e alineación.
- Unión de componentes a las fuentes de los datos.
- Asociación de acciones a eventos.
- Diálogos modales e non modales.
- Edición do código generado por la herramienta de diseño.
- Clases, propiedades e métodos.
- Eventos: escuchadores.

→ Introducción

No fue hasta finales de la década de los sesenta cuando aparecieron las primeras interfaces gráficas de usuario tal como las entendemos hoy. Anteriormente las interfaces de usuario eran simples **CLI (Interfaces Command Line)** o donde se introducían las órdenes mediante comandos y que aún hoy en día se utilizan.



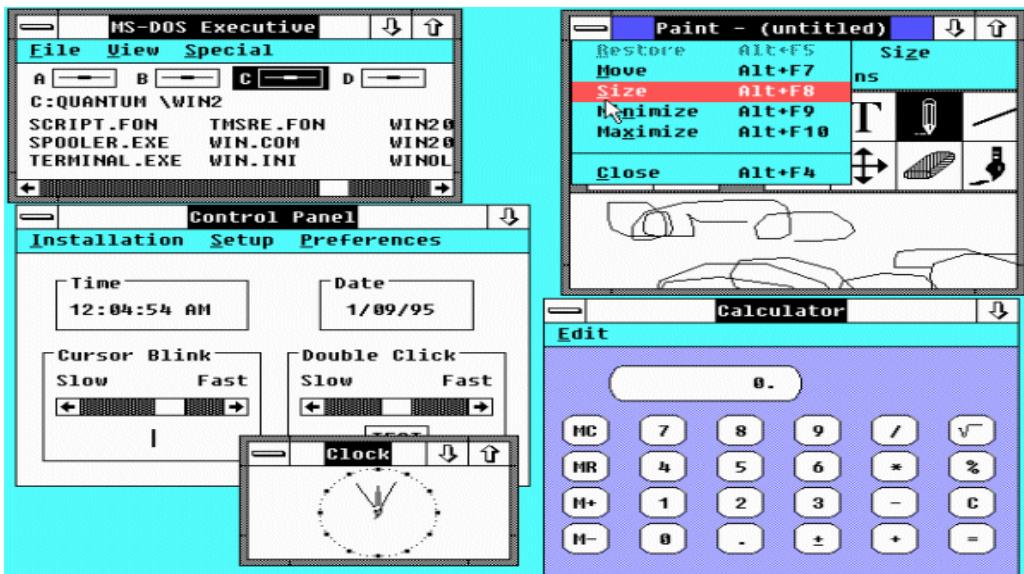
```

C:\ Open pdfColorConvert Command Line window
28.04.2006 18:09 <DIR> CUSROOT
14.04.2006 13:54 1.733 features.html
19.04.2006 10:10 101.752 setup.exe
28.04.2006 11:00 <DIR> Softpedia
25.04.2006 12:45 <DIR> Softpedia test
11.04.2006 11:47 2.881 Softpedia Test-001.mp3
11.04.2006 11:47 29.776 Softpedia Test-002.mp3
21.03.2006 18:57 1.143.042 Softpedia Test.mp3
28.04.2006 11:43 12.770.521 Softpedia Test.rmvb
31.03.2006 13:02 945.053 Softpedia Test_mp3.sit
18.04.2006 17:05 9.352 Softpedia.3DF
17.04.2006 10:08 75 softpedia.dbf
29.03.2006 16:09 1.248 Softpedia.htm
17.04.2006 11:00 1.039 Softpedia.pdf
28.04.2006 18:26 <DIR> www.softpedia.com
12 File(s) 15.027.589 bytes
6 Dir(s) 19.920.474.112 bytes free

F:\Softpedia Test\Softpedia>

```

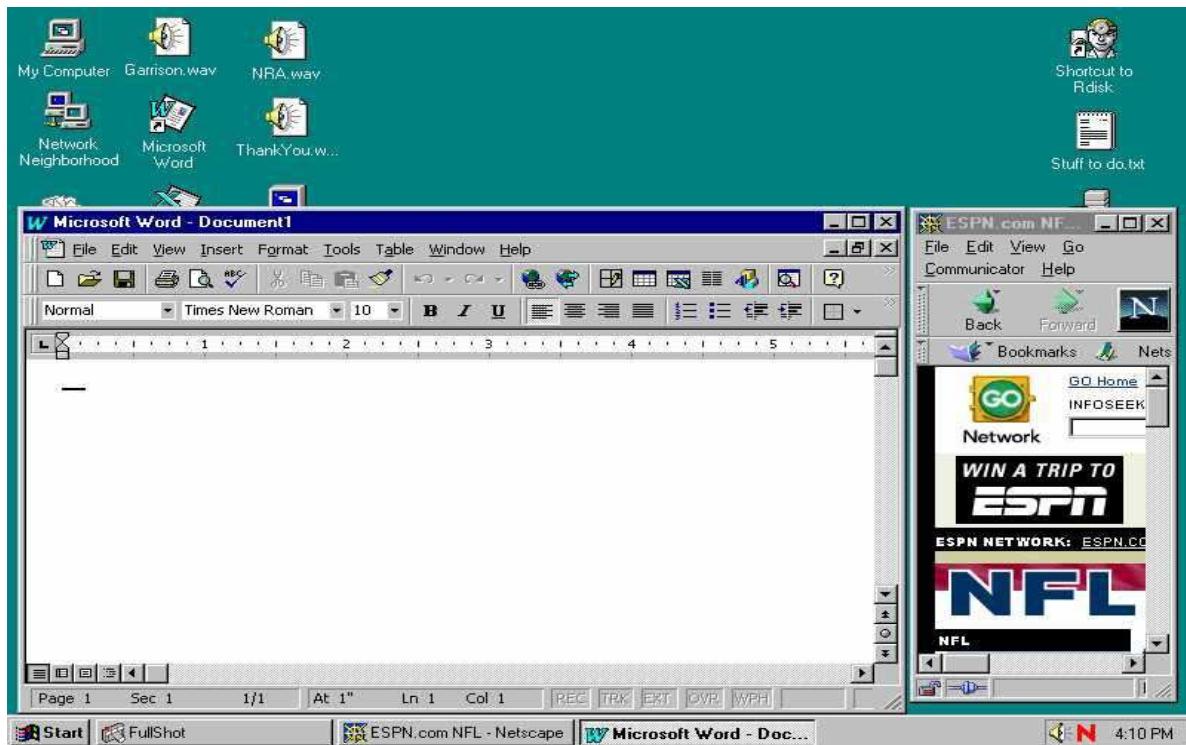
Posteriormente aparecieron **los primeros IU o Interfaces de usuario** con menús jerárquicos como por ejemplo Windows 3.1.



A finales de los ochenta aparecieron las **interfaces WIMP¹ (Windows, Icons, Menus and Pointing device)**, es decir, ventanas, iconos, menus y punteros. Con el tiempo se mejoraron con

¹ No confundir con WYSIWYG

versiones con simulación 3D y la manipulación. Dicha manipulación se hace directa mediante la representación visual de los objetos y la aparición de los eventos.



En el futuro se irá hacia la generación eventos mediante voz o vista e incluso el ordenador tomará sus decisiones y acciones en función de la observación del usuario.

Más del 50% del código de un programa es generado por la parte de interfaces gráficas de ahí la importancia de utilizar IDE, es decir, entornos de desarrollo que nos permite un diseño rápido y eficiente de dichas interfaces.

Apunte histórico

- 1981 – Xerox Star (ventanas y 1^a aparición del ratón)
- 1984 – Apple Macintosh, X Windows (popularización del ratón)

"There is no evidence that people want to use these things"
- 1984/7 – X Window System (X11, MIT, separado del SO y WM)
- 1985 – MS Windows (integrado en el SO e incluye WM)
- Finales de los 90 – Interfaces Web...



→ Librerías de componentes disponibles para diferentes sistemas operativos y diversos lenguajes de programación: características.

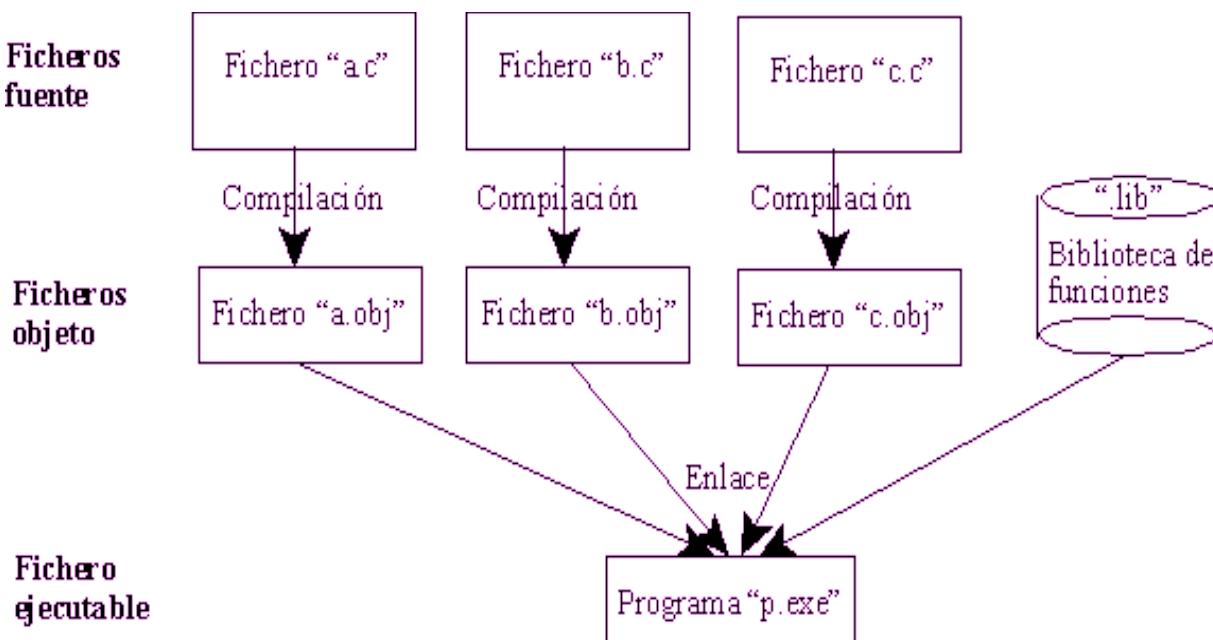
El término **librería**, en programación, es una incorrecta traducción del término inglés *library* y que en realidad hace referencia al término en castellano de *biblioteca*.

Una **biblioteca** es un conjunto de implementaciones o **subprogramas** con una interfaz bien definida para ser invocados y que son utilizados para desarrollar software. Las bibliotecas contienen **código y datos**, que **proporcionan servicios a programas independientes**, es decir, pasan a formar parte de estos. Esto permite que el código y los datos **se compartan y puedan modificarse de forma modular**. Ejecutables y bibliotecas hacen referencias entre sí a través de un proceso conocido como **enlace** realizado por un software denominado **enlazador**.

A diferencia de un programa ejecutable, el comportamiento que implementa una biblioteca **no espera ser utilizada de forma autónoma** (un programa sí: tiene un punto de entrada principal), sino que su fin es **ser utilizada por otros programas, independientes y de forma simultánea**.

Por otra parte, el comportamiento de una biblioteca no tiene porqué diferenciarse en demasiado del que pudiera especificarse en un programa. Es más, unas bibliotecas pueden requerir de otras para funcionar, pues el comportamiento que definen refina, o altera, el comportamiento de la biblioteca original; o bien la hace disponible para otra tecnología o lenguaje de programación.

La mayoría de los sistemas operativos modernos **proporcionan bibliotecas que implementan los servicios del sistema** a los programas instalados en el equipo. De esta manera, estos servicios se han convertido en una "materia prima" que cualquier aplicación moderna que el sistema operativo ofrece a los mismos (por ejemplo el módulo de impresión).



Podemos dividir las librerías en dos tipos:

- **Librerías estáticas:** es aquella que se enlaza en **tiempo de compilación**. La ventaja de este tipo de enlace es que hace que un programa no dependa de ninguna biblioteca (la enlazó al compilar), haciendo más fácil su distribución. Su inconveniente es que los **programas son más pesados y menos flexibles** a la hora de modificar su código.
- **Librería dinámicas:** es aquella enlazada cuando un determinado programa **se ejecuta**. La ventaja de este tipo de enlace es que el programa es más liviano, y que evita la duplicación de código (por ejemplo, cuando dos programas requieren usar la misma biblioteca, se necesita sólo una copia de ésta).

Las bibliotecas de enlace dinámico, o bibliotecas compartidas, suelen encontrarse en **directorios específicos del sistema operativo**, de forma que, cada vez que un programa necesite usar alguna, el sistema operativo conozca el lugar en el que se encuentra, para así poder enlazarla. Esto ocasiona algunos problemas de **dependencias**, principalmente entre diferentes versiones de una misma biblioteca. En Windows las librerías dinámicas se denominan **DLL (Dynamic-Link Library)**

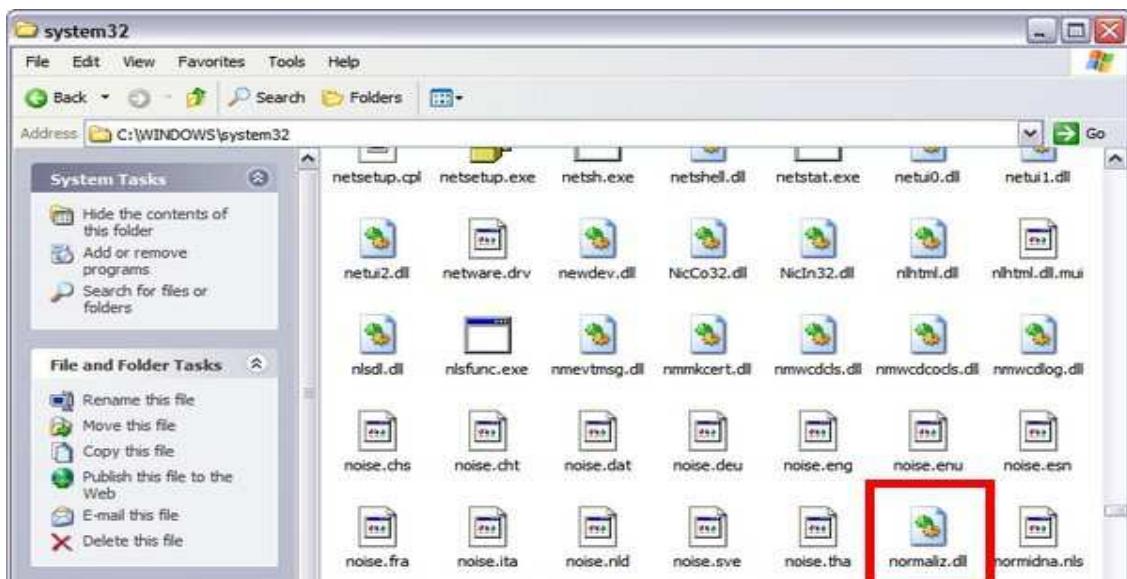
Muchos programas tienen procedimientos a los que no llaman, salvo en circunstancias excepcionales. Haciendo uso de bibliotecas de enlace dinámico reusables evita la codificación repetitiva de las mismas.

Una de las mayores desventajas del enlace dinámico es que el funcionamiento correcto de los ejecutables **depende de una serie de bibliotecas almacenadas de forma aislada**. Si la biblioteca es borrada, movida o renombrada, o si una versión incompatible de DLL es copiada en una ubicación que aparece antes en la ruta de búsqueda, el ejecutable no se podrá cargar.

Los tipos de librerías también se pueden clasificar según otros criterios, por ejemplo, según el lenguaje de programación. Solo se hará referencia a los lenguajes de uso más habitual ya que el número es muy elevado si considerásemos a todos los lenguajes de programación. Entre los tipos de módulos o librerías a destacar tenemos:

- **C/C++:** podemos dividirlas en:
 - **Estáticas:** Denominadas también librerías-objeto, son colecciones de ficheros objeto (compilados) agrupados en un solo fichero de extensión **.lib**, **.a**, etc. junto con uno o varios ficheros de cabecera (generalmente **.h**). Estas últimas contienen los ficheros estándar **stdio.h**, **math.h**, **string.h...** y **la graphics.h** que aún no siendo considerada como estándar está incluida en los compiladores Borland C++ para el desarrollo de aplicaciones.

- **Dinámicas:** conocidas como **DLL's**, acrónimo de su nombre en inglés ("Dynamic Linked Library"). Estas librerías se utilizan mucho en la programación para el SO Windows. Este Sistema contiene un gran número de tales librerías de terminación .DLL, aunque en realidad pueden tener cualquier otra terminación .EXE, .FON, .BPI, .DRV etc. Cualquiera que sea su terminación, de forma genérica nos referiremos a ellas como DLL's, nombre por el que son más conocidas. Suelen encontrarse en el carpeta **system32**. En Linux en el directorio **/lib**. La falta o fallo de alguna de ellas necesaria impide la ejecución de uno o más programas.



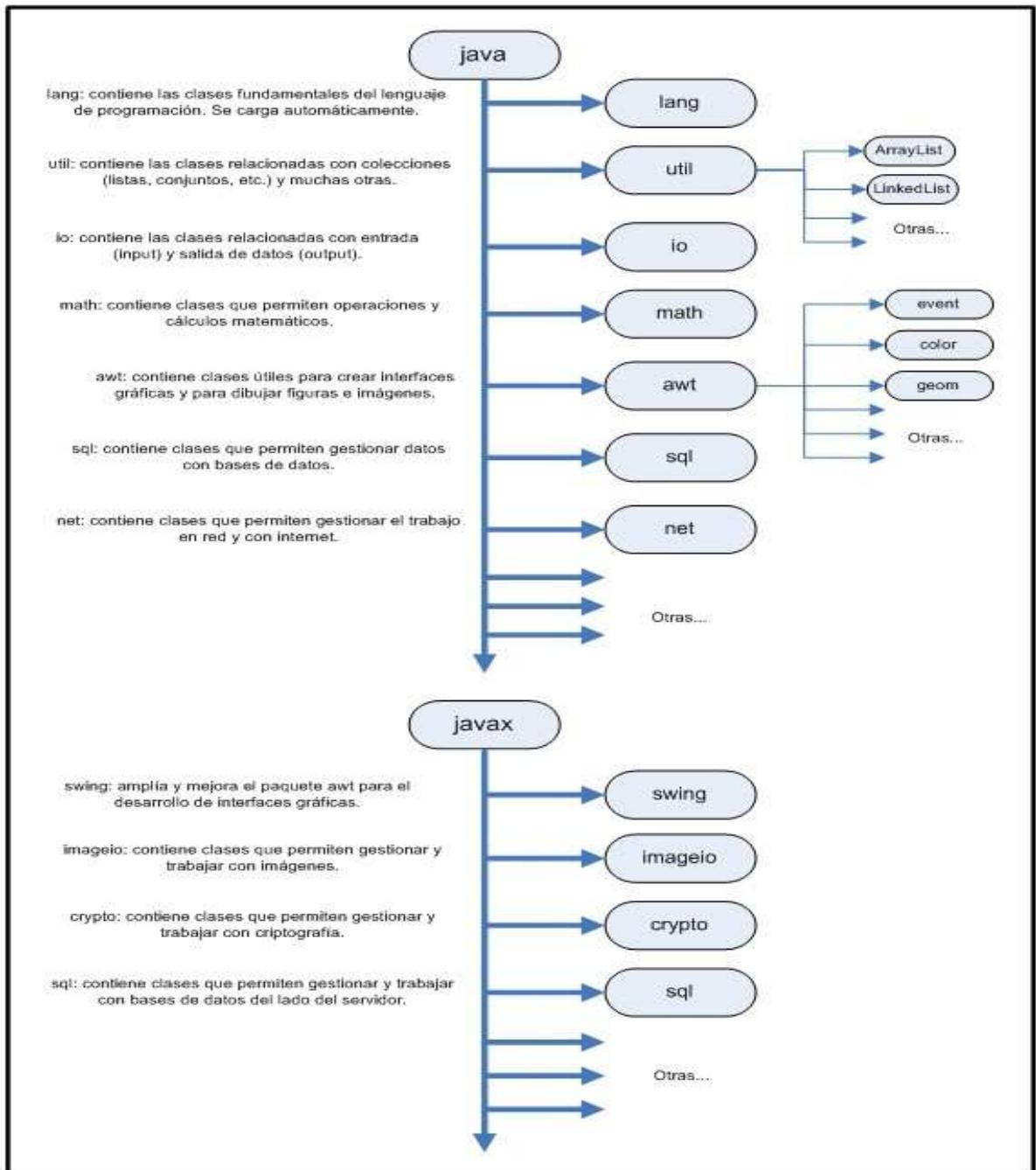
- **Java:** La biblioteca estándar de Java está compuesta por cientos de clases como System, String, Scanner, ArrayList, HashMap, etc. que nos permiten hacer casi cualquier cosa. Nos centraremos en las más interesantes para nosotros, es decir, aquellas relacionadas directamente con la creación de entorno gráficos de usuario. Estas son:

- **AWT (Abstract Window Toolkit):** esta contenida en la rama **java de la API del lenguaje** permite desarrollar interfaces de usuario gráficas. Es la librería básica y aunque ahora se usa más la Swing.

Esta compuesta por:

- Componentes (*java.awt.Component*), como los Buttons, Labels,..
- Contenedores (*java.awt.containers*), contienen componentes.
- Gestores de posición (*java.awt.LayoutManager*), que posiciona los componentes dentro de los contenedores.
- Los eventos (*java.awt.Event*), que nos indican las acciones o eventos.

- **Javax.swing:** proporciona una serie de clases e interfaces que amplían la funcionalidad del anterior. Es la versión más moderna del API de Java. Están escritos en Java y son independientes de la plataforma.



- **XML:** es un lenguaje de marcas desarrollado por el World Wide Web (W3C) utilizado para almacenar datos en forma legible. Cualquier procesador de texto, que sea capaz de producir archivos .txt es capaz de generar documentos en XML. Los entornos de desarrollo, como Eclipse, Visual Studio o Netbeans, reconocen los formatos y ayuda a generar un XML bien formado gracias a las librerías que lo implementan.

De hecho XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el **intercambio de información estructurada entre diferentes plataformas**. Se puede usar en bases de datos, editores de texto, hojas de cálculo... Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Entre las herramientas más importantes relacionadas con XML tenemos:

- **Estándares de parseo (analizadores sintácticos de documentos XML)**
 - Se recomienda utilizar el estándar **DOM** con documentos XML pequeños, cuando queramos analizar el documento múltiples veces o editararlo, ya que se encuentra cargado en memoria, o queramos generar un documento XML desde cero.
 - Se recomienda utilizar el estándar **SAX** con documentos XML grandes, cuando queramos analizar el documento una sola vez o por partes (capturando los elementos importantes). También se usa cuando no se requiere una modificación estructural.
- **Análisis del XML**
 - **Xerces2** permite el procesamiento de documentos XML tanto con el estándar DOM o con el estándar SAX, y errores. Además, integra a otras librerías independientes de parseado.
 - **JDOM** permite leer, escribir, crear y manipular ficheros XML de forma sencilla e intuitiva. Está totalmente programada en Java, lo que le permite utilizar las capacidades particulares del lenguaje, simplificando significativamente su manejo para programadores expertos en Java. Se basa en el procesamiento de un documento XML y la construcción de un árbol. Una vez construido el árbol se puede acceder directamente a cualquiera de sus componentes.
 - **JAXP (Java API for XML Processing)** permite procesar tanto el estándar DOM como el estándar SAX. Este API está diseñado para ser flexible y uniformar el desarrollo de aplicaciones Java con XML. Además, proporciona una capa intermedia que nos permite usar cualquier analizador XML compatible dentro de nuestra aplicación, reduciendo el acoplamiento de los componentes de la aplicación con la implementación del analizador.
- **Serialización del XML**
 - La **serialización** es un mecanismo ampliamente usado para transportar objetos a través de una red, para **hacer persistente un objeto en un archivo o base de datos**, o para distribuir objetos idénticos a varias aplicaciones o localizaciones.

XML es un formato estándar de documentos basados en texto para almacenar datos legibles por aplicaciones que proporciona un fácil procesamiento. Para llevar a cabo este proceso se han de definir mediante reglas tanto el proceso de transformación de objetos Java a XML como el proceso de transformación inversa.

Se recomienda utilizar las librerías **Xerces2**, **Xstream** o **JAXB** para serializar objetos Java en un medio de almacenamiento, como puede ser un archivo o un buffer de memoria, con el fin de transmitirlo a través de una conexión en red, ya sea como una serie de bytes o usando un formato humanamente más legible, como XML o JSON, entre otros. La serie de bytes o el formato empleado para la transmisión pueden ser usados para crear un nuevo objeto que es idéntico en todo al original, incluido su estado interno. Por tanto, el nuevo objeto es un CLON del original.

- **Transformaciones a XML**
 - En ocasiones es necesario permitir realizar transformaciones de documentos XML con lenguajes basados en XSL (XSLT, XPath). Un documento XML puede ser transformado en distintos formatos, como HTML, o en otro documento XML. Para realizar estas transformaciones basadas en XSLT, se recomienda el empleo de la librería **Xalan**.
- **Vinculación con objetos Java**
 - Por la influencia del lenguaje Java en la programación actual usando **JiBX**, **JAXB** o **XMLBeans** podemos vincular datos en XML con objetos Java. Con cualquiera de las tres librerías podemos, o partir de un esquema XML generar código Java. Se recomienda utilizar una de estas tres librerías para la vinculación de los documentos XML con los objetos Java.
- **Librerías gráficas.**

De una forma sencilla podemos definir una librería gráfica como el **software que genera imágenes, en base a modelos matemáticos y diversos patrones de iluminación, texturas**, etc.... El objetivo principal es su independencia del hardware e independientes de la aplicación.

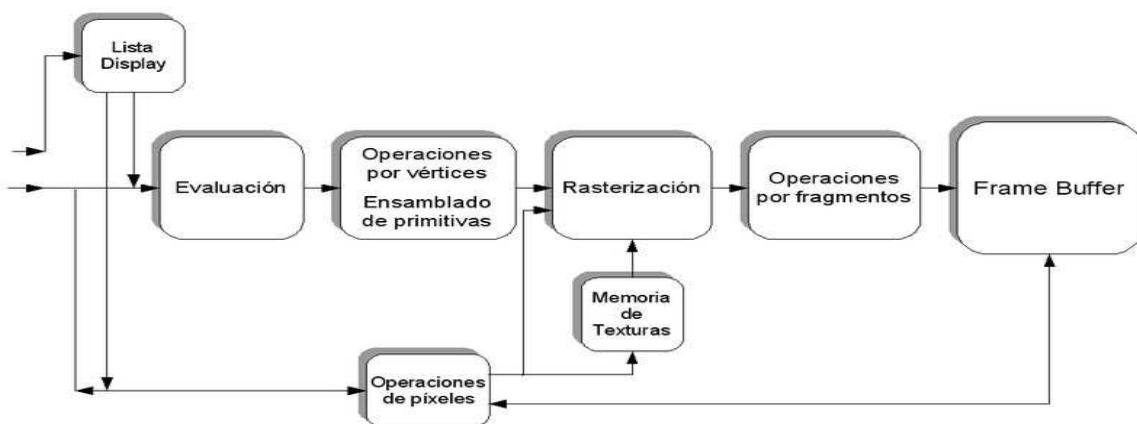
 - **OpenGL (Open Graphics Library)** es una **especificación estándar que define una API multilenguaje y multiplataforma** para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por

Silicon Graphics Inc. (SGI) y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información simulación de vuelos y desarrollo de videojuegos, donde compite con Direct3D en plataformas Microsoft Windows.

El funcionamiento básico de OpenGL **consiste en aceptar primitivas tales como puntos, líneas y polígonos, y convertirlas en píxeles**. Este proceso es realizado por una *pipeline* gráfica conocida como **Máquina de estados de OpenGL**. La mayor parte de los comandos de OpenGL bien emiten primitivas a la pipeline gráfica o bien configuran cómo la pipeline procesa dichas primitivas.

Una **descripción somera** del proceso en la pipeline gráfica podría ser:

1. **Evaluación, si procede, de las funciones polinomiales** que definen ciertas entradas.
2. **Operaciones por vértices, transformándolos, iluminándolos según su material y recortando partes no visibles** de la escena para producir un volumen de visión.
3. Rasterización, o **conversión de la información previa en píxeles**. Los polígonos son representados con el color adecuado mediante algoritmos de interpolación.
4. **Operaciones por fragmentos o segmentos**, como actualizaciones según valores venideros o ya almacenados de profundidad y de combinaciones de colores, entre otros.
5. Por último, los fragmentos son **volcados en el Frame buffer**. (dispositivo gráfico que representa los píxeles de la pantalla como ubicaciones en la memoria de acceso aleatorio o RAM gráfica).



Se han realizado **bindings o adaptaciones** de OpenGL a diferentes lenguajes, como JOGL para Java o PyOpenGL para Python, entre otros.

- **DirectX-Direct3D:** Es parte de DirectX (conjunto de bibliotecas para multimedia),

propiedad de Microsoft. Consiste en una API para la programación de gráficos 3D. Está disponible tanto en los sistemas Windows de 32 y 64 bits, como para sus consolas Xbox.

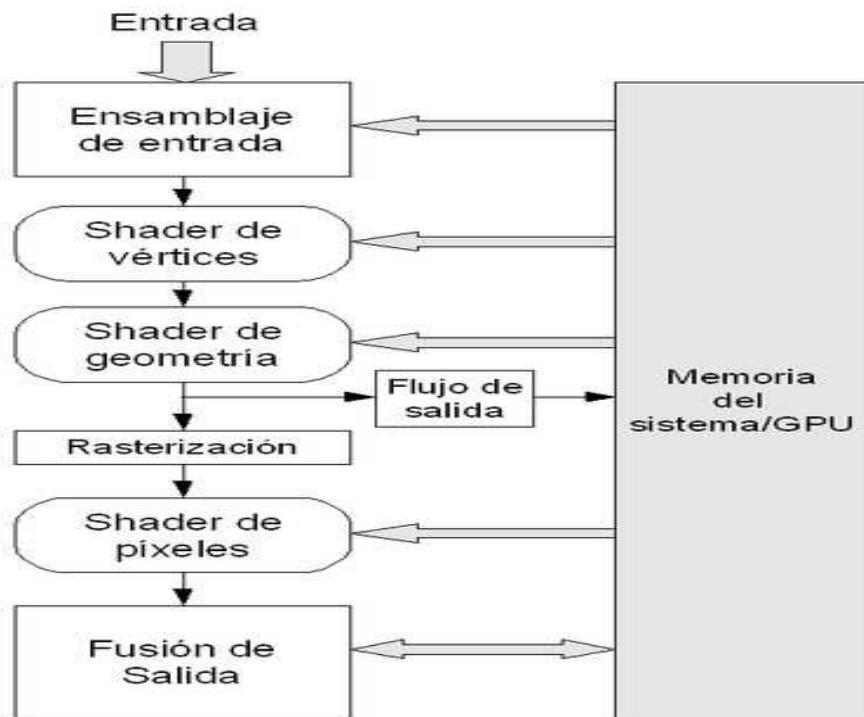
El objetivo de esta API es facilitar el **manejo y trazado de entidades gráficas elementales, como líneas, polígonos y texturas**, en cualquier aplicación que despliegue gráficos en 3D, así como efectuar de forma transparente transformaciones geométricas sobre dichas entidades. Direct3D provee también una interfaz transparente con el hardware de aceleración gráfica.

Direct3D está compuesto por dos grandes APIs: el **modo inmediato** y el **modo retenido**.

- El **modo inmediato** da soporte a todas las primitivas de procesamiento 3D que permiten las tarjetas gráficas (luces, materiales, transformaciones, control de profundidad, etc).
- El **modo retenido**, construido sobre el anterior, presenta una abstracción de nivel superior ofreciendo funcionalidades preconstruidas de gráficos como jerarquías o animaciones.

Las diferentes etapas del proceso de **renderización** son los siguientes:

1. **Input Assembler**: aporta los datos de entrada para la elaboración de la imagen (líneas, puntos y triángulos).
2. **Vertex Shader**: se encarga de las operaciones de vértices (iluminación, texturas, transformaciones). Trata los vértices individualmente.
3. **Geometry Shader**: realiza operaciones con entidades primitivas (líneas, triángulos o vértices). A partir de una primitiva, el geometry shader puede descartarla, o devolver una o más primitivas nuevas.
4. **Stream Output**: almacena la salida de la etapa anterior en memoria. Resulta útil para realimentar la pipeline con datos ya calculados.
5. **Rasterizer**: convierte la imagen 3D en píxeles.
6. **Pixel Shader**: operaciones con los píxeles.
7. **Output Merger**: se encarga de combinar la salida del pixel shader con otros tipos de datos, como los patrones de profundidad, para construir el resultado final.



- **GTK "GIMP Tool Kit"** es una biblioteca del equipo GTK+, la cual contiene los objetos y funciones para crear la interfaz gráfica de usuario. Maneja **widgets** como ventanas, botones, menús, etiquetas, deslizadores, pestañas, etc GNOME utiliza estas librerías. GTK+ se ha diseñado para permitir programar con lenguajes como C, C++, C#, Java, Ruby, Perl, PHP o Python.

GTK+ se basa en varias bibliotecas desarrolladas por el equipo de GTK+ y de GNOME:

- **GLib.** Biblioteca de bajo nivel estructura básica de GTK+ y GNOME. Proporciona manejo de estructura de datos para C, portabilidad, interfaces para funcionalidades de tiempo de ejecución como ciclos, hilos, carga dinámica o un sistema de objetos.
- **GTK.** Biblioteca la cual realmente contiene los objetos y funciones para crear la interfaz de usuario. Maneja **widgets** como ventanas, botones, menús, etiquetas, deslizadores, pestañas, etc.
- **GDK.** Biblioteca que actúa como intermediario entre gráficos de bajo nivel y gráficos de alto nivel.
- **ATK.** Biblioteca para crear interfaces con características de una gran accesibilidad muy importante para personas discapacitadas o minusválidos. Pueden usarse utilerías como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o ratón.
- **Pango.** Biblioteca para el diseño y renderizado de texto, hace hincapié especialmente en la internacionalización. Es el núcleo para manejar las fuentes y el

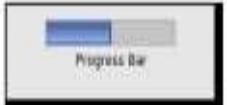
texto de GTK+2.

Algunos ejemplos de clases de GTK:

- Clases para **construir ventanas**

| Clase | Descripción |
|---|---|
| GtkDialog | Crea una ventana <i>popup</i> . |
| GtkInvisible | Un <i>widget</i> que no es mostrado. |
| GtkMessageDialog | Una ventana de mensaje. |
| Una ventana que puede contener otros <i>widgets</i> . | |
| GtkWindowGroup | Una ventana como la anterior pero con funciones limitadas. |
| GtkAboutDialog | Muestra información sobre una aplicación |
| GtkAssistant | Un <i>widget</i> que se usa para guiar a los usuario a través de operaciones de varios pasos (asistente). |

- **Widgets** que solo muestran información sin permitir la entrada de datos:

| Clase | Descripción | Imagen |
|----------------|--|---|
| GtkAccelLabel | Etiqueta que muestra información de una tecla aceleradora a la derecha del texto |  |
| GtkImage | Widget que permite mostrar una imagen. |  |
| GtkLabel | Widget que permite mostrar una pequeña o mediana cantidad de texto. |  |
| GtkProgressBar | Widget que indica el estado de un progreso. |  |
| GtkStatusbar | Reporta mensajes de menor importancia al usuario en la barra de estado. |  |
| GtkStatusIcon | Muestra un ícono en el área de notificación del sistema. | |

- Clases para construir botones

| Clase | Descripción | Imagen |
|------------------------|---|--------|
| GtkButton | Botón convencional. Crea una señal cuando se da click sobre él. | |
| GtkCheckButton | Etiqueta con un botón el cual puede estar en dos estados: seleccionado o no seleccionado. | |
| GtkRadioButton | Opción de una lista múltiple de opciones. | |
| GtkToggleButton | Botón convencional el cual puede mantener su estado. | |
| GtkLinkButton | Botón que enlaza a una URL. | |
| GtkScaleButton | Botón que hace aparecer una escala. | |
| GtkVolumeButton | Botón que hace aparecer un control de volumen. | |

- **QTK:** es una biblioteca multiplataforma usada para desarrollar **aplicaciones con interfaz gráfica de usuario**, así como también para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores. Usada preferentemente **por escritorio como KDE**. Qt utiliza el lenguaje de programación C++ de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de **bindings**. También es usada en sistemas informáticos empotrados para automoción, aeronavegación y aparatos domésticos como frigoríficos.
- **wXWINDOWS:** Es una biblioteca de clases para **C++ y Python**, que permite el desarrollo de aplicaciones con interfaces gráficas de usuario de una manera rápida y sencilla. Su principal característica es que es **multiplataforma**. Se distribuye bajo licencia wXWindows Library License, que es similar a la GNU Library General Public License pero que además

permite usar la biblioteca para desarrollos comerciales (ya sean aplicaciones o modificaciones de la propia biblioteca), siempre y cuando estos desarrollos comerciales no usen ningún código distribuido bajo alguna licencia GNU.

➔ Herramientas propietarias y libres de edición de interfaces

Software libre, con acceso a su código, generalmente gratuito:

- **Ventajas**

- Existen aplicaciones para todas las plataformas (Linux, Windows, Mac Os).
- El precio de las aplicaciones es mucho menor, la mayoría de las veces son gratuitas.
- Libertad de copia.
- Libertad de modificación y mejora.
- Libertad de uso con cualquier fin.
- Libertad de redistribución.
- Facilidad a la hora de traducir una aplicación en varios idiomas
- Mayor seguridad y fiabilidad.
- El usuario no depende del autor del software

- **Inconvenientes:**

- Algunas aplicaciones (bajo Linux) pueden llegar a ser algo complicadas de instalar.
- Inexistencia de garantía por parte del autor.
- Interfaces gráficas menos amigables.
- Menor compatibilidad con el hardware.

Sofware privativo (no propietario mala traducción) no permite acceso al código, aunque puede ser también gratuito.

- **Ventajas**

- Facilidad de adquisición (puede venir preinstalado con la compra del pc, o encontrarlo fácilmente en las tiendas).
- Existencia de programas diseñados específicamente para desarrollar una tarea.
- Las empresas que desarrollan este tipo de software son por lo general grandes y pueden dedicar muchos recursos, sobretodo económicos, en el desarrollo e investigación.

- Interfaces gráficas mejor diseñadas según la apreciación, subjetiva, de algunos usuarios.
 - Mayor compatibilidad con el hardware.
- **Inconvenientes**
 - No existen aplicaciones para todas las plataformas.
 - Imposibilidad de copia.
 - Imposibilidad de modificación.
 - Restricciones en el uso (marcadas por la licencia).
 - Imposibilidad de redistribución.
 - Por lo general suelen ser menos seguras.
 - El coste de las aplicaciones es mayor.
 - El soporte de la aplicación es exclusivo del propietario.
 - El usuario que adquiere este software depende al 100% de la empresa propietaria.

Para desarrollar software por su alta productividad se utilizan entornos de desarrollo o IDE. Un **entorno de desarrollo (IDE)** suele tener los siguientes componentes aunque no necesariamente todos:

- Un editor de texto
- Un compilador
- Un intérprete
- Un depurador
- Un cliente
- Posibilidad de ofrecer un sistema de control de versiones.
- Facilidad para ayuda en la construcción de interfaces gráficas de usuario. Esto últimos es lo que más nos ataña.

Muchos programadores opinan que el uso de estos entornos “ensucian” con líneas de código redundantes la implementación de la aplicación. No se va a proponer un extenso listado de entornos de desarrollo sino los más conocidos por su frecuencia de uso.

Algunos de los entornos de desarrollo de interfaces gráficas:

- **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#, y Visual Basic .NET, al igual que **entornos de desarrollo web** como ASP.NET. aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. Es un producto comercial aunque Microsoft tiene versiones Express Edition gratuitas pero

no libres con ciertas limitaciones en la explotación de las aplicaciones a desarrollar. Adicionalmente, Microsoft ha puesto gratuitamente a disposición de todo el mundo una versión reducida de SQL Server o Express Edition cuyas principales limitaciones son que no soporta bases de datos superiores a 4 GB de tamaño, únicamente se ejecuta en un procesador y emplea 1 GB de RAM como máximo, y no cuenta con el Agente de SQL Server.

- **Gambas** es un lenguaje de programación libre **derivado de BASIC** (de ahí que Gambas quiere decir Gambas Almost Means Basic). Es distribuido con licencia GNU-GPL. Cabe destacar que presenta ciertas similitudes con Java ya que en la ejecución de cualquier aplicación, se requiere un conjunto de librerías interprete previamente instaladas (**Gambas Runtime**) que entiendan el bytecode de las aplicaciones desarrolladas y lo conviertan en código ejecutable por el computador. Por otro lado, es posible desarrollar grandes aplicaciones en poco tiempo.

Permite crear formularios con botones de comandos, cuadros de texto y muchos otros controles y enlazarlos a bases de datos como MySQL, Postgree o SQLite además de facilitar la creación de aplicaciones muy diversas como videojuegos (utilizando OpenGL), aplicaciones para dispositivos móviles (en desarrollo pero muy avanzado), aplicaciones de red (con manejo avanzado de protocolos HTTP, FTP, SMTP, DNS), entre otras .

- **Glade**: es una herramienta de desarrollo visual de interfaces gráficas mediante **GTK/GNOME**. Es independiente del lenguaje de programación y no generando código fuente sino **un archivo XML**. **GtkBuilder** es un formato XML que Glade usa para almacenar los elementos de las interfaces diseñadas. Estos archivos pueden emplearse para construirla en tiempo de ejecución mediante el objeto GtkBuilder de GTK+. GladeXML era el formato que se usaba en conjunto con la biblioteca libglade. Su conexión con lenguajes como Phyton o el IDE Anjuta (compilador C/C++) permite el desarrollo de interfaces en el mundo GNOME. Su sinónimo para KDE es **QTCreator**
- **Delphi**, antes conocido como **CodeGear Delphi**, **Inprise Delphi** y **Borland Delphi**, es un entorno de desarrollo de software diseñado para la programación de propósito general con énfasis en la programación visual. En Delphi se utiliza como lenguaje de programación una versión moderna de Pascal llamada Object Pascal.
- **NetBeans y Eclipse**: ambos son IDEs para JAVA. El primero es de código abierto y admite más de un lenguaje de programación como PHP y Phyton. El segundo aunque gratuito no soporta licencia GNU-GPL
- **Oracle Database**: un entorno que permite la creación de bases de datos y con una herramienta **OracleDesigner** que permite crear las interfaces para acceder a ellas. Tiene

una versión gratuita **Lite**.

- **Anjuta:** de gran influencia en el mundo Linux. Es un (IDE) para programas en los lenguajes C, C++, Java, **Python** y Vala, en sistemas **GNU-Linux** y BSD. Su principal objetivo es trabajar con **GTK** y en el **Gnome**, además ofrece un gran número de características avanzadas de programación. Es software libre y de código abierto, disponible bajo la Licencia Pública General de GNU.
- **Diseñadores web:** Por el lado comercial está **Dreamweaver** que permite no solo el diseño de interfaces web en HTML, XML... sino el desarrollo de aplicaciones basadas por ejemplo en PHP. Soporta ASP.NET, JavaScript, CSS, ColdFusion....
Más austeros pero de igual capacidad y libres tenemos **Amaya**, **Kompozer**, **Aptana** y **NVu** entre otros.

Hay bastantes más herramientas de desarrollo que permiten la creación de ventanas y entornos gráficos. Pero estás, en el mercado actual, son las de mayor implantación y/o interés.

➔ **Componentes de la interfaces visuales: características y campos de aplicación. Localización y alineamiento.**

Las **interfaces visuales** de un programa son un conjunto de elementos hardware y software que **presentan información al usuario** y le permiten interactuar con dicha información y con la computadora.

Existen una serie principios generales que deben acompañar al diseño e implementación de **Interfaces de Usuario(IU)**, ya sea para las IU gráficas, como para la Web, y que son:

- **Sencillas**
- **Intuitivas**
- **Coherentes**
- **Claras**
- **Predecibles**
- **Flexibles**
- **Consistentes**
-

Dentro del diseño de la interacción usuario-computador se tienen en cuenta una serie de disciplinas que van desde lapsicología, filosofía, ciencia cognitiva, ergonomía, ingeniería, sociología, antropología, lingüística y documentación entre otras.

Los elementos básicos de una interfaz gráfica son:

- **Componentes GUI (widgets)**
 - **Objetos visuales** del interfaz o conjunto de componentes anidados: ventanas, contenedores, menús, barras, botones, campos de texto, etc.
- **Disposición (layout): cómo se colocan los componentes para lograr un GUI cómodo de utilizar**
 - **Layout Managers** que gestionan la organización de los componentes gráficos de la interfaz
- **Eventos o interactividad, respuesta a la entrada del usuario :**
 - Desplazamiento del ratón, selección en un menú, botón pulsado, etc.
- **Creación de gráficos y texto - Bibliotecas Graphics**
 - Define fuentes, pinta textos...
 - Para dibujo de líneas, figuras, coloreado,...

El **Escritorio es el contexto más global** dentro de la interfaz gráfica de usuario ya que representa el espacio donde se mueve y administra la información. En base a este concepto se agrupan los demás, como las carpetas, documentos y herramientas en general.



La **metáfora del escritorio** es un excelente recurso en que el usuario puede, de forma intuitiva, relacionar a través de signos o más bien **representaciones simbólicas**, qué tipo de elemento es y cuál es la acción que puede realizar. Esta metáfora es ampliamente utilizada por la mayoría de los sistemas operativos modernos que trabajan con interfaces gráficas; como Windows, Mac OS X, Linux y similares a Unix. Sus elementos están en constante evolución, acondicionamiento y acoplamiento a la semántica humana.

La metáfora de escritorio trata al monitor **como si fuera el escritorio físico del usuario**, sobre el cual pueden ser colocados los objetos tales como documentos y carpetas de documentos. Un documento puede ser abierto en una ventana, que representa **una copia de papel del documento** colocada en el escritorio. También están disponibles pequeñas aplicaciones llamadas **accesorios de escritorio**, como por ejemplo una calculadora o una libreta de notas, etc.

Es de suma importancia señalar que algunos sistemas actuales, específicamente **Windows 8 y su interfaz metro**, buscan romper con este paradigma, es decir, el escritorio aun existe dentro del sistema operativo **pero funciona como una aplicación más**, dejando de ser el protagonista al momento de definir nuestro espacio de trabajo como lo había sido hasta no mucho tiempo.

➔ Enlace de componentes a los orígenes de datos.

Los **datos** son el corazón de todas las interfaces de usuario. Desde las reservas de hotel hasta la consulta del mercado de valores, las interfaces de usuario proporcionan una forma de visualizar e interactuar con alguna forma de datos. La elección de los componentes de la interfaz de usuario que van a estar visibles y la forma de disponerlos para que proporcionen un flujo de trabajo útil dependen principalmente de la naturaleza de los datos con los que se va a trabajar.

Puede que la aplicación trabaje con:

- **orígenes de datos internos**, quizás realizando cálculos con números que un usuario ha escrito en un formulario por ejemplo una calculadora.
- **orígenes de datos externos**, como bases de datos, fuentes o servicios Web, o archivos locales que contengan información.

Las aplicaciones pueden admitir, entre otros, tipos de orígenes de datos externos de tipo XML. En este caso un archivo XML local o remoto que puede suministrar datos en formato XML a la aplicación, puede usar un archivo XML que haya agregado al proyecto o puede establecer el origen de datos en la dirección URL de un archivo XML de un sitio Web.

El **enlace de datos** es el **proceso de conectar los elementos de un origen de datos a los componentes de la interfaz de usuario (controles)**. Esto significa que **cada vez que cambien**

los datos, los componentes de la interfaz reflejarán de manera opcional dichos cambios, y viceversa.

Un enlace se construye, básicamente, entre un **origen** y un **destino**. El **origen** suele ser un **origen de datos u otro control**, y el **destino** siempre es un **control**. En el ejemplo de una barra de desplazamiento, el origen es la propiedad *Value* del control de barra de desplazamiento y el destino es la propiedad *Width* del rectángulo.

El **flujo de datos** se define como la **dirección en la que fluyen los datos entre el origen y el destino**. En el caso de la barra de desplazamiento que escala un rectángulo, sólo se necesita un enlace en una dirección: desde la barra de desplazamiento (origen) al rectángulo (destino). La mayoría de los IDEs ofrecen las siguientes configuraciones de enlace para el flujo de datos:

- **OneWay** Los cambios que se realizan en el origen actualizan automáticamente el destino, pero los cambios en el destino no actualizan el origen. Por ejemplo, cuando elegimos España automáticamente se vuelquen las provincias en un *comboBox*
- **TwoWay**: Los cambios que se realizan en el origen actualizan automáticamente el destino, y viceversa. Por ejemplo, cuando cambiamos el IVA en un *textBox* para modificarlo en la base de datos automáticamente se cambie el valor del subtotal de todas las facturas que se visulen.
- **OneWayToSource**: Ésta es la opción opuesta a OneWay y con ella se actualiza automáticamente el origen tras modificar el destino. Esta opción es útil en casos especiales en los que la propiedad de destino no está visible en el panel Propiedades, lo que puede ocurrir si no se trata de una propiedad de dependencia. El enlace OneWayToSource: permite configurar el enlace de datos en el destino.
- **OneTime**: Provoca una única inicialización del origen al destino, pero los cambios posteriores en el origen no actualizan el destino.

Para finalizar comentar, para enlaces externos a datos, los dos **estándares más utilizados** por los desarrolladores de aplicaciones: el **ODBC** creado por Microsoft y **JDBC** para aplicaciones de JAVA. Ambas son estándares que permiten, **independientemente de la estructura**, crear una interfaz homogénea para el acceso a la base de datos. Permiten acceder a cualquier dato desde cualquier aplicación, sin importar qué sistema de gestión de bases de datos (DBMS) almacene los datos sin que el desarrollador tenga que preocuparse de cómo lo hace simplemente realiza la conexión.

Hablaremos de ellos en unidades posteriores.

→ Asociación de acciones a eventos.

La programación de aplicaciones para un GUI implicaron un cambio radical de filosofía y estructura a la hora de codificar los programas.

Un **programa tradicional** tiene una estructura “lineal”, con el código repartido en una serie de funciones u operaciones. Existe **una función u operación principal o main** donde comienza la ejecución y a partir de ahí se encadenan las llamadas de unas funciones a otras hasta que en un punto determinado acaba la ejecución

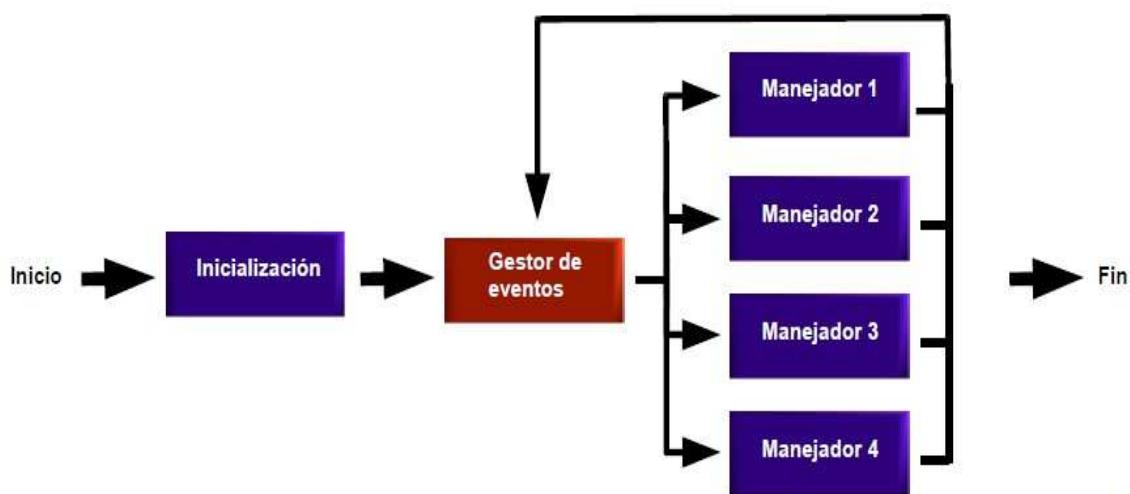
En cambio la programación en GUIs (Interfaz Gráfica de Usuario) esta orientada a **eventos**. La mayoría de los eventos son sucesos **asíncronos** producidos por la interacción del usuario con la aplicación, y están ligados a algún elemento de la interfaz. Algunos ejemplos son:

- Pulsar un botón
- Cambiar el tamaño de una ventana
- Mover una barra de desplazamiento
- Pulsar una tecla
- Tocar alguno de los botones minimizar/maximizar/cerrar de la ventana
- Hacer un click de ratón sobre un elemento determinado

Algunos eventos no relacionados directamente con el usuario y **generalmente síncronos** son:

- Aparición de una ventana
- “Tick” de un reloj programado con antelación

La mayor parte del código en un programa para un GUI está en los llamados **manejadores de eventos**. Cada manejador se encarga de realizar el conjunto de acciones asociadas a un evento determinado. Existe un **gestor de eventos** que se encarga de recibir todos los eventos de la aplicación y llamar al manejador adecuado.



La programación en GUIs suele ser un proceso iterativo de tres pasos:

- **Diseñar la interfaz** de una parte de la aplicación, utilizando los **widgets** disponibles en el toolkit o barra de herramientas de desarrollo e incluye a su vez dos tareas:
 - Posicionar los widgets y establecer sus dimensiones
 - Modificar sus características visuales y funcionales (títulos, colores, comportamiento)
- Realizar la **captura de los eventos** de la interfaz que permitan implementar la funcionalidad requerida
- **Implementar o codificar** cada uno de los manejadores correspondientes a los eventos capturados

Los IDEs actuales permiten realizar los dos primeros pasos de forma rápida.



Una vez realizado el diseño, los entornos de desarrollo eligen una de las siguientes estrategias para llevar a cabo la codificación de los eventos:

- Salvar el diseño y las propiedades en ficheros ocultos especiales que **son compilados**

junto al código (Delphi, C++ Builder, Visual Basic)



- Generar el código correspondiente de la interfaz que el desarrollador completa con su propio código.

Como acabamos de ver, en un programa para un GUI la tarea fundamental a realizar es **gestionar adecuadamente los eventos recibidos**.

La información asociada a un evento suele ser como mínimo **un campo indicador del tipo de evento y el identificador del elemento** que genera dicho evento (botón, ventana, etc.)

Un aspecto fundamental en el diseño de un toolkit o IDE es la forma en que se produce la **conexión entre el gestor de eventos y los manejadores**.

En lenguajes primitivos como C el programador realiza tanto la implementación del gestor de eventos sino la llamada a los distintos manejadores.

En lenguajes más modernos el gestor de eventos ya está integrado en el propio IDE y no es necesario implementarlo.

➔ Diálogos modales y no modales.

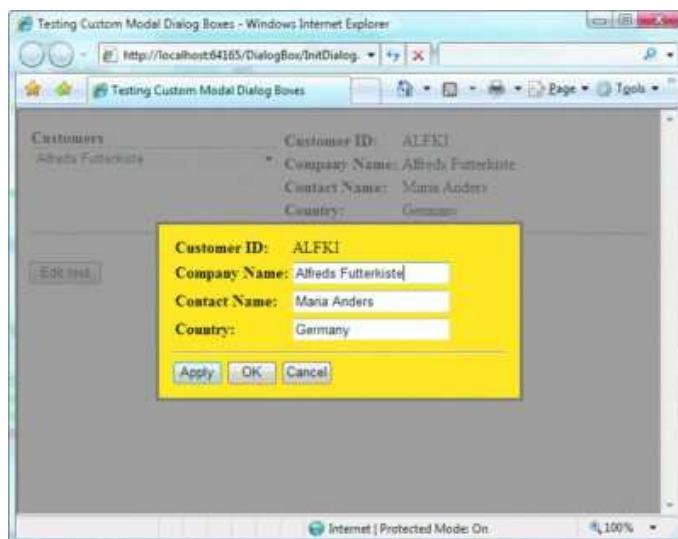
Las aplicaciones independientes tienen normalmente una **ventana principal**, que tanto muestra los datos principales sobre los que funciona la aplicación como expone la funcionalidad de procesamiento de datos a través de mecanismos de interfaz de usuario (UI) tales como barras de menús, barras de herramientas y barras de estado.

Una aplicación no trivial también puede mostrar ventanas adicionales para hacer lo siguiente:

- Mostrar información específica a los usuarios
- Recopilar información de los usuarios.
- Tanto mostrar como recopilar información.

Estos tipos de ventanas se conocen como **cuadros de diálogo** y hay dos tipos: **modales** y **no modales**.

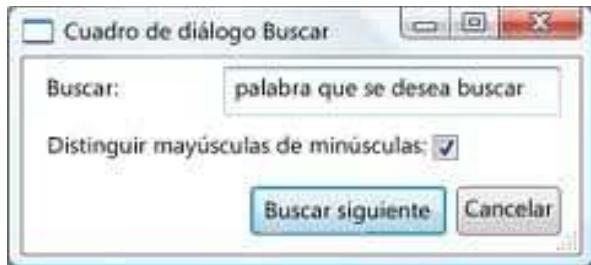
- Los cuadros de diálogo **modales** los muestra las funciones **cambiar necesitan datos adicionales de los usuarios** para continuar. Dado que la función depende del cuadro de diálogo modal para recopilar los datos, el cuadro de diálogo modal también impide que un usuario active otras ventanas de la aplicación mientras permanece abierto. En la mayoría de los casos, los cuadros de diálogo modales permiten a los usuarios señalar que han terminado con el cuadro de diálogo modal presionando un botón **Aceptar** o **Cancelar**. Al presionar el botón **Aceptar** se indica que el usuario ha introducido los datos y desea que la función continúe su proceso con esos datos. Presionar el botón **Cancelar** indica que el usuario desea detener la ejecución de la función. Los ejemplos más comunes de cuadros de diálogo modales se muestran para abrir, guardar e imprimir datos.



Cuadro diálogo modal

- Un cuadro de diálogo **no modal**, por otra parte, **no impide que el usuario active otras ventanas mientras está abierto**. Por ejemplo, si un usuario desea buscar apariciones de una palabra determinada en un documento, una ventana principal abrirá habitualmente un cuadro de diálogo para preguntar al usuario qué palabra está buscando. Dado que la búsqueda de una palabra no impide que un usuario edite el documento, no obstante, no es necesario que el cuadro de diálogo sea modal. Un cuadro de diálogo no modal proporciona

al menos un botón **Cerrar** para cerrar el cuadro de diálogo y puede proporcionar botones adicionales para ejecutar funciones concretas, como un botón **Buscar siguiente** para buscar la palabra siguiente que coincida con los criterios de una búsqueda de palabra.

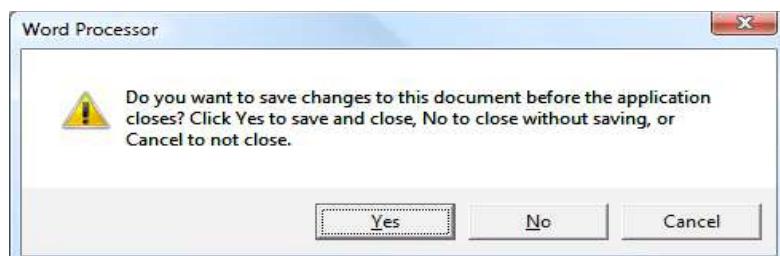


Cuadro diálogo no-modal

Entre los diferentes tipos de cuadros de diálogo los más comunes entre los diferentes IDE de programación nos podemos encontrar:

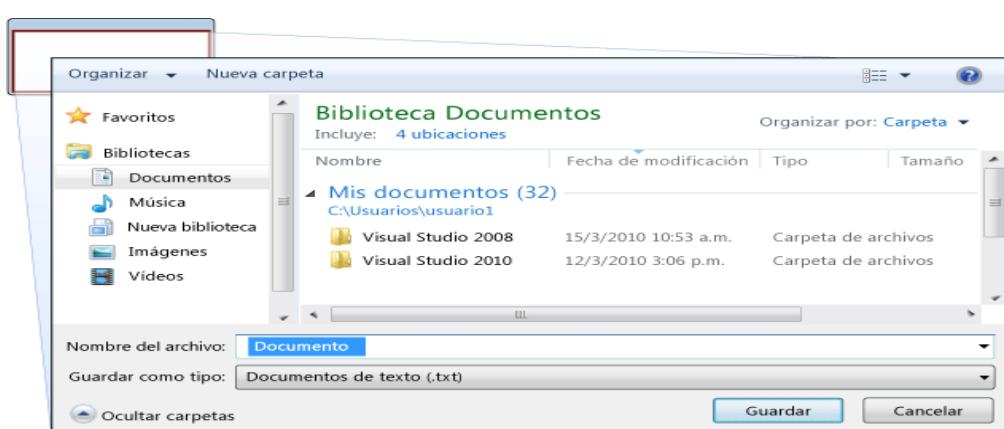
- **Cuadros de mensaje**

Un *cuadro de mensaje* es un cuadro de diálogo que se puede utilizar para mostrar información textual y permitirles que los usuarios tomen decisiones con botones. La figura siguiente muestra un cuadro de mensaje que muestra información textual, hace una pregunta y proporciona al usuario tres botones para responder a la pregunta.



- **Cuadros de diálogo comunes**

Aquí podemos encontrar la mayoría de los cuadros de diálogo más comunes y reutilizables por la mayoría de los IDE: *Imprimir*, *Abrir fichero*, *Guardar Fichero*,



- **Cuadros de diálogo personalizados**

A veces se hace necesario crear nuevos cuadros de diálogo debido a que los más comunes antes citados no cumplen las expectativas del programador.

Para finalizar señalar que un abuso del uso de cuadros de diálogo, preferentemente modales, **dan lugar a diseños confusos para los usuarios de la aplicación**. En lo posible **no deben llegar a un tercer nivel de cuadros de diálogo**, es decir, que desde la ventana principal de la aplicación no es aconsejable abrir más allá de tres cuadros de diálogo en modo jerárquico.

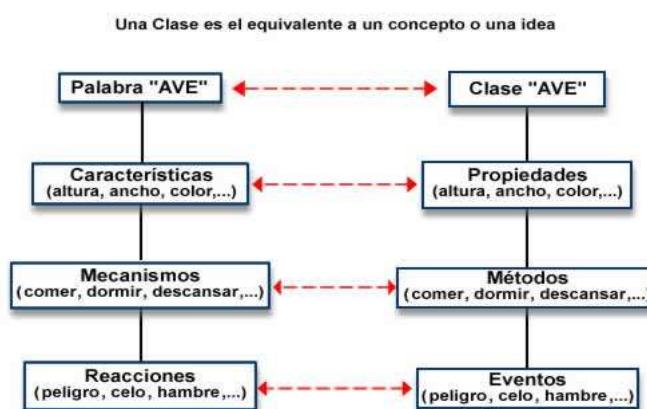
➔ Clases, propiedades y métodos

Una **clase** es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo iguales o con pequeñas variaciones. El modelo **describe el estado y el comportamiento** que todos los objetos de la clase comparten. Su comportamiento, **métodos**, y su estado, **atributos** se encapsulan.

Un **objeto** de una determinada clase se denomina una **instancia** de la clase. Una clase tiene tanto una interfaz y una estructura. La interfaz describe cómo interactuar con la clase y sus instancias con métodos, mientras que la estructura describe cómo los datos se dividen en atributos dentro de una instancia.

La funcionalidad de una clase se implementa mediante los **métodos**. Cuando se desea realizar una acción sobre un objeto se desencadenará un **evento**, se dice que **se le manda un mensaje** invocando a un método que realizará la acción.

Las **propiedades** son un tipo especial de métodos. Debido a que suele ser común que las variables miembro sean privadas para **controlar el acceso** y mantener la coherencia, surge la necesidad de permitir consultar o modificar su valor mediante pares de métodos: *GetVariable* y *SetVariable*.



Más técnicamente, una clase es un conjunto coherente que consiste en un tipo particular de **metadatos**. Los **metadatos** han cobrado gran relevancia en el mundo de Internet, por la necesidad de utilizar los metadatos para la clasificación de la enorme cantidad de datos. Además de la clasificación los metadatos pueden ayudar en las búsquedas. Por ejemplo, si buscamos un artículo sobre vehículos, este dato tendrá sus correspondiente metadatos clave adjuntos, como 4 ruedas, cuatro ruedas, motor, etc. Hoy en día, por ejemplo, es común codificar datos mediante **XML** así son legibles tanto para humanos como para computadores.

→ Eventos: escuchadores.

Una vez se han “dibujado” la interfaz gráfica en la pantalla la aplicación suele quedar a la espera sin ejecutar código. Cuando se acciona sobre la interfaz (click botón, escritura....) el escuchador correspondiente a uno de los componentes gráficos se activa.

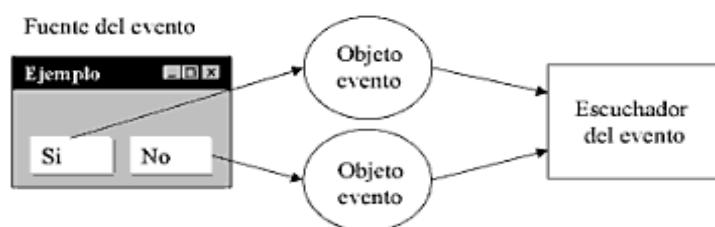
```
public interface IListener
```

```
{
```

```
public void Update(String eventType, Object oldValue, Object newValue);
```

```
}
```

En JAVA las clases escuchadoras de eventos se denominan o llevan el término *Listener*.



Suele ser en los escuchadores donde el programador incorpora el código específico para responder a los eventos generados por el usuario u otra instancia de la aplicación.

Como se ve en la figura los escuchadores son objetos que pueden recibir y manejar eventos enviados por otros objetos y para ello un escuchador debe:

- implementar las interfaces escuchadores de eventos
- registrarse en la fuente de eventos

Algunos de los escuchadores de java con respecto a sus objetos

| INTERFAZ | MÉTODOS |
|---------------------|--|
| ActionListener | actionPerformed(ActionEvent) |
| FocusListener | focusGained(FocusEvent) focusLost(FocusEvent) |
| TextListener | textValueChanged(TextEvent) |
| KeyListener | keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent) |
| MouseListener | mouseClicked(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) |
| MouseMotionListener | mouseDragged(MouseEvent) mouseMoved(MouseEvent) |
| WindowListener | windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent) |

Cuando un objeto genera un evento (fuente) realiza una llamada a determinados métodos definidos por el escuchador registrado en la fuente. Antes de iniciar la gestión del evento, la fuente precisa conocer si el escuchador ha implementado los métodos a llamar y eso se realiza mediante **la interfaz** del escuchador de eventos. Por ello el escuchador debe implementar los métodos de la interfaz del escuchador de aquellos eventos que esté interesado en escuchar.

La interfaz del escuchador tienen tantos métodos implementados como eventos puede manejar.

ACTIVIDADES

Para los ejercicios planteados usaremos la herramienta **Glade** para el diseño de las interfaces y el lenguaje **Python** para la implementación de los eventos. En la Red hay multitud de manuales de Python pero para el desarrollo de este manual recomiendo el siguiente ya que se ajusta en gran medida a lo que se pretende:

- *Python Para Todos.* Autor: Raúl Fernández Duque

Para el uso de la herramienta **Glade** no hay manuales propiamente dichos o bien son excesivamente anticuados. De todas formas su manejo es muy intuitivo. Aconsejaría el siguiente enlace:

- <http://developer.gnome.org/glade/index.html.es>. Desarrollado por el equipo de Gnome. Muy sencillo y a veces excesivamente esquemático. También el más actualizado y en castellano.
- <http://glade.gnome.org/> La propia web de Glade. Completa y en inglés

Ejemplo.

En primer lugar diseñamos mediante Glade el siguiente formulario. Los colores y tipo de fuentes no tienen porque ser los mismos, pero el contenido sí.



Es importante el uso de contenedores para la correcta distribución de los diferentes *labels* y *textbox*.

A continuación vamos a codificar en Python los siguientes eventos:

- *Aceptar*: nos mostrará una ventana nueva con los datos introducidos en las cajas de texto.
- *Cancelar*: cerrará la ventana.

Parte del código podría ser algo así:

```
#!/usr/bin/env python

import gtk
import pygtk

class formulario:

    def __init__(self):
        b = gtk.Builder()                                // inicializamos las gráficas
        b.add_from_file("hola.glade")
        self.window1 = b.get_object("window1")           // window1 es la ventana formulario
        b.connect_signals(self)
        self.window1.show()                            // se muestra window1

    def on_window1_destroy(self,widget,data=None):
        gtk.main_quit()                               // evento que cierra la ventana

formulario()                                         // llamamos a la clase
gtk.main()
```

Nos queda codificar el botón *Aceptar* cuya función sería mostrar una nueva ventana, *window2*, que contenga una serie de labels con el texto introducido en las cajas de texto.

Algunos recordatorios para evitar errores comunes:

- a. No olvidemos las tabulaciones en Python y errores de sintaxis en general
- b. Cuidado con el nombre de los objetos y de los eventos asociados a los mismos
- c. El fichero glade, en este caso, debe estar en el mismo directorio que el fichero python.

Propuesta 1. La elaboración de una calculadora sencilla con las cuatro operaciones matemáticas básicas.

Propuesta 2. Formulario de introducción de datos personales en el aparezca un Cuadro de Lista o ComboBox que permita elegir entre las provincias de tu comunidad. El DNI se introducirá el datos y calculará automáticamente la letra. Al pulsar Grabar se mostrará una ventana nueva que preguntará si está seguro o no.

Propuesta 3. A partir de la propuesta 2 lleva a cabo el grabado de los datos en una base de datos utilizando SQQLite o similar. Algunos trozos de código útiles podría ser:

```
#!/usr/bin/env python

import gtk
import pygtk
import sqlite3

bbdd = 'tEjercicio'
class mundo:
.....
# conexión a la base de datos si está en el mismo directorio que la aplicación

conex = sqlite3.connect(bbdd)
c = conex.cursor()

def listar(self):
    c.execute = ('SELECT * FROM tUsuario ORDER BY apellidos')
    ..... # ¿como los mostrarías
    c.close() # no olvidarse nunca de cerrar el cursor

def grabar_user(self, bbdd):
# pondré solo un par de campos
    u = window.txtUsuario.GetValue()
    c = window.txtPassword.GetValue()
    ...
    registro = (u,c, ...)
    try
        c.execute = ('INSERT INTO tUsuario (campo1,campo2,...) VALUES
(?,?,?,...), (registro)')
        c.commit() #muy importante
    except sqlite3.Error e:
        mensaje = "Se ha producido un error..."
        # aquí mostraríamos una ventana no-modal advirtiéndolo
        ...
        c.rollback() #por si acaso
    ...
mundo()
gtk.main()
```

UD2. Generación de interfaces gráficas de usuario empleando XML

Objetivos

RA2. Generar interfaces gráficas de usuario basados en XML utilizando herramientas específicas y adaptando el documento XML creado.

- CA2.1. Se reconocen las ventajas de generar interfaces gráficas de usuario a partir de su descripción en XML..
- CA2.2. Se genera la descripción de la interfaz en XML usando un editor gráfico.
- CA2.3. Se analiza el documento XML generado.
- CA2.4. Se modifica el documento XML.
- CA2.5. Se les asignaron acciones a los eventos.
- CA2.6. Se generó el código correspondiente a la interfaz a partir del documento XML.
- CA2.7. Se programó una aplicación sencilla para comprobar la funcionalidad de la interfaz generada.

Generación de interfaces gráficas de usuario empleando el lenguaje XML

- Lenguajes de descripción de interfaces basadas en XML: ámbito de aplicación.
- Elementos, etiquetas, atributos y valores.
- Herramientas libres e propietarias para la creación de interfaces de usuario multiplataforma.
- Controles: propiedades.
- Eventos: controladores.
- Edición y depuración del documento XML.
- Generación de código para diferentes plataformas.

→ Lenguajes de descripción de interfaces basadas en XML: ámbito de aplicación.

Un **lenguaje de marcado** o **lenguaje de marcas** es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto en su presentación.

A veces se confunde lenguaje de marcas con lenguaje de programación y es un error ya que el primero no contiene ni variables, ni expresiones aritméticas ni sentencias del tipo condicional o iterativo todo típico de un lenguaje de programación al uso.

Entre los lenguajes de marcas o de descripción más importantes y de gran influencia en el campo que nos ocupa, la generación de interfaces gráficas, se encuentra el **XML**².

XML no nació sólo para su aplicación en **Internet**, sino que se propone como un **estándar para el intercambio de información estructurada** entre diferentes plataformas. Se puede usar y se usa en bases de datos, editores de texto, hojas de cálculo e interfaces gráficas...

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Edit_Mensaje SYSTEM "Edit_Mensaje.dtd">

<Edit_Mensaje>
  <Mensaje>
    <Remitente>
      <Nombre>Nombre del remitente</Nombre>
      <Mail>Correo del remitente </Mail>
    </Remitente>
    <Destinatario>
      <Nombre>Nombre del destinatario</Nombre>
      <Mail>Correo del destinatario</Mail>
    </Destinatario>
    <Texto>
      <Asunto>
        Este es mi documento con una estructura muy sencilla
        no contiene atributos ni entidades...
      </Asunto>
      <Parrago>
        Este es mi documento con una estructura muy sencilla
        no contiene atributos ni entidades...
      </Parrago>
    </Texto>
  </Mensaje>
</Edit_Mensaje>
```

Ejemplo de Documento XML

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Este es el DTD de Edit_Mensaje --&gt;

&lt;!ELEMENT Mensaje (Remitente, Destinatario, Texto)*&gt;
&lt;!ELEMENT Remitente (Nombre, Mail)&gt;
&lt;!ELEMENT Nombre (#PCDATA)&gt;
&lt;!ELEMENT Mail (#PCDATA)&gt;
&lt;!ELEMENT Destinatario (Nombre, Mail)&gt;
&lt;!ELEMENT Nombre (#PCDATA)&gt;
&lt;!ELEMENT Mail (#PCDATA)&gt;
&lt;!ELEMENT Texto (Asunto, Parrago)&gt;
&lt;!ELEMENT Asunto (#PCDATA)&gt;
&lt;!ELEMENT Parrago (#PCDATA)&gt;</pre>

```

Documento DTD

2 Ya que en 1º de DAM existe un módulo dedicado al lenguaje de marcas no entraremos en detalle de sus características.

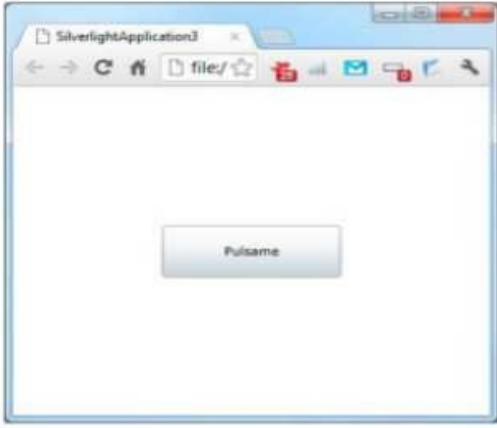
Entre los **lenguajes de descripción** de interfaces basados en XML más importantes (no están todos ya que algunos aún se están desarrollando) tenemos:

- **UIML (User Interface Markup Language)**: es un sencillo lenguaje basado en XML que permite realizar una descripción declarativa de la interfaz de usuario de un modo **independiente del dispositivo**. Para describir una interfaz de usuario en UIML se debe realizar, por un lado la **definición de la interfaz genérica**, y por otro un documento UIML que **representa el estilo de presentación apropiado para el dispositivo** en el cuál la interfaz de usuario se va a ejecutar. De este modo, una misma aplicación solamente necesitará un único documento UIML de especificación válido para cualquier dispositivo y un documento de estilo propio para cada dispositivo.
- **XIML (eXtensible Interface Markup Language)** es un lenguaje de especificación basado en XML. Se propone como lenguaje de especificación común e infraestructura de desarrollo para profesionales de la interfaz de usuario en todos los ámbitos, diseñadores, ingenieros de software o expertos en usabilidad.
- **MXML (Macromedia)** es un lenguaje descriptivo desarrollado inicialmente por Macromedia hasta el 2005 para la plataforma FLEX de Adobe. MXML se basa en XML y su acrónimo "Macromedia eXtensible Markup Language". Es un lenguaje que describe interfaces de usuario, crea modelos de datos y tiene acceso a los recursos del servidor, del tipo RIA (Rich Internet Application). MXML tiene una mayor estructura en base a etiquetas, similar a HTML, pero con una sintaxis menos ambigua, proporciona una gran variedad e inclusive permite extender etiquetas y crear sus propios componentes. Una vez compilado genera **ficheros .swf**.
- **GladeXML-GtkBuilder** más que un lenguaje es una herramienta de desarrollo visual de interfaces gráficas mediante GTK/GNOME. Es independiente del lenguaje de programación y no genera código fuente sino un archivo **XML**. El **IDE Anjuta** lo lleva integrado para el desarrollo de interfaces pero puede trabajar independientemente. Actualmente está en la versión Glade3 que fue reescrita totalmente aumentando el número de widgets y haciéndolo más ligero. **GtkBuilder** es un formato XML que Glade usa para almacenar los elementos de las interfaces diseñadas. Estos archivos pueden emplearse para construir las interfaces en tiempo de ejecución mediante el objeto GtkBuilder de GTK+. **GladeXML** era el formato que se usaba en conjunto con la biblioteca *libglade* (ambos obsoletos en favor de GtkBuilder).
- **XAML (eXtensible Application MarkupLanguage)** es el lenguaje de formato para la interfaz de usuario para la **Base de Presentación de Windows** y **Silverlight**, el cual es uno de los

"pilares" de la interfaz de programación de aplicaciones .NET en su versión 3.0 (conocida con anterioridad con el nombre clave WinFX).

XAML – Controles

- Elementos de interfaz de usuario (UI) reutilizables
- <Button x:Name="MyButton" Content="Pulsame" Width="150" Height="50" />



XAML es un lenguaje declarativo optimizado para describir gráficamente interfaces de usuarios visuales ricas desde el punto de vista gráfico, tales como las creadas por medio de Adobe Flash. En su uso típico, los archivos tipo XAML serían producidos por una herramienta de diseño visual, como Microsoft Visual Studio. El XML resultante es interpretado en forma instantánea por un subsistema de despliegue de los sistemas Windows a partir del Vista que reemplaza al GDI de las versiones anteriores de Windows. Los elementos de XAML se interconectan con objetos del Entorno Común de Ejecución. Los atributos se conectan con propiedades o eventos de esos objetos.

XAML fue diseñado para soportar las clases y métodos de la plataforma .NET que tienen relación con la interacción con el usuario, en especial el despliegue en pantalla.

Un archivo XAML puede ser compilado para obtener un archivo binario XAML **.baml**, el cual puede ser insertado como un recurso en un ensamblado de Framework .NET. En el momento de ejecución, el motor del Framework extrae el archivo .baml de los recursos del ensamblado, se analiza sintácticamente, y crea el correspondiente árbol visual WPF o Workflow. Cuando se use en **Windows Presentation Foundation**, **XAML es usado para describir interfaces visuales para usuarios**. WPF permite la definición de objetos en 2D y 3D, rotaciones, animaciones y otra variedad de características y efectos. Cuando es usado en el contexto de **Windows Workflow Foundation**, **XAML es usado para**

describir lógica declarativa, como aquellos creados en el proceso de sistemas de modelado y herramientas. El formato de serialización para WorkFlows había sido llamado previamente XOML, para diferenciarlo de su uso en IU de los XAML, pero esa diferenciación ya no existe. Sin embargo las extensiones de los archivos que contienen marcado de workflow es todavía XOML.

Este ejemplo en XAML muestra un texto "Hola Mundo!" dentro de un contenedor del tipo Canvas.

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <TextBlock>Hola Mundo!</TextBlock>
</Canvas>
```

- **Xforms** es un formato XML diseñado por el W3C para poder definir interfaces de usuario, **principalmente formularios web**. XForms fue diseñado para la nueva generación de formularios HTML/XHTML, pero es lo suficientemente genérico como para que pueda ser usado, de una manera independiente, para describir cualquier interfaz de usuario e incluso para realizar tareas simples y comunes de manipulación de datos.

Desde el de 2009 y hasta hoy en día la recomendación oficial del W3C es XForms 1.1. Existen varios plugins y extensiones que le dan soporte a diferentes navegadores. Firefox soporta XForms a través de una extensión. Para IE6 está *formsPlayer*, un plugin que extiende al navegador haciendo que soporte XForms, DOM 2 Events, DOM 3 XPath, XML Events y el DOM 3 Implementation Registry. La versión 2.0 y superiores de Openoffice.org soportan Xforms.

XForms también puede ser usado a través de varias tecnologías de servidor que convierten el código de XForms a formularios de HTML en tiempo de ejecución y de manera transparente. Recientemente ha sido dada a conocer una nueva herramienta, llamada AJAXForms, que transforma, en tiempo de compilación, documentos XHTML/XForms en páginas HTML con Javascript, que sí entienden los navegadores actuales. Estas páginas gestionan, sin interactuar con el servidor, tanto la presentación como la lógica de la interfaz de usuario y su comunicación con el servidor se restringe al intercambio de datos utilizando técnicas AJAX.

XSLTForms transforma documentos XHTML/XForms en páginas XHTML con JavaScript en los navegadores con XSLT.

- ➔ **XUL** (*XML-based User-interface Language*, es la aplicación de XML a la descripción de la interfaz de usuario en el navegador Mozilla. No es un estándar. Destaca por su portabilidad

y su independencia de dispositivo. Provee un gran conjunto herramientas para crear menús, paneles, barras de herramientas, wizards, entre otras. Gracias a esto, no será necesario utilizar un lenguaje de programación propietario o incluir un gran código JavaScript para manejar el comportamiento de la interfaz de usuario.



XUL – Barra de progreso

```
<?xml version='1.0'?>
<window
  id="hello"
  title="Progressmeter"
  width="400"
  height="300"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<vbox style="padding:16px">
  <hbox width="400" >
    <progressmeter mode="determined" id="mpm" value="0" />
    <button label="Switching"
      onmouseover = "setLoading(true)"
      onmouseout = "setLoading(false)" />
  </hbox>
</vbox>

<script>

  function setLoading(state)
  {
    var pm = document.getElementById('mpm');
    if(state)
      pm.mode = 'determined';
    else
      pm.mode = 'undetermined';
  }

</script>
```

Una interfaz XUL es definida mediante la especificación de tres grupos de componentes distintos:

- **Content:** Aquí se encuentran los documentos XUL, que definen el diseño de la interfaz.
- **Skin:** Contiene las hojas de estilos (CSS) y las imágenes, las cuales definen la apariencia de la interfaz.
- **Locale:** Los documentos DTD se encuentran aquí, estos documentos facilitan la localización de páginas XUL.

Muchas de las interfaces desarrolladas para las extensiones de Mozilla Firefox son basadas en XUL.

➔ **XBL (eXtensible Bindings Language)** es un lenguaje de marcas que se emplea para definir el comportamiento y la apariencia de aplicaciones XUL y elementos XML. El lenguaje XUL

define la disposición de la interfaz de usuario de una aplicación, que puede adoptar diferentes aspectos dependiendo del estilo definido. Sin embargo resulta imposible definir cómo funciona cada elemento, como por ejemplo, la forma en que funcionan una barra de progreso. Es aquí donde entra en juego el lenguaje XBL.

Un archivo XBL contiene asociaciones (***bindings***). El elemento raíz de todo documento XBL es **<bindings>**, que contiene a su vez uno o varios elementos **<binding>**. Cada uno de estos últimos declara un *binding* que puede asignarse a cualquier elemento XUL. La forma de realizar esta asignación es a través de las hojas de estilo: la propiedad **-moz-binding** del elemento XUL debe indicar la URL del documento XBL.

```
scrollbar {
    -moz-binding: url('somefile.xml#binding1');
}
```

➔ Elementos, etiquetas, atributos y valores.

Existen tres términos comúnmente usados para describir las partes de un documento XML: **etiquetas, elementos y atributos**. Aquí está un documento de ejemplo que ilustra estos términos:

```
<direccion><datos>
<titulo>Mrs.</titulo>
<nombre> Mary </nombre>
<apellidos> McGoon </apellidos> </datos>
<calle> 1401 Main Street </calle>
<ciudad estado="NC">Anytown</ciudad>
<codigo-postal> 34829 </codigo-postal>
</direccion>
```

Una **etiqueta** es un texto entre el símbolo menor que (<) y el símbolo mayor que (>). Existen etiquetas de **inicio** (como **<nombre>**) y etiquetas de **fin** (como **</nombre>**).

Un **elemento** consta de la etiqueta de inicio, la etiqueta de fin y de todo aquello que este entre ambas. En el ejemplo anterior, el elemento **<datos>** contiene tres elementos hijos: **<titulo>**, **<nombre>**, y **<apellidos>**.

Un **atributo** es un par **nombre-valor** dentro de la etiqueta de inicio de un elemento. En este ejemplo, estado es un atributo del elemento <ciudad>, en ejemplos anteriores <estado> era un elemento.

```
<html xmlns="http://www.w3.org/2002/06/xhtml2" xml:lang="sp">
  <head> <title>Busqueda</title>
  <model>
    <submission action="http://www.ejemplo.org/busca" method="get" id="busca"/></model>
  </head>
  <body>
    <p></p>
    <input ref="cad"><label>Cadena : </label></input>
    <submit submission="busca"><label>Buscar</label></submit>
  </body>
</html>
```

Ejemplo de Xforms (Formulario basado en XML)

➔ Herramientas libres y propietarias para la creación de interfaces de usuario multiplataforma.

Las herramientas de diseño de interfaces gráficas se encuadran dentro de las denominadas **RAD (Rapid Application Development)**.

En nuestro caso usaremos **Glade**, de software libre, se considera una herramienta de diseño de interfaces pura ya que ésta es su única función a diferencia de otras como Visual Basic que además soporta la codificación de código. Posteriormente podemos enlazarla con aquellos lenguajes de programación que deseemos. Otras como **Gambas**, **Mono**, **KDevelop**, **Lazarus** y **Anjuta**, también de software libre, incluyen además las herramientas necesarias para elaborar el código. En Java podemos citar a **Eclipse** y **NetBeans** ambas de software libre aunque estos son compatibles con otros lenguajes añadiéndoles los plugins correspondientes.

Comparación

| | Eclipse | Neatbeans | Jcreator | Mono | Sharp Develop | MS-Visual Studio |
|----------|-----------------------------|------------------|-------------------------|------------------|------------------|------------------------------------|
| S.O. | Multi-plataforma | Multi-plataforma | Windows | Multi-plataforma | Multi-plataforma | Windows |
| Licencia | Licencia Pública de Eclipse | CDDL | Privativo | GPL, LGPL y MIT | GPL, LGPL y MIT | Privativo |
| Uso | IDE java, c++, etc | IDE java | IDE java | C#, java | C#, .NET | C#, .NET |
| Precio | Gratis | Gratis | 1x\$89 30x\$1600 USD | Gratis | Gratis | Standar x \$299 Pro x \$799 USD |

De software privativo tenemos a **Visual Studio**, **Velneo**, **Jcreator**, **Delphi** y **Oracle** como las más destacadas y todas ellas son **IDE** que incluyen el software de desarrollo de código además de la interfaz de usuario.

Comparación

| | Aptana Studio | Adobe Dream-weaver | Nova Mind | FreeMind | Flash Develop | Adobe Flash |
|----------|----------------------------|---|----------------------------|------------------|------------------|----------------------|
| S.O. | Multi-plataforma | Windows | Windows, MacOS | Multi-plataforma | Multi-plataforma | Windows |
| Licencia | Licencia Pública de Aptana | Privativo | Privativo | GPL | MIT/X11 | Privativo |
| Uso | PHP, HTML, AJAX, etc | PHP, HTML, AJAX, etc | Mapas mentales | Mapas mentales | Action script | Flash, Action script |
| Precio | Gratis Pro x 1 \$99 USD | CS4x1 \$399 Creative Suite x1 \$1699 | 1x \$681 5x \$20720 USD | Gratis | Gratis | CS4 x 1 \$699 USD |

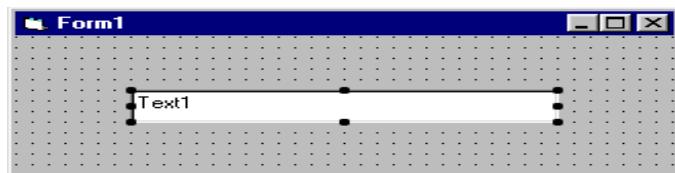
→ Controles: propiedades

Todos los controles de los que disponen los diferentes **Entorno de Diseño de Interfaces** disponen de una serie de propiedades las cuales podemos cambiar al incluirlos en las aplicaciones. Ejemplos de propiedades son el color, el tipo de letra, el nombre, el texto, etc... Solo la práctica nos permite conocer todas sus posibilidades.

A continuación mencionaremos los controles más generales y usuales que aparecen en los interfaces de usuario:

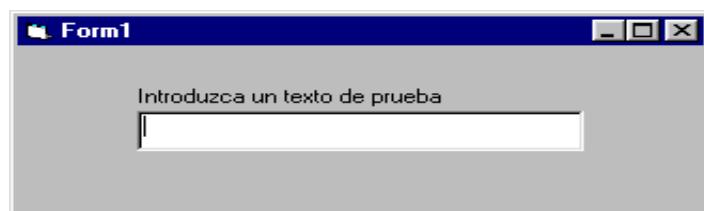
- **Textbox (entry en Glade)**

Mediante este control podremos realizar tanto la entrada como la salida de datos en nuestras aplicaciones.



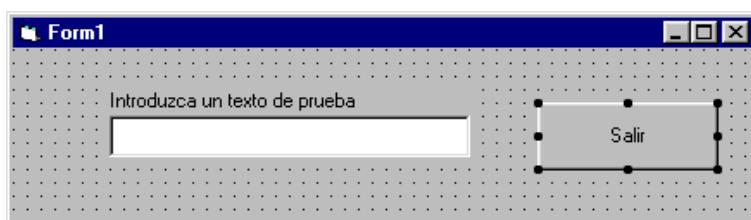
- **Label**

Este control es también uno de los más utilizados, aunque su utilidad queda restringida a la visualización de datos en el mismo, no permitiendo la introducción de datos por parte del usuario. Lo que si puede hacer en algún caso es mostrar datos cargados.



- **CommandButton o Button**

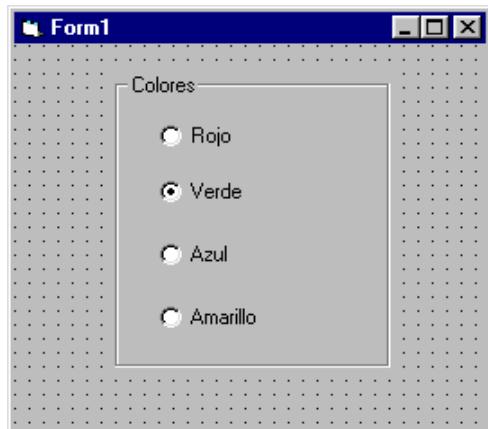
Este control es el típico botón que aparece en todas las aplicaciones y que al hacer click sobre él nos permite realizar alguna operación concreta, normalmente *Aceptar* o *Cancelar*. Aunque según el código que le asociemos podremos realizar las operaciones que queramos.



- **OptionButton o Option**

Este control nos permite elegir una opción entre varias de las que se nos plantean. Cada opción será un control *optionbutton* diferente. De todas las opciones que se nos ofrece, en este caso los 4 colores, sólo podremos activar una. Si activamos cualquier otra opción, se desactivará automáticamente la última que teníamos activada.

Facilita la introducción de datos por parte del usuario:



El marco que está alrededor de los 4 controles optionbutton se trata del control **Frame**, es opcional, aunque es conveniente colocarlo siempre que hagamos uso de las opciones. No sólo por motivos estéticos sino porque de esta manera podremos establecer grupos de controles *optionbutton* independientes en los que en cada grupo sólo pueda haber una opción activada a la vez. También, al mover el marco se moverán los controles incluidos en él facilitándonos las modificaciones. En Glade sería los **Container**.

- **Combobox**

Permite la selección de un lista que aparece al desplegarla.



Hay muchos más controles dependiendo del entorno de desarrollo, pero si analizamos cualquier aplicación en su mayoría la presencia de estos es mayoritaria. A lo largo de las prácticas veremos más.

→ Eventos: controladores

La programación orientada a eventos o evento de programación basado en un paradigma de programación en el que el flujo del programa es **determinada por los acontecimientos**, por ejemplo, salidas de los sensores o las acciones del usuario o mensajes de otros programas o hilos.

Para eventos “anormales” como un error de hardware, desbordamiento o "errores del programa" puede ocurrir que posiblemente impide su posterior procesamiento y para ello tenemos **los manejadores de excepciones** para dar una solución a la aplicación dando una salida que impida su bloqueo (un mensaje de vuelta atrás, un botón de salida....)

El primer paso en el desarrollo de un programa orientado a eventos es escribir una serie de subrutinas, o métodos, **llamados rutinas de controlador de eventos**. Estas rutinas manejan los eventos a los que el programa principal responderá. Por ejemplo, un solo botón izquierdo del ratón y haga clic en un botón de comando en un programa de interfaz gráfica de usuario puede activar una rutina que se abrirá otra ventana, guardar los datos en una base de datos o salir de la aplicación. Muchos entornos de programación de hoy en día proporcionan al programador plantillas de eventos para que el programador sólo necesita suministrar el código de evento.

El segundo paso es **enlazar controladores de eventos a los eventos** para que el funcionamiento correcto se llama cuando tiene lugar el evento. Editores gráficos se combinan los dos primeros pasos: *Haga doble clic en un botón*, y el editor crea un controlador de eventos asociados con el usuario hace clic en el botón y se abre una ventana de texto para que puedas editar el controlador de eventos.

El tercer paso en el desarrollo de un programa orientado a eventos es **escribir el bucle principal**. Esta es una función que comprueba la ocurrencia de los hechos, y luego llama al controlador de eventos correspondiente a procesarlo. La mayoría de los entornos de programación orientados a eventos ya ofrecen este bucle principal, por lo que no serán necesarios específicamente por el programador de la aplicación.

La programación controlada por eventos es **ampliamente utilizado en las interfaces gráficas** de usuario, ya que ha sido adoptado por la mayoría de los entornos de desarrollo para como modelo para la interacción.

Como hemos visto entonces un **evento** es un **suceso que ocurre en el entorno donde se ejecuta una aplicación** y para su atención se debe programar en ella un **método o controlador** que controle las acciones necesarias para responderle a cada tipo de evento que se necesite. A

diferencia de la programación secuencial, batch por lotes o tipo consola (en la que el programador exige una secuencia estricta de aparición de los eventos), **no se sabe cual será el orden de aparición** de ellos pues normalmente son el usuario o el sistema operativo quienes los generan según su intención y necesidades. La aplicación debe adoptar, entonces, la forma de intérprete o de máquina de estados para que esté atenta a reaccionar según sea el caso.

El sistema operativo ya existe pero debe invocar un método (enviar un mensaje a la aplicación) que solo ahora la persona programa. Por ello el programador debe hacer dos cosa:

- **Elaborar el método o algoritmo** (mensaje que la aplicación entenderá) que se ejecutará ante la aparición de un tipo de evento.
- Inscribir ese método como un **controlador o manejador** dentro de la aplicación para que el sistema operativo sepa qué invocar.

Para lo primero debe seguirse un protocolo o firma del método que case con lo que ya existe programado en el Sistema Operativo. Constará de un nombre y los parámetros de entrada pues él enviará información a la aplicación sobre el control en el que ocurrió el evento y un paquete con datos relevantes a ese tipo de evento. La forma de los métodos manejadores de eventos será, entonces:

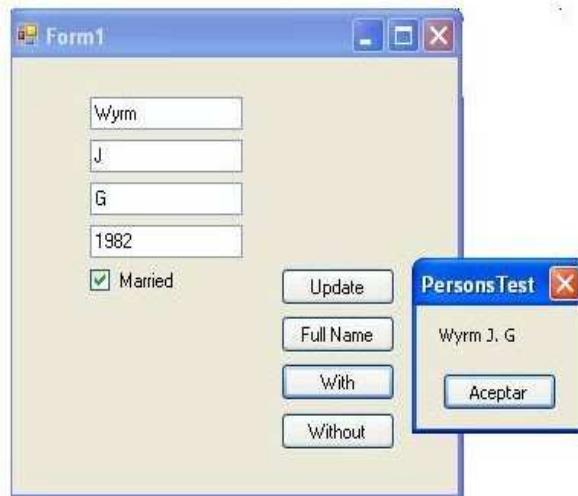
```
nombre_del_método(deme NC, deme detalles es un TipoDeArgumentosDeEvento)
{
    /* ...aquí irá la lógica del control y puede utilizar la información enviada en detalles
       por el Sistema Operativo....(el TipoDeArgumentosDeEvento que se recibe varía
       dependiendo del tipo de evento) */
}
```

Para lo segundo, debe instruir al Sistema Operativo sobre el control donde interesa la ocurrencia del evento, el tipo de evento y el método que lo atenderá, mediante el uso del método llamado **manejador**, así por ejemplo cuando exista el botón *BOT* en la aplicación *Aplicación* y se desee atender el evento *Click* con el algoritmo *MiAlgoritmo* se programará en su constructor:

Hay muchos eventos reconocidos por el sistema operativo y algunos de los más utilizados son:

- *Paint*: Pintura se genera cada vez que el S.O. detecta la necesidad de restaurar la imagen de un formulario
- *KeyDown*: Tecla Presionada por el usuario (o simulada por algún programa)
- *MouseDown*: Un botón del ratón ha sido presionado (o simulado).
- *Tick*: Un temporizador ha generado un aviso o alarma
- *MouseUp*: Un botón del ratón ha sido liberado (o simulado).
- *MouseMove*: El ratón ha sido movido (o simulado).

Resumiendo un controlador de eventos es un **procedimiento del código** que determina qué acciones se ejecutan cuando se produce un evento, como cuando un usuario hace clic en un botón o una cola de mensajes recibe otro mensaje. Cuando se produce un evento, se ejecuta el controlador o controladores de eventos que reciben dicho evento. Los eventos pueden asignarse a múltiples controladores, y los métodos que controlan determinados eventos pueden modificarse de manera dinámica.



→ Edición y depuración del documento XML.

La **validación XML** (*eXtensible Markup Language*) es la **comprobación** de que un documento en lenguaje XML **está bien formado** y se ajusta a una estructura definida. Un documento bien formado sigue las reglas básicas de XML establecidas para el diseño de documentos. Un documento válido además respeta las normas dictadas por su **DTD (definición de tipo de documento) o esquema utilizado**.

La validación se encarga de verificar:

- **La corrección de los datos.** Aunque validar contra un esquema no garantiza al 100% que los datos son correctos, nos permite detectar formatos nulos o valores fuera de rango y por tanto incorrectos.
- **La integridad de los datos.** Al validar, se comprueba que toda la información obligatoria está presente en el documento.
- **El entendimiento compartido de los datos.** A través de la validación se comprueba que el emisor y receptor perciban el documento de la misma manera, que lo interpreten igual.

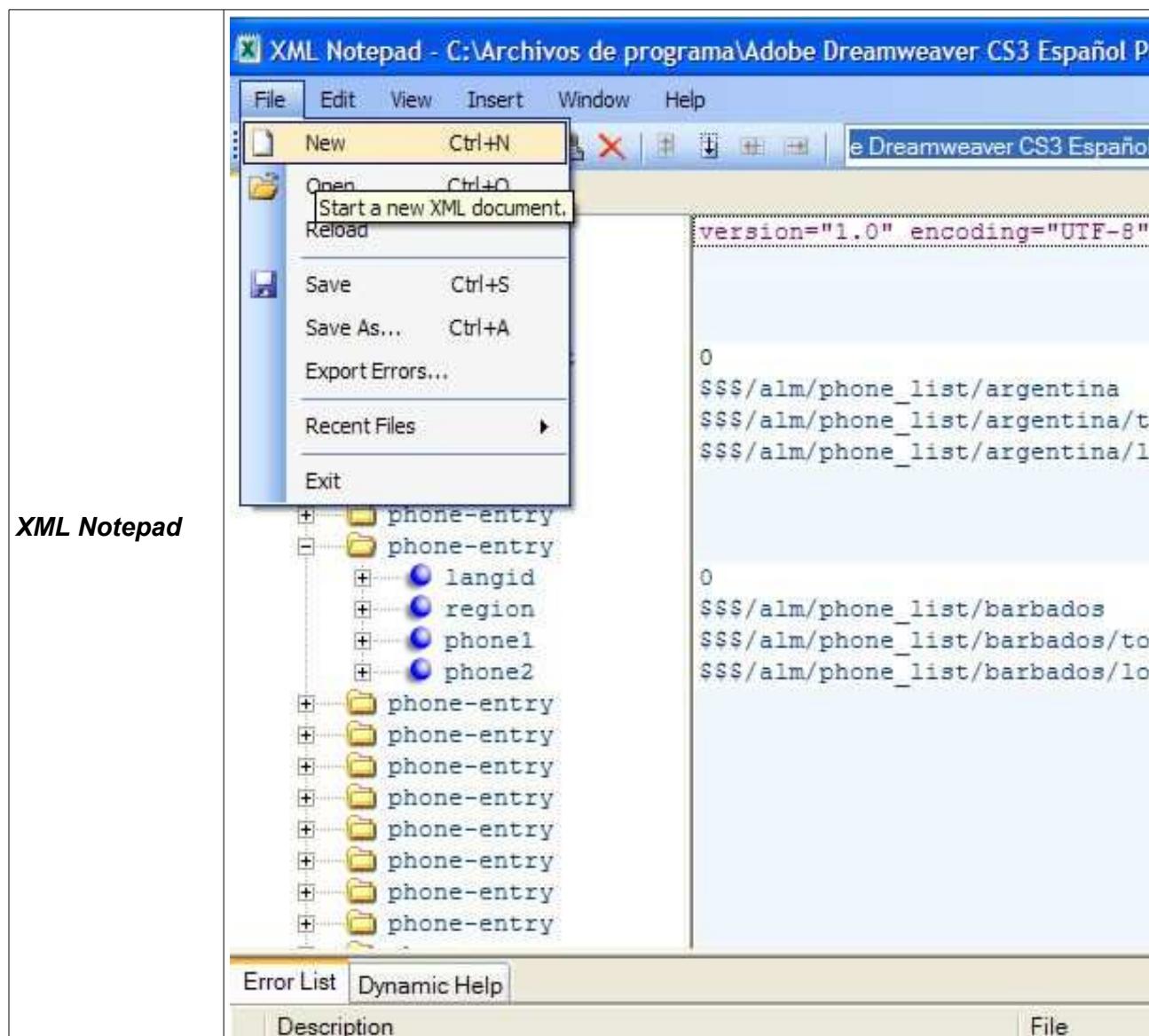
La creación manual de documentos XML e incluso su manipulación pueden introducir todo tipo de errores, tipográficos, sintácticos y de contenido. Existen editores de XML que facilitan la tarea de crear documentos válidos y bien formados, ya que pueden advertir de los errores básicos

cometidos e incluso escribir automáticamente la sintaxis más sencilla necesaria.

Cuando necesitamos obtener un documento válido, el editor XML ha de ser capaz de:

- Leer la DTD del documento y presentar una lista desplegable con los elementos disponibles enumerados en la [DTD](#), evitando así la inclusión de algún elemento no definido en el esquema.
- Advertir el olvido de una etiqueta obligatoria e incluso no permitir este tipo de descuidos o errores, no dando por finalizado el documento si existen errores de este tipo.

Algunos editores:



XML Spy de Altova

TheAgency.xsd

PersonalData

IdentificationType

Contact

E-Mail

Phone

Identification

PhotoURL

ReferenceCode

Clerkment

bookType

authorName

first-name

last-name

genre

price

publicationdate

ISBN

Properties

Cardinality

Content

Facets

Properties

Name

Design Split Source

Ready

Liquid XML

BookStore.xsd*

Properties

Cardinality

Content

Facets

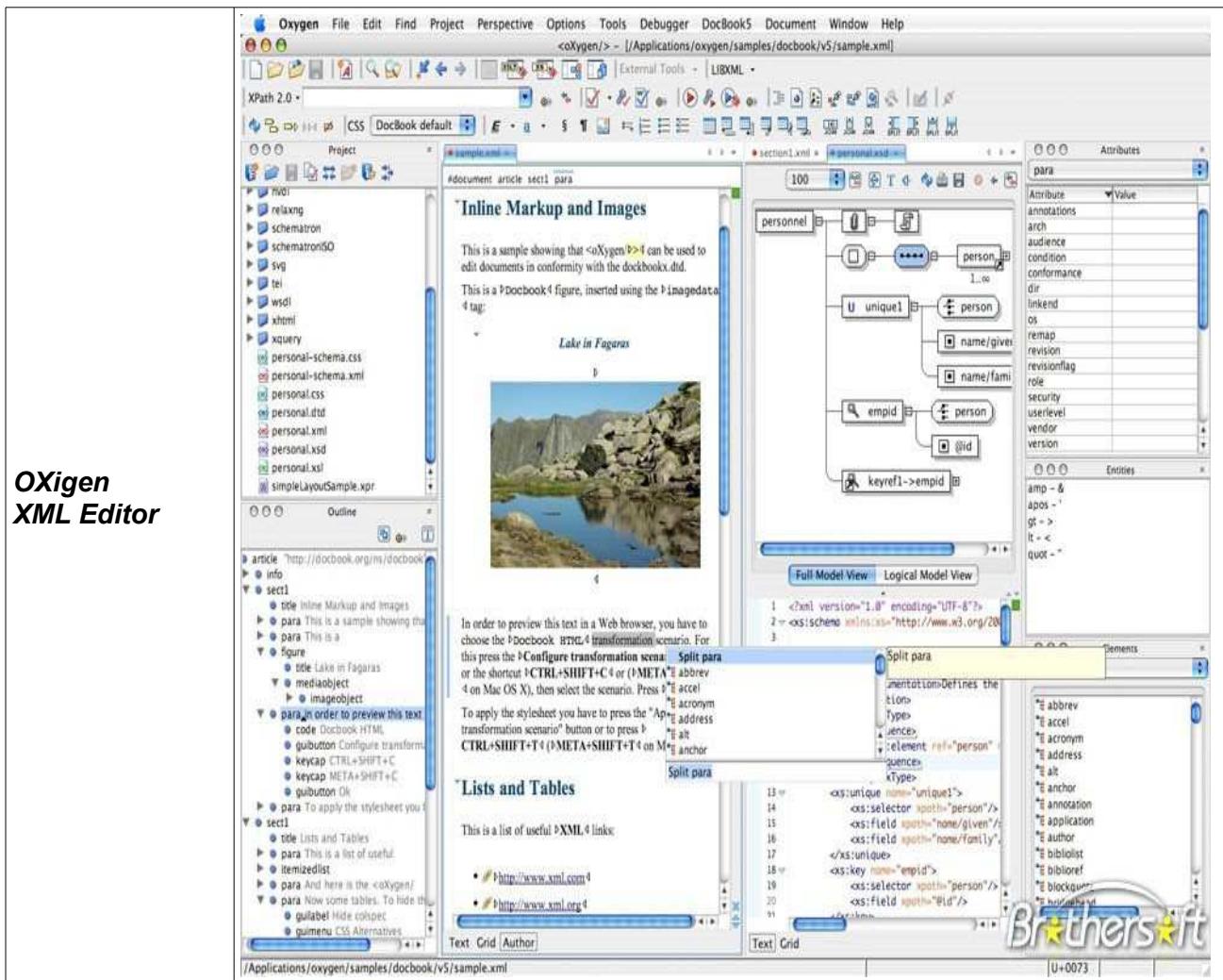
Properties

Name

```

<xsd:simpleType>
</xsd:simpleType>
<xsd:element>
<xsd:element name="author" type="bs:authorName" />
<xsd:element minOccurs="0" name="genre">
<xsd:simpleType>
<xsd:restriction base="xsd:string">

```



A partir de la propia búsqueda del programador de aquellos errores en un documento XML, cabe la posibilidad de usar herramientas para la depuración y optimización del código. Los editores de XML citados como Altova, Macromedia, Exchanger XML Editor... y cualquier editor XML medianamente profesional permiten depurar el código XML. Otros ejemplos:

- **xmllint** es una aplicación, analizador de XML basada en la consola, diseñada para imitar xmllint en Windows. Actualmente sólo tiene el formato xmllint implementado. El programa xmllint analiza uno o más archivos XML, especificados en la línea de comandos como archivo_xml. Imprime diversos tipos de salida, dependiendo de las opciones seleccionadas. Es útil para detectar errores tanto en el analizador XML propio como en el código XML.
- **XML Cleaner** es un filtro de validación para ficheros XML, muy similar a xmllint. Nos permite eliminar declaraciones *namespace* superfluas de un fichero XML, nos da elección entre varios tipos de formularios y etiquetas vacias, ofrece un validador XML, soporta Unicode y permite eliminar los comentarios y espacios en blanco. Actualmente ha dejado

de desarrollarse. Ejemplo de ejecución:

```
c:\> xmllint example.xml
```



The screenshot shows a terminal window with the title "xmllint output". The content of the window is as follows:

```

xmllint output

error: Document labelled UTF-16 but has UTF-8 content
<?xml version="1.0" encoding="UTF-16"?>
^

error: Opening and ending tag mismatch: LayoutCatalog line 13484 and Group
</Group></Group></Group></LayoutCatalog>
^

error: Opening and ending tag mismatch: File line 3 and Group
</Group></Group></Group></LayoutCatalog>
^

error: Opening and ending tag mismatch: FMPReport line 2 and LayoutCatalog
</Group></Group></Group></LayoutCatalog>
^

error: Extra content at the end of the document
<ValueListCatalog>
^

```

Ejemplo de Salida de xmllint

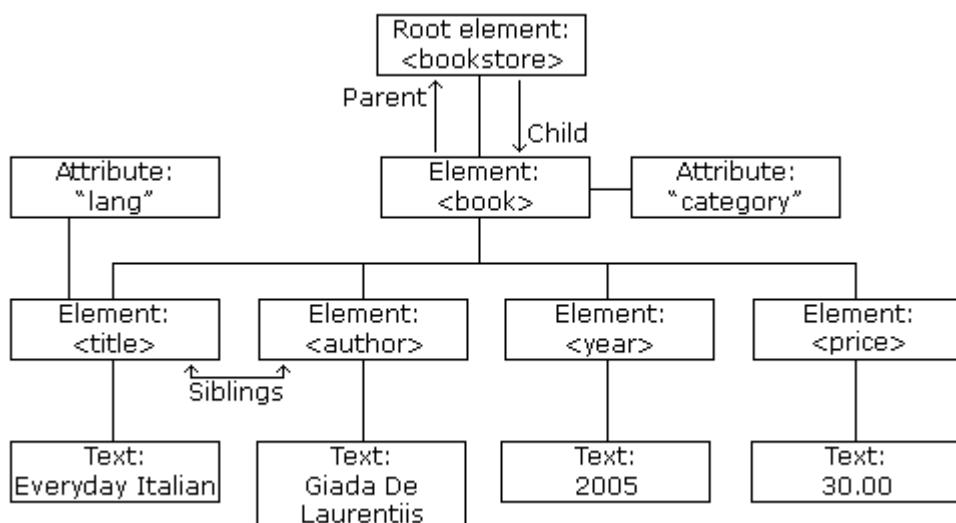
Para finalizar comentar que los documentos XML **se procesan a través de analizadores**, aplicaciones que leen el documento, lo interpretan y generan una salida basada en sus contenidos y en la marca utilizada para su descripción. El resultado se muestra en un dispositivo de visualización, como una ventana de navegación o una impresora. Los procesadores hacen posible la presentación y distribución de documentos XML.

Una aplicación que consume información XML debe leer un fichero de texto codificado según dicho estándar, cargar la información en memoria y, desde allí, procesar esos datos para obtener unos resultados (que posiblemente también almacenara de forma persistente en otro fichero XML).

El proceso anterior se enmarca dentro de lo que se conoce en informática como “**parsing**” o **análisis léxico-sintáctico**, y los programas que lo llevan a cabo se denominan “**parsers**” o **analizadores (léxico-sintácticos)**. Más específicamente podemos decir que el “**parsing XML**” es el proceso mediante el cual se lee y se analiza un documento XML para comprobar que está bien formado para, posteriormente, pasar el contenido de ese documento a una aplicación cliente que necesite consumir dicha información.

Hay dos tipos de analizadores para documentos XML: **analizadores dirigidos por la estructura (parsers DOM); y analizadores orientados a eventos (parsers SAX)**.

- **DOM:** XML es un metalenguaje de **estructura jerárquica**: las marcas dentro de un documento XML tiene una relación padre-hijo estricta. Esta estructura lleva a que la representación natural para documentos de este tipo sea la denominada (en informática) **estructura de “árbol”**. DOM ofrece, a través de sus interfaces, una visión abstracta de esa estructura de árbol. Cada uno de los elementos constructivos del documento XML viene representado por objetos de **tipo “nodo”** basado en los principios del diseño orientado a objetos. Estos nodos serán: elementos, atributos, comentarios, instrucciones de procesamiento, etc.



- **SAX:** El API SAX no especifica como ha de ser la implementación del *parser*, sino como ha de comportarse, es decir, especifica interfaces de programación y no clases. Por lo tanto, SAX no es ningún programa concreto, sino una especificación abstracta que envuelve (mediante la técnica de “layering” habitual en POO) a implementaciones de *parsers* XML distintos.

En su núcleo SAX está compuesto por dos interfaces: el XMLReader que representa al *parser*; y el ContentHandler que recibe datos del *parser*. Estos dos interfaces son suficientes para realizar el 90% de la funcionalidad que podemos necesitar de SAX. En el presente curso vamos a ver las operaciones básicas de XMLReader y con un poco más de detalle el interfaz ContentHandler.

➔ Generación de código para diferentes plataformas.

Resumiendo XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Cuando nos referimos a plataformas incluimos Linux, Android para móviles, Appel en sus diferentes

dispositivos, Microsoft en sus desarrollos Visual Studio o XPS. También se puede usar en bases de datos, editores de texto, hojas de cálculo, dispositivos móviles, portátiles y casi cualquier cosa imaginable.

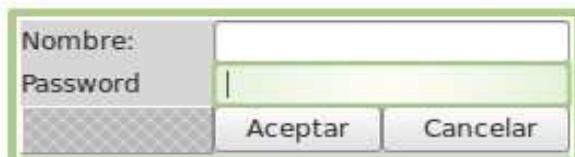
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/principal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <TextView>
        android:id="@+id/etiqueta1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Esto es una etiqueta de texto">
    </TextView>
    <Button>
        android:id="@+id/boton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Esto es un botón">
    </Button>
</LinearLayout>
```

Código XML para Interfaz en Android

En nuestro caso XML es la base para la creación de interfaces gráficas de muchos generadores de código o IDE como Visual Studio, NetBeans y Eclipse para Java y en el caso práctico del proyecto en el desarrollo de interfaces con Glade+Phyton.

ACTIVIDADES

Ejemplo. Dado el siguiente formulario codifica el fichero XML que le corresponde



```
<?xml version="1.0" encoding="UTF-8"?>
<interface>
  <!-- interface-requires gtk+ 3.0 -->
  <object class="GtkWindow" id="form1">
    <property name="can_focus">False</property>
    <property name="title" translatable="yes">Acceso</property>
    <child>
      <object class="GtkGrid" id="grdEtiquetas">
        <property name="visible">True</property>
        <property name="can_focus">False</property>
        <child>
          <object class="GtkLabel" id="lblName">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="xalign">0</property>
            <property name="label" translatable="yes">Nombre:</property>
            <property name="wrap">True</property>
          </object>
          <packing>
            <property name="left_attach">0</property>
            <property name="top_attach">0</property>
            <property name="width">1</property>
            <property name="height">1</property>
          </packing>
        </child>
        <child>
          <object class="GtkEntry" id="entName">
            <property name="visible">True</property>
            <property name="can_focus">True</property>
            <property name="invisible_char">●</property>
          </object>
          <packing>
            <property name="left_attach">1</property>
            <property name="top_attach">0</property>
            <property name="width">1</property>
            <property name="height">1</property>
          </packing>
        </child>
        <child>
          <object class="GtkLabel" id="lblPass">
            <property name="visible">True</property>
            <property name="can_focus">False</property>
            <property name="xalign">0</property>
            <property name="label" translatable="yes">Password:</property>
          </object>
          <packing>
            <property name="left_attach">0</property>
```

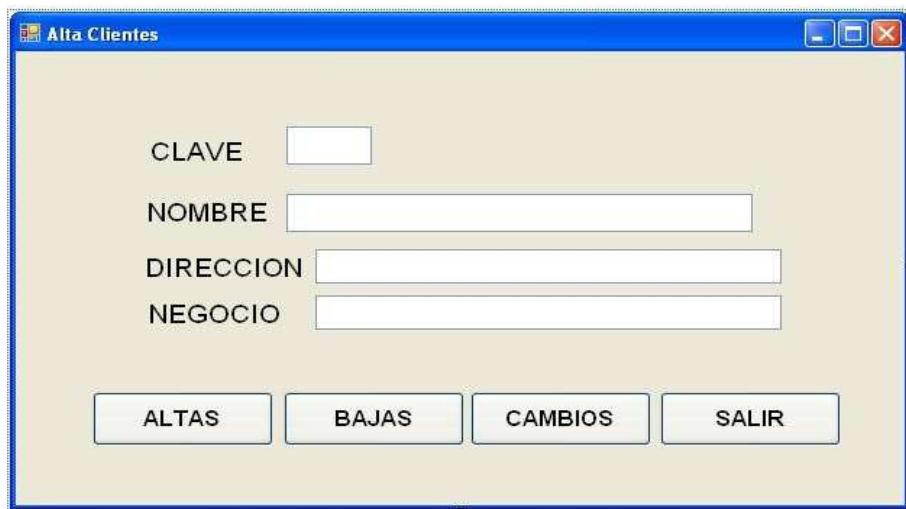
```

<property name="top_attach">1</property>
<property name="width">1</property>
<property name="height">1</property>
</packing>
</child>
<child>
<object class="GtkEntry" id="entPass">
<property name="visible">True</property>
<property name="can_focus">True</property>
<property name="invisible_char">●</property>
</object>
<packing>
<property name="left_attach">1</property>
<property name="top_attach">1</property>
<property name="width">1</property>
<property name="height">1</property>
</packing>
</child>
<child>
<placeholder/>
</child>
<child>
<object class="GtkGrid" id="grdBoton">
<property name="visible">True</property>
<property name="can_focus">False</property>
<property name="orientation">vertical</property>
<child>
<object class="GtkButton" id="btnAceptar">
<property name="label" translatable="yes">Aceptar</property>
<property name="visible">True</property>
<property name="can_focus">True</property>
<property name="receives_default">True</property>
<property name="xalign">0.5899999737739563</property>
</object>
<packing>
<property name="left_attach">0</property>
<property name="top_attach">0</property>
<property name="width">1</property>
<property name="height">1</property>
</packing>
</child>
<child>
<object class="GtkButton" id="btnCancelar">
<property name="label" translatable="yes"> Cancelar</property>
<property name="visible">True</property>
<property name="can_focus">True</property>
<property name="receives_default">True</property>
<property name="xalign">0.56999999284744263</property>
</object>
<packing>
<property name="left_attach">1</property>
<property name="top_attach">0</property>
<property name="width">1</property>
<property name="height">1</property>
</packing>
</child>
</object>
<packing>

```

```
<property name="left_attach">1</property>
<property name="top_attach">2</property>
<property name="width">1</property>
<property name="height">1</property>
</packing>
</child>
</object>
</child>
</object>
</interface>
```

Propuesta 1. Dado el siguiente formulario, escribe el código XML que le corresponde.



Propuesta 2. Descarga xmllint de <https://code.google.com/p/xmllint/downloads/list> y comprueba la salida al ejecutarlo con el fichero del ejercicio anterior.

UD3. Creación de componentes visuales

RA3. Crea componentes visuales, para lo que se valora el uso de herramientas específicas.

- CA3.1. Se identifican las herramientas para el diseño y prueba de componente.
- CA3.2. Se crean componentes visuales.
- CA3.3. Se definieron sus propiedades y se asignaron sus propiedades por defecto.
- CA3.4. Se determinan los eventos a los que puede responder el componente asociándole las acciones correspondientes.
- CA3.5. Se realizaron pruebas unitarias sobre los componentes desarrollados.
- CA3.6. Se documentan los componentes creados.
- CA3.7. Se empaquetaron los componentes.
- CA3.8. Se programaron aplicaciones cuya interfaz gráfica utiliza los componentes creados.

Creación de componentes visuales

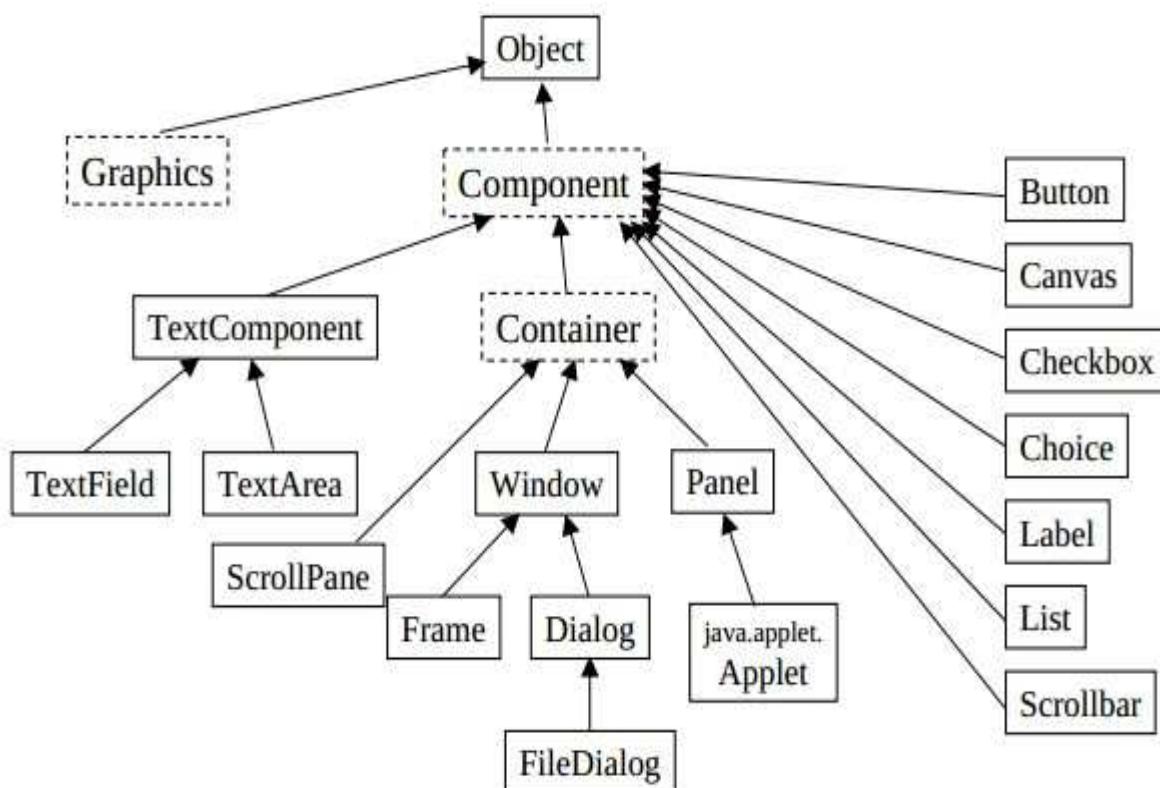
- Concepto y características de los componentes.
- Propiedades y atributos.
- Eventos: asociación de acciones a los eventos.
- Persistencia de un componente.
- Herramientas para desarrollar los componentes visuales.
- Empaquetado de componentes.

→ Componentes visuales de una interfaz gráfica.

Una aplicación gráfica puede tener **componentes visuales** o que vemos (botones, cajas de texto...) y **componentes no-visuales** que no están a la vista del usuario (temporizadores, conexiones a bases de datos...)

Los **componentes GUI (widgets en lo sucesivo)** de una interfaz son los objetos visuales de la misma. De hecho un programa gráfico no es más que un conjunto de **componentes anidados** (ventanas, contenedores, menús, barras, botones, campos de texto, etc...)

Un aspecto importante es la **disposición (layout)** es decir, cómo se colocan los componentes para lograr un GUI cómodo de utilizar. Los **layout managers** gestionan la organización de los componentes gráficos de la interfaz.



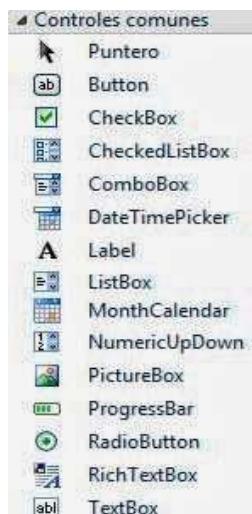
Esquema de componentes gráficos de IU de Java

Los componentes visuales podemos dividirlos a su vez en dos tipos:

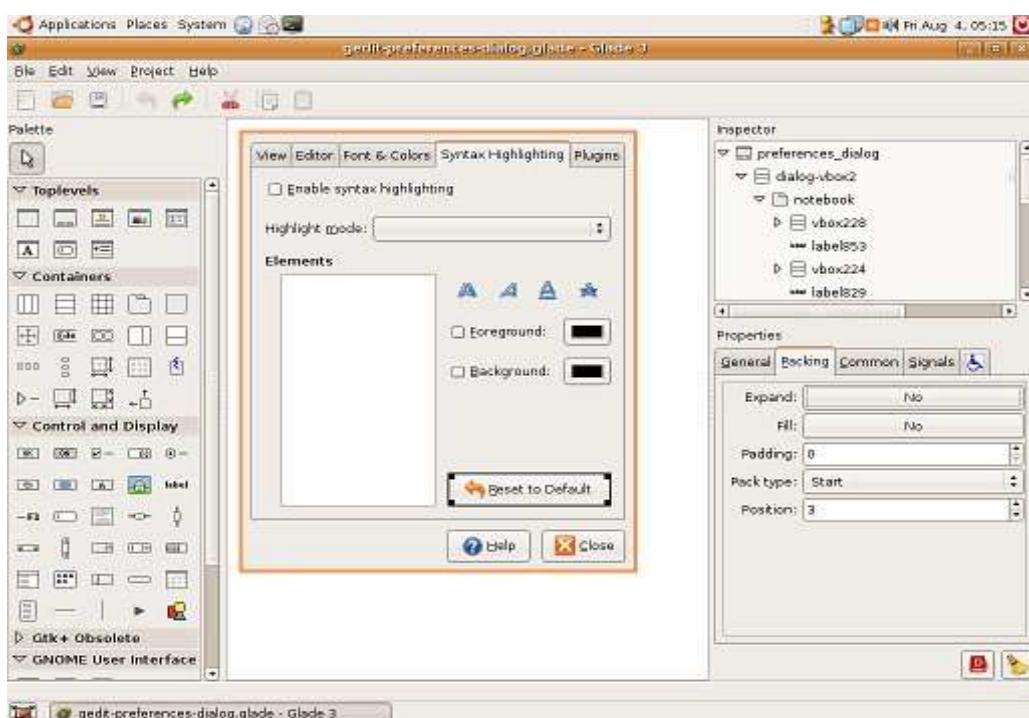
- **Componentes interactivos:** permiten que el usuario final los manipule, ya sea introduciendo datos, seleccionando elementos, etc. De forma que estos componentes pueden recibir el **foco**, es decir, permite enviar el cursor a cualquier zona de la interfaz (con `setFocus`) así como los eventos propios del teclado y del ratón. Normalmente, el **propio sistema operativo es el encargado de dibujar** el aspecto del componente, haciendo el componente las llamadas correspondientes para que este aspecto cambie.

- **Componentes gráficos:** el propio componente es el encargado de dibujar en la pantalla lo que crea oportuno, bien a través de las funciones básicas del API de Windows según el IDE utilizado o bien a través de otras librerías gráficas, como OpenGL, DirectX, etc. Estos componentes, no suelen recibir eventos del usuario final, aunque si eventos del propio programador, ya que su cometido no suele ir más allá de mostrar ciertos gráficos o imágenes en la pantalla.

Habiendo muchos componentes si tuviésemos que elegir los más habituales en una ventana o GUI estos serían quizás los más importantes (hay bastantes más):



Controles Visual Basic



Panel de Trabajo de Glade. A la izquierda los controles.

Como comentamos antes la distribución de los componentes en la ventana es fundamental para la

legibilidad de la misma por el usuario.

Para agrupar/clasificar/distinguir elementos dentro de una ventana, se suele dividir ésta en **áreas**.

Las áreas se pueden formar:

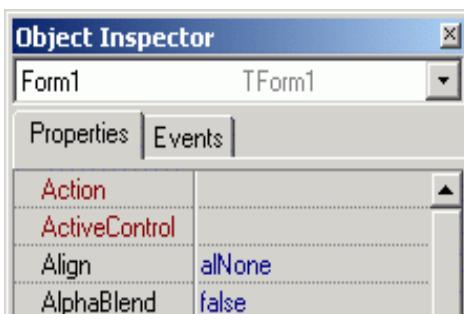
- Utilizando **elementos visuales distintivos**, como líneas de división, tipos de letra, colores, etc.
- Las etiquetas **para titular una sección**, un grupo de elementos o un elemento simple pueden ser independientes (Label) o el dato de título (Caption) de un objeto gráfico.
- Utilizando **objetos contenedores**, que usualmente imponen algún tipo de organización y que suelen ser lo más recomendable porque facilita la manipulación del diseño de la interfaz.

→ Propiedades y atributos de los componentes visuales

Como hemos dicho varias veces a lo largo del curso los **componentes visuales**, a diferencia de los no visuales, son aquellos que, al utilizarlos, muestran algún elemento (o dibujo) en la pantalla y es el usuario de nuestros programas el que interactúa con él. El componente es el principal responsable de dibujar en la pantalla lo que sea oportuno, dependiendo de su estado, del valor de sus atributos, etc.

Todo componente tiene **propiedades, eventos, métodos y atributos**

- Las **propiedades** son datos públicos del componente, muy **parecidas a los atributos de una clase**, aunque se accede a ellas a través de dos métodos: un método para leer su valor, y otro para modificarlo. Existen propiedades de sólo lectura, en las que podemos consultar pero no modificar su valor, y propiedades de sólo escritura. Por ejemplo, las propiedades “Alto” (Width) y “Ancho” (Height) de un botón permiten que un programador pueda cambiar las dimensiones del componente. Cuando el programador cambia alguna de ellas, el componente debe redibujarse en la pantalla, para mostrar los nuevos cambios.



- Los **atributos**. Tienen la misma misión que en programación orientada a objetos, es decir: almacenar datos internos al objeto (o clase). En el mundo de los componentes, los atributos siempre son internos y de uso privado, y debemos utilizar las propiedades para

que un programador pueda leer o establecer un dato. Por ejemplo, propiedad **color** le corresponde **atributos** *rojo, verde....*

No hay que confundir “propiedad” con “atributo” :

1. Ambos describen el estado interno y comportamiento, pero en dos contextos diferentes
2. Normalmente los atributos nunca son públicos. En cambio las propiedades siempre son características públicas de los componentes
3. Los atributos son accesibles **únicamente a nivel de programación**. Las propiedades son accesibles tanto a nivel de programación como interactivamente desde el entorno de desarrollo

Finalmente los **eventos** los que le dan funcionalidad a los **widgets** y de los que hablaremos a continuación. Los eventos vienen regidos por los **métodos o funciones** que son llamados desde el programa por acción del usuario. Cada tipo de objeto o control tiene sus propios métodos.

→ **Eventos: asociación de acciones a eventos.**

Como indicamos en apartados anteriores un **evento** es una **acción realizada por el usuario de una interfaz** y que da lugar a un efecto mediante la ejecución de una función.

El **ratón**, por ejemplo y según el lenguaje de programación utilizado, tiene los siguientes eventos:

- *Mouse Over*. Se produce cuando el puntero del ratón se encuentra por encima de una determinada zona.
- *Mouse Out*. Se produce cuando el puntero del ratón abandona una determinada zona.
- *Mouse Clicked*. Se produce cuando se pulsa un botón del ratón.
- *Mouse Double-Clicked*. Se produce cuando se pulsa dos veces en un intervalo pequeño de tiempo un botón del ratón.

También depende del objeto sobre el que actúa, si es un enlace directo lanza una aplicación si es sobre un CheckBox, activa y desactiva.

Por otro lado están los eventos vinculados al **teclado** y también dependiendo del tipo de lenguaje y objeto sobre el que actúa puede ser:

- *onKeyPress*: evento que se activa cuando un usuario pulsa y deja de pulsar una tecla o también cuando la mantiene pulsada;
- *onKeyDown*: activado cuando se pulsa la tecla;

- *onKeyUp*: activado cuando una tecla, que se había pulsado, deja de pulsarse;

Similar al caso anterior puede haber más eventos pero estos son los más utilizados.

La **programación dirigida por eventos** es un paradigma de la programación el que tanto la estructura como la **ejecución de los programas** van determinados por los **sucesos** que ocurrán en el sistema, definidos por el **usuario o que ellos mismos provoquen**.

Para entender la programación dirigida por eventos, podemos compararla con la programación estructurada. Mientras en la programación secuencial o estructurada es el programador el que define cuál va a ser el flujo del programa, en la programación dirigida por eventos **será el propio usuario** —o lo que sea que esté accionando el programa— el que dirija el flujo del programa. Aunque en la programación secuencial puede haber intervención de un agente externo al programa, estas intervenciones ocurrirán cuando el **programador lo haya determinado**, por ejemplo a la espera de una introducción de datos para seguir la ejecución del programa, y no en cualquier momento como puede ser en el caso de la programación dirigida por eventos.

El creador de un programa dirigido por eventos **debe definir los eventos** que manejarán su programa y las acciones que se realizarán al producirse cada uno de ellos, lo que se conoce como el administrador de eventos. Los eventos soportados estarán determinados por el lenguaje de programación soportado por el SO o los creados por el programador.

En la programación dirigida por eventos, al comenzar la ejecución del programa se llevarán a cabo las inicializaciones y demás código inicial y a continuación el programa quedará **bloqueado hasta que se produzca algún evento**. Cuando alguno de los eventos esperados por el programa tenga lugar, el programa pasará a ejecutar el código del correspondiente al administrador o **manejador del evento**. Por ejemplo, si el evento consiste en que el usuario ha hecho click en el botón de play de un reproductor de películas, se ejecutará el código del manejador que será el que haga que la película se muestre por pantalla.

La programación dirigida por eventos **es la base** de lo que llamamos **interfaz de usuario**, aunque puede emplearse también para desarrollar interfaces entre componentes de Software.

La programación orientada a eventos permite **interactuar con el usuario en cualquier momento de la ejecución**. Esto se consigue debido a que los programas creados bajo esta arquitectura se componen por un bucle exterior permanente encargado de recoger los eventos, y distintos procesos que se encargan de tratarlos. Habitualmente, este bucle externo permanece oculto al programador que simplemente se encarga de tratar los eventos, aunque en algunos lenguajes será necesaria su construcción.

```

while (true){
    Switch (event){
        case mouse_button_down:
        case mouse_click:
        case keypressed:
        case Else:
    }
}

```

La programación orientada a eventos supone una complicación añadida con respecto a otros tipos de programación, debido a que el **flujo de ejecución del software escapa al control del programador**. En cierta manera podríamos decir que en la programación clásica el flujo estaba en poder del programador y era este quién decidía el orden de ejecución de los procesos, mientras que en programación orientada a eventos, es el usuario el que controla el flujo y decide.

Pongamos como ejemplo de la problemática existente:

"un menú con dos botones, *botón 1* y *botón 2*. Cuando el usuario pulsa *botón 1*, el programa se encarga de recoger ciertos parámetros que están almacenados en un fichero y calcular algunas variables. Cuando el usuario pulsa el *botón 2*, se le muestran al usuario por pantalla dichas variables."

Es sencillo darse cuenta de que la naturaleza indeterminada de las acciones del usuario y las características de este paradigma pueden fácilmente desembocar en el error fatal de que se pulse el *botón 2* sin previamente haber sido pulsado el *botón 1*. Aunque esto no pasa si se tienen en cuenta las propiedades de dichos botones, haciendo inaccesible la pulsación sobre el *botón 2* hasta que previamente se haya pulsado el *botón 1*.

Con la evolución de los lenguajes orientados a eventos, la interacción del software con el usuario ha mejorado enormemente permitiendo la aparición de interfaces que, aparte de ser la vía de comunicación del programa con el usuario, son la propia apariencia del mismo. Estas interfaces, o **GUI (Graphical User Interface)**, han sido la herramienta imprescindible para acercar la informática a los usuarios, permitiendo en muchos casos, a principiantes utilizar de manera intuitiva y sin necesidad de grandes conocimientos, el software que ha colaborado a mejorar la productividad en muchas tareas.

Con todo ello observamos que en la **programación orientada a eventos** no existe un único flujo de ejecución y que el funcionamiento es **asíncrono** porque depende de acciones externas al mismo. Aún así los tipos de eventos los podemos dividir en:

- **externos:** producidos por el usuario (teclado o ratón)
- **internos:** producidos por el sistema (temporizadores, transmisiones de datos..)

Uno de los periféricos que ha cobrado mayor importancia tras la aparición de los programas orientados a eventos ha sido el ratón, gracias también en parte a la aparición de los sistemas operativos modernos con sus interfaces gráficas. Sus interfaces graficas suelen dirigir directamente al controlador interior que va entrelazado al algoritmo.

➔ Persistencia del componente

La persistencia permite al programador **almacenar, transferir y recuperar** el estado de los objetos. Para esto existen varias técnicas:

- **serialización** (o *marshalling* en inglés) consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML, entre otros. La serie de bytes o el formato pueden ser usados para crear un nuevo objeto que es idéntico en todo al original, incluido su estado interno (por tanto, el nuevo objeto es un clon del original). La **serialización** es un mecanismo ampliamente usado para transportar objetos a través de una red, para hacer persistente un objeto en un archivo o base de datos, o para distribuir objetos idénticos a varias aplicaciones o localizaciones.

La **serialización** tiene una serie de ventajas:

- Un método de persistencia de objetos que es más conveniente que escribir sus propiedades a un archivo de texto en disco.
- Un método de emisión de llamadas a procedimiento remoto, por ejemplo, como en SOAP.
- Un método para la distribución de objetos, especialmente en los componentes software, tales como COM, CORBA, etc.
- Un método para detectar cambios en variables en el tiempo.
- **base de datos orientada a objetos**, la información se representa mediante objetos como los presentes en la POO. Un **sistema gestor de base de datos orientada a objetos (ODBMS, object database management system)** hace que los objetos de la base de datos aparezcan como objetos de un lenguaje de programación en uno o más lenguajes de programación a los que dé soporte. Un ODBMS extiende los lenguajes con datos persistentes de forma transparente, control de concurrencia, recuperación de datos, consultas asociativas y otras capacidades.

Las bases de datos orientadas a objetos se diseñan para trabajar bien en conjunción con lenguajes de programación orientados a objetos como Java, C#, Visual Basic.NET y C++. Los ODBMS son una buena elección para aquellos sistemas que necesitan un buen rendimiento en la manipulación de tipos de dato complejos. Los ODBMS proporcionan los costes de desarrollo más bajos y el mejor rendimiento cuando se usan objetos gracias a que almacenan objetos en disco y tienen una integración transparente con el programa escrito en un lenguaje de programación orientado a objetos, al almacenar exactamente el modelo de objeto usado a nivel aplicativo, lo que reduce los costes de desarrollo y mantenimiento.

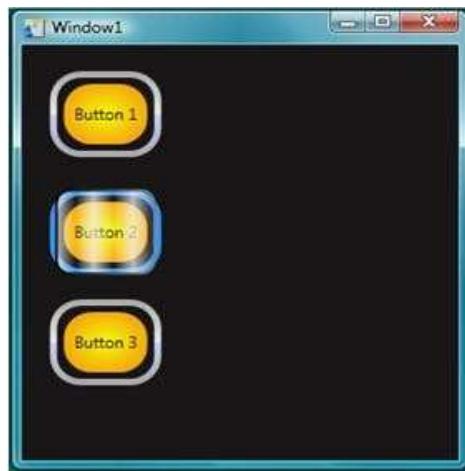
En años recientes, han aparecido muchos prototipos experimentales y sistemas de bases de datos comerciales orientados a objetos. Entre los primeros se encuentran los sistemas **ORION**, **OpenOODB**, **IRIS**, **ODE** y el proyecto **ENCORE/ObServer**. Y entre los sistemas disponibles en el mercado están: **GEMSTONE/OPAL** de ServicLogic, **ONTOS** de Ontologic, **Objectivity** de Objectivity Inc., **Versant** de Versant Technologies, **ObjecStore** de Object Design y **O2** de O2 Technology.

- **motor de persistencia** encargado de traducir entre los dos formatos de datos: **de registros a objetos y de objetos a registros**. Cuando el programa quiere grabar un objeto llama al motor de persistencia, que traduce el objeto a registros y llama a la base de datos para que guarde estos registros. De la misma manera, cuando el programa quiere recuperar un objeto, la base de datos recupera los registros correspondientes, los cuales son traducidos en formato de objeto por el motor de persistencia. Una ventaja del motor de persistencia es que es el mismo para todas las aplicaciones. De esta forma sólo debe programarse una vez y puede usarse para todas las aplicaciones que se desarrollen.

➔ **Herramientas para el desenvolvimiento de componentes visuales.**

Con cualquier herramienta de diseño gráfico podemos diseñar diferentes tipos de componentes visuales, preferentemente botones e iconos ... Una herramienta libre y muy versátil es **Gimp**. A la hora de diseñar un componente visual es importante guardar la una relación adecuada en sus dimensiones.

Otro ejemplo es **Microsoft Expression Blend** es una herramienta profesional desarrollada por Microsoft, de diseño que permite controlar la eficacia del XAML, .NET y Silverlight para proporcionar experiencias de usuario atractivas como la generación de nuevos botones, cajas de texto... en escritorios conectados y Web.



Diseño de Botones con Expression Blend

Una vez diseñados los componentes queda incluirlos en el entorno de desarrollo o en la aplicación. En las prácticas plantearemos la modificación del diseño de algunos componentes.

ACTIVIDADES

Propuesta 1. Diseña un formulario con los siguientes campos:

NIF, Nombre, Apellidos, Calle, Número, Piso, Puerta, CP, Ciudad, Provincia

Con cuatro botones

Grabar, Modificar, Borrar, Salir

Los botones deberás diseñarlos con **Gimp**.

A. Diseñar el GUI con Glade

B. Programas los siguientes eventos:

- Botón *Salir*: el programa se cierra
- Botón *Grabar, Borrar y Modificar*: mostrará una nueva ventana donde nos indicará si estamos seguros de llevar a cabo ese paso. Dicha ventana contiene a su vez dos botones *Aceptar* y *Cancelar*.
- Genera la base de datos con SQLite y que los botones realicen las operaciones sobre la misma que se indican en el nombre de los mismos.
- Diseña un botón que indique Mascotas que lance la ventana con los siguientes datos *Mascota, Genero, Especie, Historial Sanitario*.
- Dicha ventana tendrá una lista desplegable que muestre las diferentes mascotas del dueño y que nos permitirá seleccionar una existente o bien un botón que nos permita dar una de alta.
- Dicha ventana tendrá los mismos botones con las mismas funciones que la UI de Usuario sobre la tabla correspondiente de mascotas.

UD4. Usabilidad

RA4. Diseñar interfaces gráficas, para identificar y aplicar de criterios de usabilidad.

- CA4.1. Se crearon menus ajustados a los estándares.
- CA4.2. Se crearon menus contextuales cuya estructura y contenido sigan los estándares.
- CA4.3. Se distribuyen las acciones en menús, barras de herramientas, botones de comando... según un criterio coherente.
- CA4.4. Se distribuyeron adecuadamente los controles en la interfaz de usuario.
- CA4.5. Se utilizó el tipo de control más adecuado al caso.
- CA4.6. Se diseño el aspecto de la interfaz (colores, fuentes...) atendiendo a su legibilidad.
- CA4.7. Se verificó que los mensajes generados por la aplicación sean adecuados y legibles en extensión y claridad.
- CA4.8. Se realizaron pruebas para evaluar la usabilidad de la aplicación.

BC4. Usabilidad³

- Usabilidad: características y atributos. Normas relacionadas con la usabilidad.
- Medida de la usabilidad de las aplicaciones: tipos de métricas.
- Pautas de diseño da estructura de la interfaz de usuario: menús, ventanas, cuadros de diálogo, atajos de teclado, etc.
- Pautas de diseño del aspecto de la interfaz de usuario: colores, fuentes, iconos, distribución de los elementos, etc.
- Pautas de diseño de los elementos interactivos de la interfaz de usuario: botones de comando, listas desplegables, etc.
- Pautas de diseño de la secuencia de control de la aplicación.
- Pruebas de usabilidad.

³ Las pautas presentadas en este tema más allá de pretender regir el diseño de la interfaz de las aplicaciones son un compendio de recomendaciones planteadas por:

- ✓ El equipo de Diseño de Interfaz de Usuario de Microsoft®.
- ✓ El proyecto de Usabilidad del Grupo GNOME, Desarrolladores de Software de distribución libre para plataformas Linux.

→ Usabilidad: características y atributos. Normas relacionadas con la usabilidad.

Usabilidad se refiere a la **facilidad** con que las personas pueden utilizar una herramienta particular o cualquier otro objeto fabricado por humanos con el fin de alcanzar un objetivo concreto. La usabilidad también puede referirse al estudio de los principios que hay tras la eficacia percibida de un objeto.

En la **interacción persona-ordenador**, la usabilidad se refiere a la claridad y la **elegancia** con que se diseña la interacción con un programa de ordenador o un sitio web. El modelo conceptual de la usabilidad, **proveniente del diseño centrado en el usuario**, no está completo sin la idea utilidad. En inglés, **utilidad + usabilidad** es lo que se conoce como *usefulness*. El uso, funcionalidad o utilidad de un diseño, programa u objeto no está sólo en la facilidad de manejo de la aplicación inmediata que hacemos de ellos, sino también y sobre todo en el beneficio o solución que obtenemos

Algunas conclusiones y casos recogidos en estudios e investigaciones realizados por Sun Microsystems (hoy de Oracle) demostraron que:

- Una buena usabilidad demuestra reducciones del ciclo de desarrollo de los productos de 33-50%
- 63% de todos los proyectos de desarrollo de software sobrepasan su presupuesto, siendo las cuatro causas más importantes las relacionadas con usabilidad.
- El porcentaje de código que se dedica al desarrollo de la interfaz con los usuarios ha ido aumentando a lo largo de los años hasta un promedio 47-60% del conjunto de la aplicación como vimos en unidades anteriores.
- 80% de las tareas de mantenimiento se deben a requerimientos de usuarios no previstos, quedando el resto debido a fallos y errores.

La Organización Internacional para la Estandarización (ISO) ofrece dos definiciones de usabilidad relacionadas con el software:

- **ISO/IEC 9126:**

"La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso"

Esta definición hace énfasis en los atributos internos y externos del producto, los cuales contribuyen a su funcionalidad y eficiencia. La usabilidad depende no sólo del producto si no que también hace incidencia en el **factor usuario**.

- **ISO/IEC 9241:**

"Usabilidad es la eficacia, eficiencia y satisfacción con la que un producto permite alcanzar objetivos específicos a usuarios específicos en un contexto de uso específico"

Es una definición centrada en el concepto de calidad en el uso, es decir, se refiere a cómo el usuario realiza tareas específicas en escenarios específicos con efectividad.

A partir de la conceptualización llevada a cabo por la ISO, se infieren los principios básicos en los que se basa la usabilidad:

- **Facilidad de Aprendizaje:** facilidad con la que nuevos usuarios desarrollan una interacción efectiva con el sistema o producto. Está relacionada con la **predicibilidad**, sintetización, familiaridad, la generalización de los conocimientos previos y la consistencia.
- **Facilidad de Uso:** facilidad con la que el usuario hace uso de la herramienta, con **menos pasos o más naturales** a su formación específica. Tiene que ver con la eficacia y eficiencia de la herramienta.
- **Flexibilidad:** relativa a la variedad de posibilidades con las que el usuario y el sistema pueden intercambiar información. También abarca la posibilidad de diálogo, la multiplicidad de vías para realizar la tarea, similitud con tareas anteriores y la optimización entre el usuario y el sistema.
- **Robustez:** es el nivel de apoyo al usuario que facilita el cumplimiento de sus objetivos. Está relacionada con la capacidad de observación del usuario, de recuperación de información y de ajuste de la tarea al usuario.



Uno de estos expertos y gurú de la usabilidad es Jakob Nielsen, quien definió la usabilidad en el año 2003 como "un atributo de calidad que **mide lo fáciles de usar** que son las interfaces".

Para acabar esta introducción mencionar las normas o estándares relacionados con la **usabilidad**:

- **ISO 13407:** Señala los principios a llevar a cabo en el proceso de diseño centrado en el usuario para sistemas interactivos
- **ISO/TR 16982:** Métodos de usabilidad que soportan diseño centrado en el usuario. Presenta una lista de métodos ergonómicos que pueden ser aplicados a las diferentes etapas del ciclo de diseño, precisando sus ventajas y desventajas
- **ISO 9241-10:** Principios para diálogos y diseño y evaluación de diálogos entre el usuario y los sistemas de información (adaptación a la tarea, carácter auto descriptivo, control por parte del usuario,
- **ISO 9241-11:** Guía de especificaciones y medidas de usabilidad
- **ISO 9241-12:** Presentación de la información. Se aborda por lo tanto la organización de la información (ubicación de la información, adecuación de las ventanas, zonas de información, zonas de entrada/salida, grupos de información, listas, tablas, etiquetas, campos, etc.), los objetos gráficos (cursors y punteros, etc.), y las técnicas de codificación de la información (codificación alfanumérica, abreviación de códigos alfanuméricos, codificación gráfica, codificación por colores, marcadores, etc.).
- **ISO 9241-13:** Guía del usuario. Relativas a las ayudas del usuario.
- **ISO 9241-14:** Diálogos de menús. recomendaciones para el diseño ergonómico de los menús, es decir tipos de interacción en el que se presentan opciones a los usuarios bajo diferentes formas (ventanas de dialogo con casillas a marcar, botones, campos, etc.).. y según las características del usuario.
- **ISO 9241-15:** Diálogos de tipo lenguaje de órdenes
- **ISO 9241-16:** Diálogos de manipulación directa. Esta parte aborda las metáforas gráficas, la apariencia de los objetos utilizados en la manipulación directa, el feedback, los dispositivos de entrada de datos, la manipulación de objetos, el punteo y la selección, el dimensionamiento, la manipulación directa de las ventanas y los iconos, etc.
- **ISO 9241-17:** Diálogos por cumplimentación de formularios. recomendaciones dadas en esta parte tienen que ver con la estructura de los formularios, los campos y etiquetas, las entradas (textuales alfanuméricas, de opción, los controles, las validaciones, etc.), el feedback y la navegación en el formulario.
- **ISO 14915:** Ergonomía del software para interfaces de usuario multimedia

→ Medida de usabilidad de las aplicaciones: tipos de métricas.

En la literatura sobre el tema se pueden encontrar multitud de métodos para evaluar la usabilidad ya que este concepto es crucial para asegurar un bien diseño de la interfaz de usuario y de la interacción entre el sistema y el usuario final. Solo haremos breves referencias a los mismos y concretamente a dos de ellos:

- **User testing:** en este método **el usuario tiene un gran influencia en el desarrollo** del prototipo de la aplicación. Por ello requiere una gran planificación y supone un gran coste.
- **Usability inspection:** en este caso la **participación del usuario es testimonial** y se echa mano de expertos en desarrollo y usabilidad además de consulta de guías de estilo y heurísticas ya establecidas.

En la siguiente tabla que muestra una serie de métodos aplicables en las diferentes fases del ciclo de desarrollo del sistema y que permiten alcanzar mayores cotas de usabilidad y las fases del ciclo en que pueden utilizarse:

| <i>Fase del ciclo de desarrollo</i> | <i>Método de mejora</i> |
|---------------------------------------|--|
| Recopilación de requisitos y análisis | <ul style="list-style-type: none"> - Indagación - Observación (estudio etnográfico) - Tormenta de ideas - Encuestas, cuestionarios, entrevistas - Categorización (por tarjetas) - Prototipado |
| Diseño e implementación | <ul style="list-style-type: none"> - Técnicas de composición de interfaz - Diseño paralelo - Secuencias de escenarios - Construcción de escenarios - Cuadros de asignación de tareas - Análisis de tareas - Matriz de funcionalidad - Guías de estilo, heurísticas, patrones - Inspecciones formales de usabilidad - Evaluación heurística - Revisiones cognitivas - Guías de comprobación - Cuestionarios, entrevistas - Pensando en alto |
| Evaluación de la interfaz | |

Las **métricas** que se presentan a continuación y propuestas por Constantine & Lockwood se utilizan para medir la **complejidad y la adecuación del diseño de la interfaz de usuario** así como la cohesión de los datos. La información dada por estas métricas es útil para hacer una estimación de la facilidad de aprendizaje o de memorización de la interfaz, con la consiguiente disminución de errores en la utilización del interfaz por parte del usuario.

- **Essential Efficiency. (Eficiencia esencial).** Esta métrica pretende medir la **simplicidad** de la interfaz. Para ello tiene en cuenta el **número de pasos que realiza un usuario** para una tarea (por ejemplo, nº de clicks para formatear un documento)
- **Task Concordance. (Concordancia de tareas).** Pretende medir la eficiencia y la simplicidad para ello valora que las **tareas más frecuentes sean las más sencillas**.
- **Task Visibility. (Visibilidad de la tarea).** La visibilidad de la tarea a realizar. Mide la proporción de objetos del interfaz o elementos necesarios para completar una tarea
- **Layout Uniformity. (Uniformidad del esquema).** Mide la **uniformidad y regularidad del esquema de diseño** de la interfaz. Cómo de similares son los elementos de la interfaz.
- **Visual Concordance. (Coherencia visual).** Trata de **medir la comprensibilidad** de la interfaz.

→ **Pautas de diseño da estructura da interfaz de usuario: menús, ventanas, cuadros de diálogo, atajos de teclado, etc.**

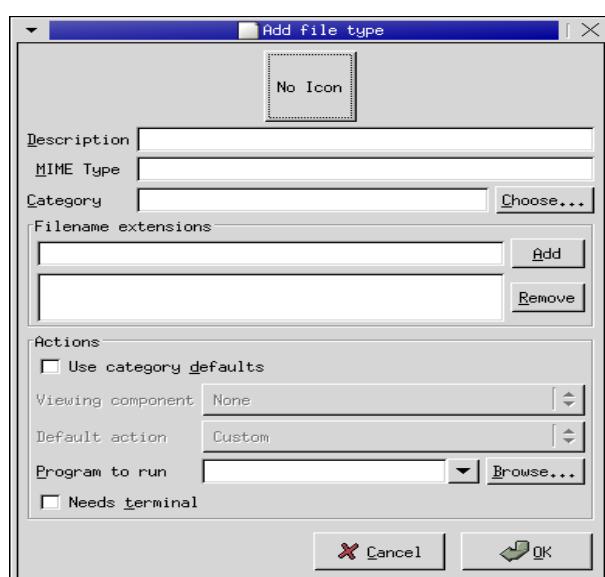
Las pautas de diseño se basan en los siguientes parámetros:

- **Estructura de la información y las tareas del usuario en la aplicación:** Se distingue aquí la **posición y jerarquía** de los elementos visuales con respecto a los otros elementos que componen la ventana. Influye además, el **orden de ejecución de las tareas** del usuario, debe facilitarse la comprensión de este orden.
- **Punto Focal en la ventana:** Se determina la **ubicación de los elementos prioritarios**. Determinada la idea central, surge el punto focal para la actividad. Este punto focal, debe **destacarse sobre los demás elementos o controles de la interfaz**, con técnicas que estimulen el proceso cognitivo de la Selección (de la información pertinente): uso del espaciado, aislamiento u otros métodos. La ubicación de los elementos en la interfaz gráfica puede estar afectada por la cultura, las pautas de diseño y en ocasiones, por las técnicas de diseño instruccional que se apliquen. En general, en la **cultura occidental**, donde se lee de izquierda a derecha y de arriba hacia abajo, las personas buscan la información importante en la parte superior izquierda de la pantalla, luego, en este orden, se tienden a ubicar los elementos en la interfaz, de acuerdo a su importancia y relación.
- **Estructura y Consistencia entre ventanas:** La estructura de la organización de los elementos en todas las ventanas de una aplicación debe ser constante, estandarizándose elementos como: presentación de menús, botones de comandos, etiquetas, etc.

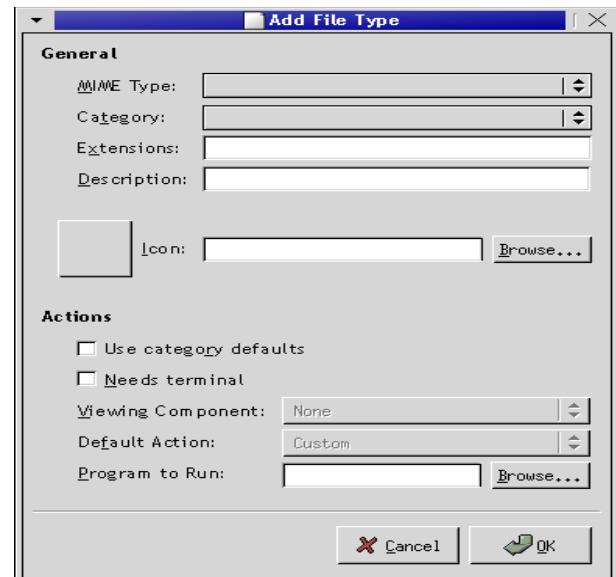
- **Relación entre elementos:** Trata de la proximidad espacial que debe existir entre elementos de la interfaz que presenten nexo informativo-comunicativo, por ejemplo: una lista que permita seleccionar valores que son cargables a un cuadro de texto, en este caso, ambos controles deben estar espacialmente cercanos
- **Legibilidad y Flujo entre los elementos:** Facilidad de lectura y comprensión de la comunicación de las ventanas, dada en función de espaciado y alineación de los elementos de la interfaz.
- **Integración:** Se mide la relación entre el diseño visual de la aplicación y las aplicaciones del sistema u otras aplicaciones del entorno gráfico con las que se utiliza.

La localización visual de los componentes es importante porque la relación entre los componentes es indicada por su posición. Esto se llama “**LAYOUT**” en diseño de interfaces. Un **layout limpio es crucial para crear un flujo visual de información** sin problemas para el usuario.

- **Cuadros de diálogo:** cuando un usuario revisa un cuadro de dialogo complejo (que contiene muchas etiquetas, cuadro de texto, botones varios, etc.), es fácil ver, como esta interfaz puede convertirse en un estorbo cuando el layout es deficiente y en consecuencia, el diseño visual, es pobre también.



Pésima distribución



Óptima distribución

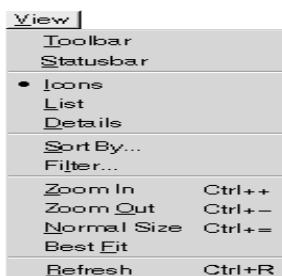
El cuadro de Dialogo de la izquierda, presenta las etiquetas sin alinear. Si miramos fijamente se notará la dificultad para hacer un “scaneo” o revisión rápida de la pantalla. Algunos consejos importantes son:

- Cuando los controles (cuadro de texto) tengan la **misma longitud**, se recomienda **alineación izquierda**.
- Si la mayoría de un grupo de etiquetas **difieren en longitud**, **se recomienda alineación derecha**.
- **No abusar de los bordes.** consistencia de los componentes de la ventana en términos de alineación y tamaño. En resumen evitar que los ojos del usuario estén dando saltos de un sitio a otro.
- **Menús:** componen el rango de comando u **operaciones ejecutables** de una aplicación. Cuando se diseña una nueva aplicación, es recomendable colocar los **ítems del menú, en el mismo lugar que éstos aparecen en otras aplicaciones**, esto contribuye, dentro de los Criterios de Usabilidad, a que la nueva aplicación **sea más fácil de aprender** por los usuarios.
- **Barra de menús:** Proporciona acceso a un subconjunto de menús desplegables (**drop-down o pull-down menu**). Solo se muestra el título del menú hasta que el usuario hace clic sobre el ítem. Esta barra está siempre visible y accesible desde el teclado y con el mouse. En el diseño de aplicaciones, debe proporcionarse acceso a todas las tareas desde la barra de menú.



Una barra de menú **como mínimo** debe contener el ítem **Ayuda**. Se debe seguir la disposición estándar del SO sobre el que correrá la aplicación. Nunca tener ítems inactivos en el menú y no usar palabras compuestas. Finalmente evitar menús que se oculten.

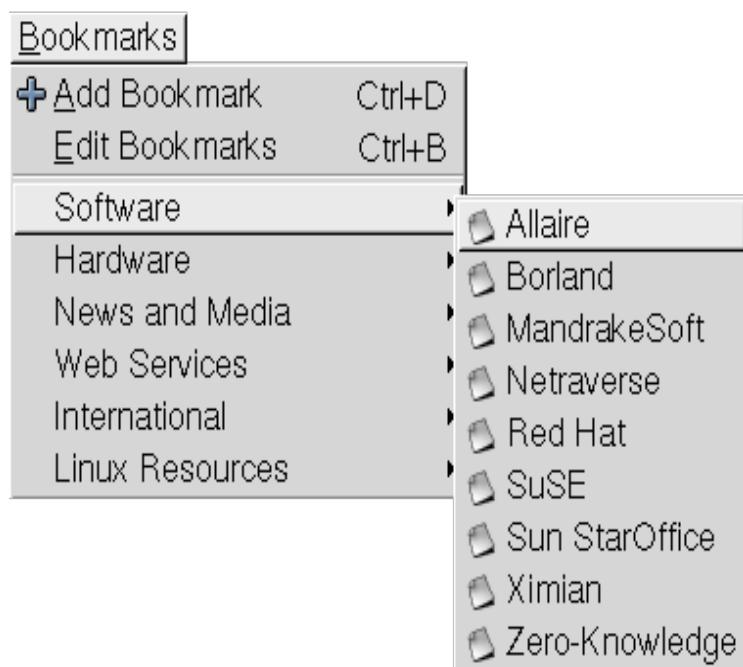
Dentro del menús están los **submenús desplegables**.



Cuando se selecciona un ítem de la barra de menú (presionando clic con el mouse o con el foco en título, clickeando Enter) aparecen los mencionados submenús desplegables. Son llamados también como Drop-Down o Pull-Down menú.

- Se deben organizar los items en grupos relacionados **sobre la función que realizan**. Por ejemplo, no poner el desplegable *guardar fichero* en el item principal *formato*.
- **No superar más allá de 15 sub-items** en un mismo item. En caso necesario reorganizar los items de grado superior.
- **Evitar** la creación de nuevos items en tiempo de ejecución.
- **Usar** combinación de teclas en los más frecuentes.

Finalmente tenemos los **menús en cascada**.



Se debe aplicar su uso solo en **casos necesarios**. Su utilización hace **difícil la navegación** entorpece la búsqueda y revisión de los items contenidos en él. **No diseñar submenú con menos de tres opciones**, a menos que sus items sean agregados dinámicamente (tipo Archivos Recientes Usados de algunas herramientas de productividad). **Evitar más de dos niveles de jerarquía** ya que son difíciles de memorizar y navegar.

- **Menú Contextuales o Emergentes (PopUp Menú)**

Es un tipo de menú desplegable que se muestra bajo determinadas situaciones cuando estan enfocado un objeto y presionarse el **botón derecho del mouse**. Se emplean como su nombre lo indica, para proporcionar la ejecución “Contextual” de una **serie de comandos asociados al**

objeto que tiene el foco al momento de ser invocado (con botón secundario del mouse) el menú. Se debe ser cuidadoso con su aplicación, ya que este tipo de menú es usado principalmente por **usuarios intermedios y avanzados**.



Dado que **muchos usuarios podrían no estar conscientes** de su existencia se deben seguir los siguientes consejos:

- **proporcionar acceso alternativo** para cada una de las funciones o tareas que configure en un menú contextual.
- Deben ser lo más simples posibles para maximizar su eficiencia: **Colocar un máximo de 10 items**.
- **Evitar el uso de menú de cascada** dentro de los menú contextuales.
- Ordenar los items de acuerdo al criterio lógico, operativo o funcional según convenga.
- Usar la línea como separador gráfico para denotar agrupación de opciones relacionadas.

➔ Pautas de diseño del aspecto da interfaz de usuario: colores, fuentes, iconos...

El **color** debe ser considerado como una **herramienta adicional en el diseño**, no una necesidad básica. No se debe depender de colores para mostrar información importante, ya que si los colores no son correctamente percibidos (en casos de que el usuario tenga sistemas de poca

resolución o posea algún impedimento visual leve), no tienen la utilidad que se busca. En el uso del color aplique las siguientes referencias:

- **El color es una forma de información secundaria:** Evitar confiar en el color como único medio de informar una condición o valor.
- **Evitar un número excesivo y colores llamativos**
- **Aplicar un conjunto limitado de colores.** Los colores apagados, sutiles y complementarios suelen ser los más apropiados en el diseño de interfaces en aplicaciones de corte **empresarial y académicas**. En el caso de que la audiencia de los componentes instrucionales sean niños, debe diseñarse la interfaz, como corresponde siempre, en función de los intereses de la audiencia: edad, cultura, conocimientos y conductas previas, etc., en este caso particular, se recomiendan los **colores primarios**, cálidos sin tender a “carnavalizar” la interfaz, a menos que así lo requiera la *intencionalidad* del diseño instruccional del componente de software.
- **Uso de Paletas.** El uso de paletas de combinación de colores a aplicar en los formularios brinda una apariencia de unificación, consistencia y formalidad, minimizando la posibilidad de distractores visuales.

Los usuarios con desordenes visuales (ceguera nocturna, o baja vision nocturna, daltonismo) requieren alternativas para la asignación de colores por defecto de una aplicación. Una buena interfaz de usuario se anticipa a estas necesidades, proporcionando una opción para la **personalización de las preferencias del color**. Aun mejor, si la aplicación esta ya configurada con una cuidadosa selección de color y contraste por defecto. Se estima que un 11% de la población mundial tiene algún desorden de ceguera nocturna. Esta afección se manifiesta en la incapacidad de distinguir ciertos matices tanto del color rojo como del verde (deuteranopia o protanopia) o azul y amarillo (tritanopia).

De cualquier manera, es necesario permitir al usuario que personalice los colores en cualquier parte de la aplicación que presente información importante. *Esto significa que la aplicación debe comunicar la información efectivamente en cualquier configuración o personalización de color que el usuario seleccione.*

Vischeck

Existe una herramienta muy útil en la web  (www.vischeck.com), que permite evaluar la percepción de las imágenes que se usen en la aplicación, presentando como feedback, la imagen gráfica tal y como será percibida por los usuarios con problemas en la visión (en particular los afectados de deuteranopia y tritanopia).

A parte de la diferenciación de matices, existen otros problemas con los **niveles de contraste**, por ejemplo, algunos usuarios requieren altos niveles de contraste entre los colores del fondo y el primer plano (tal como negro sobre blanco o blanco sobre negro u otras combinaciones de alto contraste). Otros pueden experimentar incomodidad si se aplica una asignación previa de bajo contraste como texto gris sobre fondo gris mas brillante.

En cuanto a las **fuentes no se deben emplear más de tres fuentes y tamaños de letras** en la aplicación. Demasiadas fuentes y tamaños de letras harán que la interfaz luzca no profesional y recargada, además de dificultar su lectura. Las fuentes también se utilizan para organizar la información y hasta para transmitir un determinado énfasis a las expresiones (por ejemplo, la mayúscula sostenida en las pautas de comunicación de correo electrónico refieren GRITOS! a nuestro receptor) por lo que se deben evitar siempre.

- **Estilo Mayúsculas de Encabezado:** Iniciar en mayúsculas todas las palabras de los elementos, con las **excepciones**:
 - Artículos: un, una, el, la, los, las
 - Conjunciones: y, pero, mas, para, todavía..
- **Estilo Mayúsculas de Oración:** Colocar en **mayúscula la primera letra de la palabra** inicial y cualquier otra palabra, normalmente iniciada en mayúscula en oraciones, tales como nombres (*fíjate en el nombre de la aplicación con el que ves este pdf*)
- **Evitar las fuentes en cursiva y Serif**, suelen ser más difíciles de leer, especialmente en bajas resoluciones.
- **Limitar el número de fuentes y estilos** usadas en las interfaces de sus aplicaciones, un uso excesivo de fuentes diferentes tenderá al desorden visual de las ventanas.
- **Usar adecuadamente las negritas:** aplicarlas para estimular los procesos cognitivos de Selección y Organización convenientemente, su **aplicación excesiva reduce el énfasis** en la información y dificulta la lectura.
- Siempre que sea posible, **usar la fuente estándar del sistema** para los elementos comunes de la interfaz para estandarizar e integrar su aplicación con las ventanas de las demás herramientas del sistema.

- Las frases deben ser **breves y concisas**, con un lenguaje claro y evidentemente sin errores gramáticos ni ortográficos. Los mensajes de avisos deben ser positivos y en la medida de lo posible ayudar al usuario en su tarea con una breve explicación.

Finalmente los **iconos** no deben ser excesivamente llamativos y acompañados con una palabra inferior que indique su función (*abrir*, *salir*, *aceptar*...). Hay muchos tipos de **iconos con imagen que confunden a los usuarios** sobre su verdadera funcionalidad.

→ **Pautas de diseño dos elementos interactivos da interfaz de usuario: botones de comando, listas desplegables, etc.**

- **Cuadros de Texto (TextBox o Enty)**

Son usados para ingresar una o más líneas de texto plano.

| | |
|-------------|--|
| First name: | Bob |
| Surname: | Helpma |
| Address: | 10 Glebe Street Broonsville Dundee |

- **Rotular los TextBox** con etiquetas textuales colocadas delante del control textbox, de acuerdo al uso de mayúsculas Oración.
- **Justificar a la derecha los cuadros de texto cuyo contenido sea exclusivamente numérico.** Esto es especialmente útil cuando en una ventana el usuario quiera comparar dos valores numéricos en la misma columna de controles, en este caso asegúrese de que el margen derecho del control relevante también esté alineado.
- **Ajustar el tamaño del cuadro de texto de acuerdo al probable tamaño de los datos de entrada.** Esto da una información visual útil acerca del tamaño de la entrada que se espera. No colocar del mismo ancho todos los campos de entrada.
- **Proporcionar un texto estático explicativo** para aquellos textbox que requieran una entrada en un formato particular o en una unidad de medida particular:

| | |
|-------|-------------|
| Time: | 18:30 hh:mm |
|-------|-------------|

- Cuando sea posible, proporcionar un control adicional o alternativo que **limite la entrada de datos requeridas a un rango valido**. Por ejemplo, usar un objeto

ScrollBar o slider si la entrada valida esta entre un rango particular de enteros; también puede usar un objeto Calendario en caso de tratarse del ingreso de una fecha valida:



- **Teclas ENTER (RETURN) y TAB:** Para proporcionar el cambio de foco entre controles de cuadros de texto de su interfaz a través de la presión de estas teclas.

- **Objeto Botones de Comando**

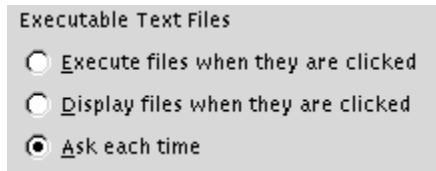
Un botón de comando o Command button, inicia una acción determinada cuando el usuario hace clic sobre él.



- Rotular todos los botones con **verbos en infinitivo**, en combinación con un adjetivo, si se requiere, aplicando el uso de mayúsculas de encabezado, por ejemplo: *Guardar, Ordenar, Actualizar Ahora*.
- Proporcionar una **tecla de acceso** en la etiqueta del botón (letra subrayada) que le permita al usuario activar directamente el botón desde el teclado.
- Usar puntos **suspensivos al final del rótulo del botón para indicar que la acción requiere valores adicionales** antes de ejecutarse la acción.
- No aplicar más de **una o dos anchuras diferentes** para botones en una misma ventana y todos los botones deben tener la misma altura. Esto dará una apariencia visual uniforme a la ventana que la hará más fácil de usar.
- No asociar acciones a los eventos Doble-Clic ni Clic-Derecho de un botón de comando.
- En las ventanas de diálogo, trate de no asignar botones por defecto para las respuestas.

- **Objeto Botones de Opción o Radio Botones**

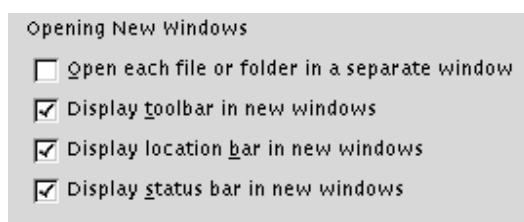
Los botones de opción proporcionan al usuario un **conjunto de valores para la selección de un único valor**. Estos valores son cada uno mutuamente excluyentes.



- Aplicar botones de opción para implementar la selección entre un conjunto de valores de, **mínimo dos elementos**.
- **No deben iniciar una acción** cuando el usuario haga clic sobre un botón de opción.
- La selección de un botón de opción **no debería afectar el valor de ningún otro control**. Sin embargo esta acción si pudiese habilitar o inhabilitar, ocultar o mostrar otros controles de la interfaz.
- Para rotular el grupo de botones de opción, use **combinación de mayúsculas** de encabezado, por ejemplo: *Estilo del Borde*. Ubique esta etiqueta del grupo arriba de los botones o al lado izquierdo de los mismos.
- Rotular cada botón en particular en combinación de **mayúsculas de oración**.
- El número de elementos para botones de opción **no debe exceder a ocho**, si se sobrepasa este valor evaluar la posibilidad de aplicar listas desplegables simples.
- Trate de alinear los botones de opción verticalmente, esto contribuye a hacer más fácil la revisión visual de la ventana.

- **Objeto Botones de Chequeo (CheckBox)**

Son usados para denotar la posibilidad de **selección de múltiples opciones** o valores dentro de un conjunto, estos valores no son mutuamente incluyentes.



- **No iniciar una acción** cuando el usuario haga clic sobre un checkbox.
- La selección de un botón de chequeo no debería afectar el valor de ningún otro control.

- Si la selección de un botón de chequeo afecta a otro control, ubicarlo inmediatamente encima o al lado izquierdo del control que es afectado.
- **Usar combinación de mayúsculas de encabezado** para rotular los el grupo botones de chequeo, ubique la etiqueta arriba o a la izquierda.
- Aplicar a los botones **combinación de mayúsculas** de oración.
- El número de elementos para botones de chequeo **no debe exceder a ocho**, si se sobrepasa este valor evaluar la posibilidad de aplicar listas desplegables con opción de marcas de chequeo .
- Tratar de alinear los botones de chequeo verticalmente ya que facilita la visibilidad.

- **Objeto Cuadro Combinado (Combo Box)**

Son **listas desplegables** usadas para brindar al usuario la capacidad de **selección dentro de un conjunto de valores dados** en la interfaz a través de una lista. En términos de objeto, este control combina la capacidad de un cuadro de texto y una lista.

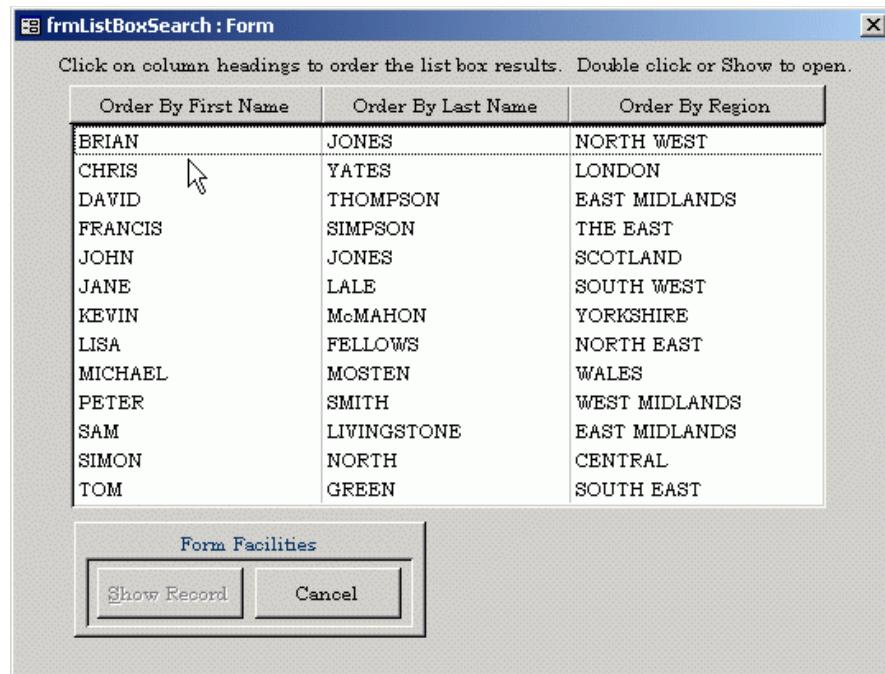


- Se recomienda su uso para gestionar la selección de un único valor entre un conjunto de valores **de más de ocho elementos**.
- La selección de un item **no debería iniciar** ninguna acción en la aplicación
- Rotular este objeto con una etiqueta colocada arriba o a la izquierda del control, **con mayúsculas**. Aplicar a los elementos combinación de mayúsculas de oración
- Tratar de **alinear los botones de opción verticalmente**, esto contribuye a hacer más fácil la revisión visual de la ventana.

- **Objeto Lista (List Box)**

Son usados para brindar al usuario la capacidad de selección de **uno o varios valores dentro de**

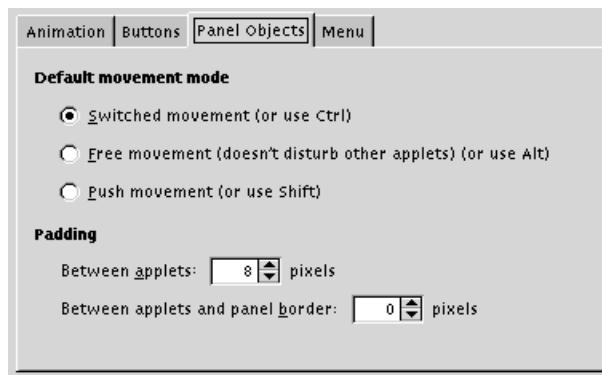
un conjunto de valores dados en la interfaz a través de una lista.



- Se debe rotular con una etiqueta colocada arriba o a la izquierda del control, usando combinación de **mayúscula de encabezado**.
- Aplicar a los elementos de la lista, combinación de mayúsculas de oración
- En la interfaz, diseñar la lista para que muestre **al menos cuatro items a la vez sin realizar scrolling**. Para listas de 10 o más items se debe adecuar al tamaño de manera apropiada.
- No diseñar listas con menos de **cinco items**.
- Sólo usar encabezado de columnas cuando:
 - la lista tenga **más de una columna**
 - la lista tenga solamente una columna pero exista **la posibilidad de reordenamiento en tiempo de ejecución**.
- Para listas de selección múltiple se debe mostrar el número de items actualmente seleccionados en un texto estático debajo de la lista. Por ejemplo: “*Número de Item Seleccionados: 5*”. Esto hace más evidente que la selección múltiple es posible.
- Considerar la **posibilidad de proporcionar botones** “Seleccionar Todo” “Deseleccionar Todo” al lado de la lista de selección múltiple si es apropiado.

- La selección de un ítem de la lista **no debería iniciar** ninguna acción en la aplicación.
- Trate de alinear los botones de opción verticalmente, esto contribuye a hacer más fácil la revisión visual de la ventana.
- **Objeto Tabbed (Control Tabbed o Tabbed NoteBooks)**

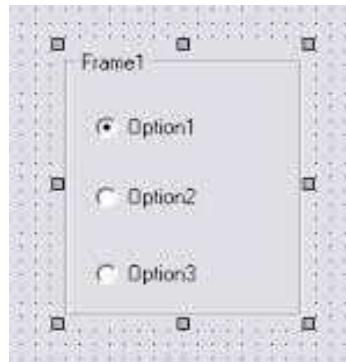
Es un objeto adecuado para presentar **información relacionada en la misma ventana**, sin tener que desplegar toda al mismo tiempo.



- No colocar demasiadas páginas en el mismo cuaderno (**se recomiendan 4 Tabs**).
- Rotular las páginas o pestañas con combinación de **mayúscula de encabezado**.
- Si un control afecta el contenido de una sola página colocarlo dentro de esta página, si afecta a todas las páginas ubicarlo fuera del Control.
- **Objeto Marcos y Separadores (Control Frame)**

Un **marco** es un cuadro con título que puede dibujarse alrededor de un grupo de objetos o controles de la interfaz para organizarlos como grupos funcionales.

Un **separador** es una línea simple, horizontal o vertical que pueden usarse para dividir la ventana en grupos funcionales.



- Antes de agregar marcos y separadores gráficos evaluar la posibilidad de diseñar con identación y espaciado para reflejar la relación entre controles, es más limpio y claro.
- No aplicar marcos y separadores **para compensar un diseño pobre** de la disposición y alineación de los objetos de la interfaz.
- **No mezclar grupos de objetos enmarcados y no enmarcados** en la misma ventana.
- **No anidar un marco dentro de otro**, esto sobrecarga la disposición visual de los elementos.

➔ Pautas de diseño de secuencia de control de una aplicación

En aplicación la mejor opción es **partir de lo general a lo particular**. Establecer un fácil acceso y visibilidad de las tareas más comunes al usuario. Por ejemplo, en un editor de texto la opción *nuevo documento* debe presentarse de manera clara y explícita cuando lanzamos dicho programa.

En un programa de gestión, el botón o acceso al módulo de *clientes, productos...* deben prevalecer sobre opciones de uso menor como obtención de listados o facturas.

En resumen la secuencia de control de una aplicación no es muy diferente de cualquier otro acto de naturaleza humana: “*antes de comer debemos o nos deben adquirir los alimentos y/o preparar la comida*”.

Las **pruebas de usabilidad son una forma de medir el grado de corrección** en que una persona puede usar un objeto hecho por el hombre, como puede ser una página web, una interfaz de usuario, un documento o un dispositivo.

Las pruebas de usabilidad **consisten en seleccionar a un grupo de usuarios de una aplicación y solicitarles que lleven a cabo las tareas para las cuales fue diseñada**, en tanto el

equipo de diseño, desarrollo y otros involucrados toman nota de la interacción, particularmente de los errores y dificultades con las que se encuentren los usuarios.

No es necesario que se trate de una aplicación completamente terminada, pudiendo tratarse de un prototipo y de hecho es recomendable para ver la evolución de la aplicación y la adaptación del futuro o futuros usuarios a la misma.

Las métricas de usabilidad son:

- **Exactitud:** número de errores cometidos por los sujetos de prueba y si estos fueron recuperables o no al usar los datos o procedimientos adecuados.
- **Tiempo:** requerido para concluir la actividad.
- **Recuerdo:** qué tanto recuerda el usuario después de un periodo sin usar la aplicación.
- **Respueta emocional:** Cómo se siente el usuario al terminar la tarea (bajo tensión, satisfecho, molesto, etcétera).

Las pruebas pueden ser:

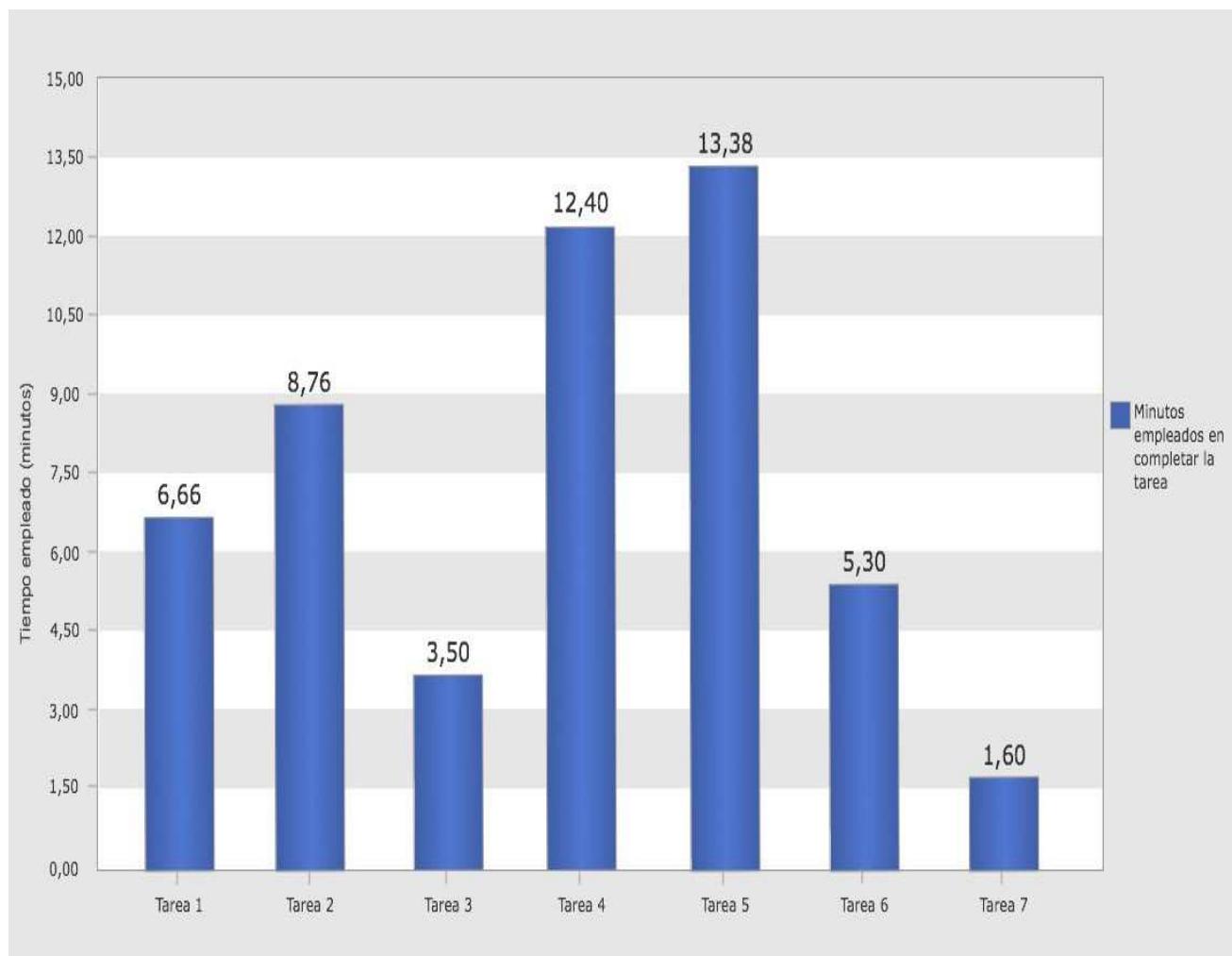
- **Pruebas informales:** los usuarios se consideran indirectamente.
 - Ventaja: Rapidez.
 - Desventaja: los problemas pueden no ser verdaderos problemas de usabilidad
- **Pruebas formales:** se involucran directamente los usuarios.
 - Ventaja: Contacto directo con los usuarios.
 - Desventaja: precisa de mayor elaboración de las pruebas

Algunos de los métodos que se pueden utilizar son:

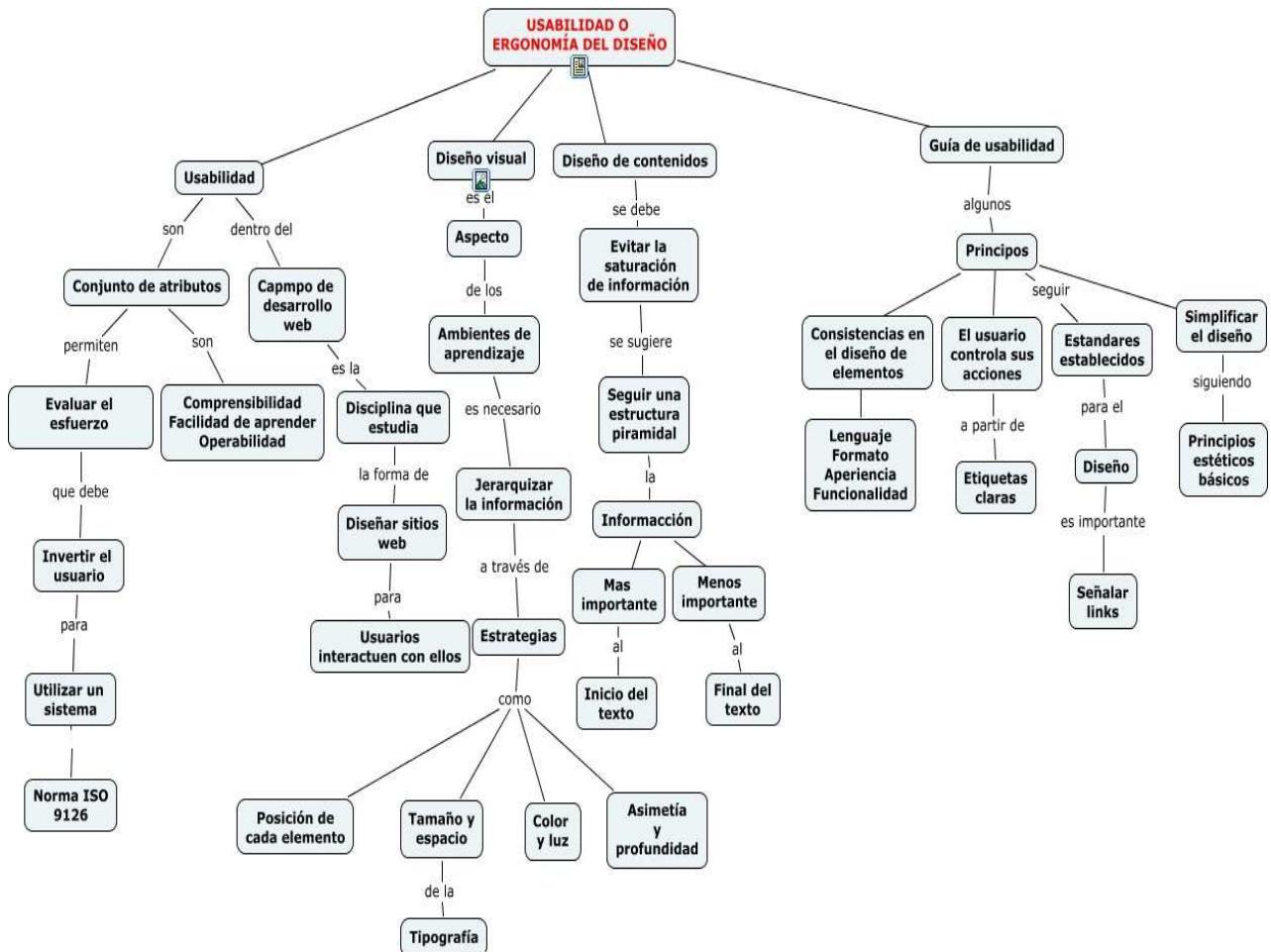
- **Entrevistas:** recogida de la información que aportan los usuarios finales de la aplicación llevando a cabo entrevistas personales con ellos antes y después del manejo de la UI
- **Encuestas:** la recogida de información se realiza mediante encuestas elaboradas sobre aspectos de la aplicación
- **Thinking-aloud:** se observan las reacciones y comentarios de forma aséptica de los usuarios durante el uso de los mismos del UI.
- **Grabaciones:** y posterior estudio de las mismas, similar a la anterior.
- **Seguimiento ocular:** se observa las reacciones de la mirada del usuario durante el uso de la UI.
- **Medidas de rendimiento:** es el ejemplo que proponemos a continuación.

Para medir la usabilidad hay varios métodos propuestos por diferentes autores pero uno sencillo abarcaría los siguientes pasos:

1. seleccionar las tareas que tienen que hacer: *dar de alta, bajar y modificar un cliente e imprimir un listado...*
2. seleccionar el número de personas, no menos de 15 o 20 que realizarán la misma tarea
3. elaborar cuestionario de recogida de datos, tiempos para llevar a cabo una determinada acción, número de errores al realizar una operación, consultas al desarrollador...
4. elaborar gráficas de control de tiempo y comparar resultados.



Datos de tiempo de realización de una serie de tareas.



ACTIVIDADES

Propuesta 1. Comenta los errores de diseño que se muestran en los siguientes formularios:

A.

The screenshot shows a window titled "Clientes" with a blue header bar and a green globe icon in the background. The main area contains several input fields for client information:

- Id. de cliente:** ALFKI
- Nombre de compañía:** Alfreds Futterkiste
- Nombre del contacto:** Maria Anders
- Cargo:** Representante de ventas
- Dirección:** Obere Str. 57
- Ciudad:** Berlín
- Región:** [empty field]
- Código postal:** 12209
- País:** Alemania
- Teléfono:** 030-0074321
- Fax:** 030-0076545

At the bottom, there is a navigation bar labeled "Registro" with arrows and a page number indicator "1 de 91".

B.

The screenshot shows a form titled "html5 forms demo" on a red background. The form consists of several input fields:

- First Name:**
- Last Name:**
- Email address:**
- Date of birthday (we like to send presents!):**
- Country:**
- How many computers do you have at home?**

Below the form, there is a small note about spam and a "Sign me up!" button.

We love spam, and we'll share your email address with all our third-party friends. Heck, we'll even sell it! If you're happy to receive annoying email on a regular basis, please click submit...
 denotes a required field.

Sign me up!

C.

filtrado de movimientos

3026-6982-65-1456865036

| | |
|--|---|
| <input type="radio"/> de hoy <input checked="" type="radio"/> por fechas: inicio: dia 12 mes 5 año 02 fin: dia 21 mes 8 año 02 | <input checked="" type="checkbox"/> por signo: <input checked="" type="checkbox"/> haber <input type="checkbox"/> debe |
| <input checked="" type="checkbox"/> por importes: imp. mínimo: <input type="text" value="500"/> imp. máximo: <input type="text"/> | <input type="checkbox"/> por documento: nº documento: <input type="text"/> referencia 1: <input type="text"/> referencia 2: <input type="text"/> |
| <input type="checkbox"/> información complementaria | |
| <input checked="" type="checkbox"/> por concepto: común: 003~DOMICIL.-RECIBOS-LETRA-PAGOS POR SU CTA. <input type="checkbox"/> propio: ***~Todos <input type="checkbox"/> | |
| cancelar borrar aceptar | |

D.



UD5. Confección de informes

RA5. Crear informes, para lo que se utiliza y evalúa herramientas gráficas.

- CA5.1. Establecer la estructura del informe.
- CA5.2. Se diseñan informes básicos a partir de una fuente de datos mediante asistentes.
- CA5.3. Se establecieron filtros sobre los datos en los que se basan los informes.
- CA5.4. Se incluyen valores calculados, recuentos e totales.
- CA5.5. Se incluyen gráficos generados a partir de los datos.
- CA5.6. Se utilizan herramientas para generar el código correspondiente a los informes de una aplicación.
- CA5.7. Se modifica el código correspondiente a los informes.
- CA5.8. Se desarrolla una aplicación que contiene informes incrustados.

BC5. Confección de informes

- Informes incrustados y no incrustados en la aplicación.
- Herramientas gráficas integradas en el IDE y externas a él.
- Estructura general: secciones. Uso de agrupamientos. Filtración de datos.
- Numeración de líneas, recuentos e totales. Valores calculados.
- Librerías para la generación de informes: clases, métodos e atributos.
- Informes con parámetros.
- Conexión coas fuentes de datos y ejecución de consultas.

➔ Informes incrustados y no incrustados en la aplicación

Al planear la creación de una aplicación una de las consideraciones más importantes es la posibilidad de la inclusión de informes.

Los informes de una aplicación se pueden dividir en dos grandes grupos desde el punto de vista de su diseño e inclusión en la misma: **informes incrustados e informes no incrustados**.

Un **informe incrustado** es un informe que se ha importado a un proyecto, o que se ha creado en él. Cuando se incrusta un informe en el proyecto, automáticamente **se genera una clase contenedora para el informe**.

Cuando se importa o se crea el informe en el proyecto, se crea una clase contenedora, con el nombre del informe. Cuando ocurre esto, todo el código del proyecto interactúa con la clase del informe que se ha creado para representarlo, en vez de hacerlo con el propio archivo de informe original.

En el caso de que sea una aplicación compilada, tanto el informe como su clase contenedora se incrustan en el ensamblado, lo mismo que ocurriría con cualquier otro recurso, módulo, paquete o clase del proyecto.

En Microsoft Visual Studio tienen la clase contenedora denominada **ReportDocument** y en Java hay varias opciones pero quizás la más utilizada es **JasperReports**. Una de las herramientas más utilizadas en el campo de la ofimática, el **ACCESS y su homólogo en Openoffice BASE**, generan también sus propios informes incrustados

Un **informe no incrustado** es un informe **externo al proyecto**. Existen muchas formas de tener acceso al informe y cargarlo en el proyecto, para habilitar la interacción con el informe mediante programación, pero el informe siempre continuará siendo externo.

A un informe no incrustado siempre se obtiene acceso externamente y se puede tener acceso a él de diversas formas:

- El informe puede estar en la unidad de disco duro en una ruta de directorio de archivos.
- El informe puede estar expuesto como servicio Web de informes y obtenerse en formato HTML o XML.
- El informe puede formar parte de un grupo de informes expuestos a través de herramientas externas. **Crystal Reports**, el generador de informes de mayor influencia, puede trabajar y generar informes no incrustados.

Nunca se importan informes no incrustados en el proyecto y, por lo tanto, **nunca se crea ninguna clase contenedora de informe**, a diferencia de los informes incrustados. En su lugar, se carga el

informe no incrustado en uno de los modelos de objetos en tiempo de ejecución.

¿Cuándo elegir informes incrustados o no incrustados?

Si desea simplificar la implementación del proyecto es preferible **informes incrustados**. El problema de estos es que no todos los lenguajes y sus correspondientes entornos de desarrollo soportan la creación de informes o tienen la herramienta o asistente para llevar a cabo su diseño.

Los **informes incrustados** tendrán menos archivos con los que trabajar y no se tendrá que preocupar de si los informes están mal colocados en la ruta de directorio de archivos equivocada. Además, esta solución es **aconsejable** siempre y cuando los informes no se exponen a modificaciones tras el compilado del proyecto.

Los informes **no incrustados son más sencillos y seguros**, pero requieren **más trabajo**. Podemos modificar su diseño sin necesidad de volver a compilar el proyecto.

Si los informes **se deben modificar regularmente**, mejor utilizar informes no incrustados para facilitar su acceso y modificación, sin preocuparse por la necesidad de volver a compilar los ensamblados cada vez. Además, existen **límites en el tamaño que puede tener un informe incrustado**. Además, los informes no incrustados tienen **ventajas de escalabilidad**.

```
private ReportDocument ChooseReport(int i)

{
    switch(i)
    {
        case 1:
            Chart chartReport = new Chart();
            return (ReportDocument)chartReport;
        case 2:
            Hierarchical_Grouping hierarchicalGroupingReport = new
            Hierarchical_Grouping();
            return (ReportDocument)hierarchicalGroupingReport;
        default:
            World_Sales_Report worldSalesReport = new
            World_Sales_Report ();
            return (ReportDocument)worldSalesReport;
    };
}
```

Ejemplo de Código Generador de un Informe Incrustado en C#

```
string reportPath="C:\WWFiles\Program\VisualStudio" + "CrystalReports\Report\Business\"
                  "SalesReport.rpt";

ReportDocument reportDocument = new ReportDocument();
```

Ejemplo de enlace con un Generador de un Informe no Incrustado

→ Herramientas gráficas integradas en el IDE y externas a él

- **Crystal Reports:** es una aplicación utilizada para diseñar y generar informes desde una amplia gama de fuentes de datos.

Varias aplicaciones, como Microsoft Visual Studio, incluyen una versión OEM de *Crystal Reports* como una herramienta de propósito general para informes/reportes. **Crystal Reports** se convirtió en el generador de informes estándar cuando Microsoft lo liberó con Visual Basic. Permite incluir datos de varias tablas, la inclusión de sentencias SQL y de estructuras condicionales e iterativas. El formato de salida puede ser .pdf, .rpt, .xls, .txt y .CSV⁴ entre otros. CrystalReports es quizás la herramientas más utilizada en programación para la obtención de reportes.

- **JasperReports** es una herramienta de creación de informes que tiene la habilidad de entregar contenido enriquecido al monitor, a la impresora o a ficheros PDF, HTML, XLS, CSV y XML. Está escrito completamente en Java y puede ser usado en gran variedad de aplicaciones de Java, incluyendo J2EE o aplicaciones web, para generar contenido dinámico. Su propósito principal es ayudar a crear documentos de tipo páginas, preparados para imprimir en una forma simple y flexible. JasperReports se usa comúnmente con **iReport**, un front-end gráfico de código abierto para la edición de informes. Se encuentra bajo licencia libre GNU, por lo que es **Software libre**. Forma parte de la iniciativa opensource Lisog.
- **ReportLab** es una librería para la obtención de reportes o informes en Python. Es software libre. Es uno de los más utilizados por los desarrolladores en Python para la elaborar los informes de sus proyectos.
- **Datareport** es una herramienta eficaz y es fácil crear informes complejos arrastrando y soltando campos fuera de la ventana de entorno de datos. Es de muy fácil uso, sin apenas necesidad de implementar código y se encuentra en el entorno de desarrollo Visual Net de Microsoft. Se utiliza para aplicaciones no demasiado complejas pero no tiene la potencialidad de CrystalReports.
- **R&OS o pdf class** es una clase para PHP que provee métodos potentes y simplificados para la creación de archivos PDF. Otra clase útil es PHP es **PHPReport Generator**.
- **Geraldo:** es un motor de informes para aplicaciones de Python o Django. Utiliza

⁴ Los ficheros **CSV** (del inglés *comma-separated values*) son un tipo de documento en formato abierto sencillo para representar datos en forma de tabla, en las que las columnas se separan por comas (o punto y coma en donde la coma es el separador decimal: España, Francia, Italia...) y las filas por saltos de línea. Los campos que contengan una coma, un salto de línea o una comilla doble deben ser encerrados entre comillas dobles, por ejemplo, → luis, 98789, vigo

ReportLab para generar informes con encabezado y pie de página.. Informe comenzar y bandas de resumen, aggregation y elementos gráficos, etc.. También es sencilla de utilizar.

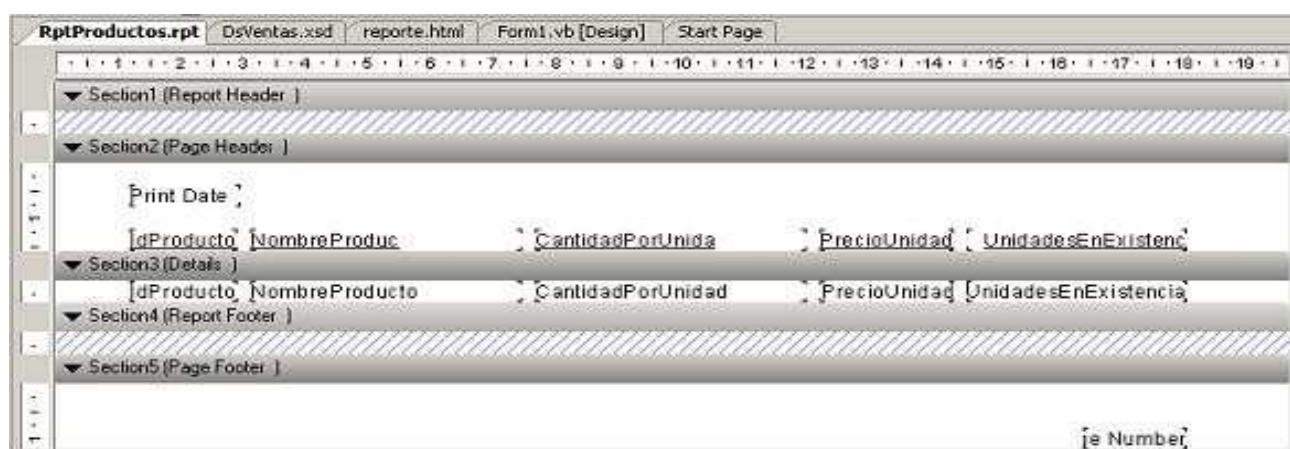
- **pyfpdf:** el que utilizamos en los ejemplos destaca por su sencillez. Utiliza una librería para PHP (FPDF) adaptada a python. La librería **xhtml2pdf** permite obtener, utilizando pyfpdf, informes en html.
- **Docutils** es un sistema de texto de código abierto para el procesamiento de documentos de procesamiento de texto plano en formatos útiles, tales como HTML, LaTeX o XML. Incluye **reStructuredText**, es fácil de leer, usar **lo-que-ve-es-lo-que-hay** (WYSWYG).

Existen algunas herramientas más dedicadas a la generación de informes en diferentes formatos pero las más utilizadas son las mencionadas.

→ Estructura general: secciones. Uso de agrupamientos. Filtrado de datos

Básicamente un informe, de forma general, contiene las siguientes secciones:

- **Sección 1: Report Header.** *Cabecera del informe*, donde se imprime una sola vez al inicio del reporte. Puede contener desde el *logo de la empresa, fecha,..*
- **Sección 2: Page Header.** *Cabecera de pagina*, donde se imprime al inicio de cada pagina impresa. Puede contener *anotaciones generales*, por ejemplo si es una factura, el *nombre y dirección del cliente, número de factura*.
- **Sección 3: Details.** *Detalle del reporte*, donde las filas o registros que conforman el reporte. Es allí donde se alojan los *campos del origen de datos*.
- **Sección 4: Report Footer.** *Pie del reporte*, donde se imprime una sola vez al finalizar el reporte. Se utiliza esta sección para imprimir los *totales generales, promedios...*
- **Sección 5: Page Footer.** *Pie de pagina*, donde se imprime al final de cada pagina. Se utiliza esta sección para imprimir la *paginacion, los totales por pagina*.



Aunque no obligatoriamente debe contener todas las secciones si es aconsejable que se mantenga una estructura por secciones por su sencillez y posibilidad a la hora de utilizar dicha estructura en otros informes.

Los datos que aparecen en el informe finalizado dependen de las **opciones de organización**. En particular, los datos del informe varían según las secciones en las que desee insertar objetos de informe concretos. Por ejemplo, si inserta un objeto de gráfico en la sección *Encabezado de informe*, el gráfico sólo aparecerá una vez al principio del informe y resumirá los datos que contiene el informe. Además, si un objeto de gráfico se añade a la sección *Encabezado de grupo*, aparecerá un gráfico individual al principio de cada grupo de datos y sólo se resumirán los datos relacionados con dicho grupo.

Los informes necesitará separar los datos en grupos para **facilitar la lectura y el análisis**. Se puede agrupar los datos de un informe **para que muestren relaciones jerárquicas**. Al agrupar datos jerárquicamente, se ordena la información según la relación entre dos campos. En los datos que se utilicen para el informe debe ser inherente una relación jerárquica.

- Para que el programa reconozca una relación entre los campos principal y secundario, ambos deben pertenecer al mismo tipo de datos.
- Los **datos del campo principal deben ser un subconjunto de los datos del campo secundario**.
- Para que el nivel superior de la jerarquía aparezca en el informe, el valor debe aparecer en los datos secundarios y la fila correspondiente de los datos principales debe estar vacía.
- **No puede existir lógica circular en los datos** (es decir, A no puede estar relacionado con B si B está relacionado con C y C está a su vez relacionado con A).

Por ejemplo, si se desea mostrar la estructura jerárquica de un departamento, se puede agrupar datos por **Id. de empleado y especificar la jerarquía usando el campo de datos que muestra a quién informa el empleado**. Otro agrupamiento sería listado de empleados por departamento siendo el campo principal el **id del departamento** y el secundario los empleados que a él pertenecen.

Se puede definir filtros en una región de datos para **seleccionar o excluir partes del conjunto de datos que utiliza la región de datos**. Los filtros limitan los datos que se muestran al usuario tras la recuperación de todos los datos. Dado que se recupera el conjunto completo de datos y luego se filtra cuando se procesa el informe, es posible que el informe no sea tan bueno como cuando el informe obtiene datos filtrados de otra manera (específicamente, si escribe un código que filtra datos antes de que pasen al informe). De esta forma en los agrupamientos pueden ser interesantes filtrados de datos, como por ejemplo listado de empleados por departamentos que obtuvieron un mínimo de X ventas o que pertenezcan a tal o cual ciudad.

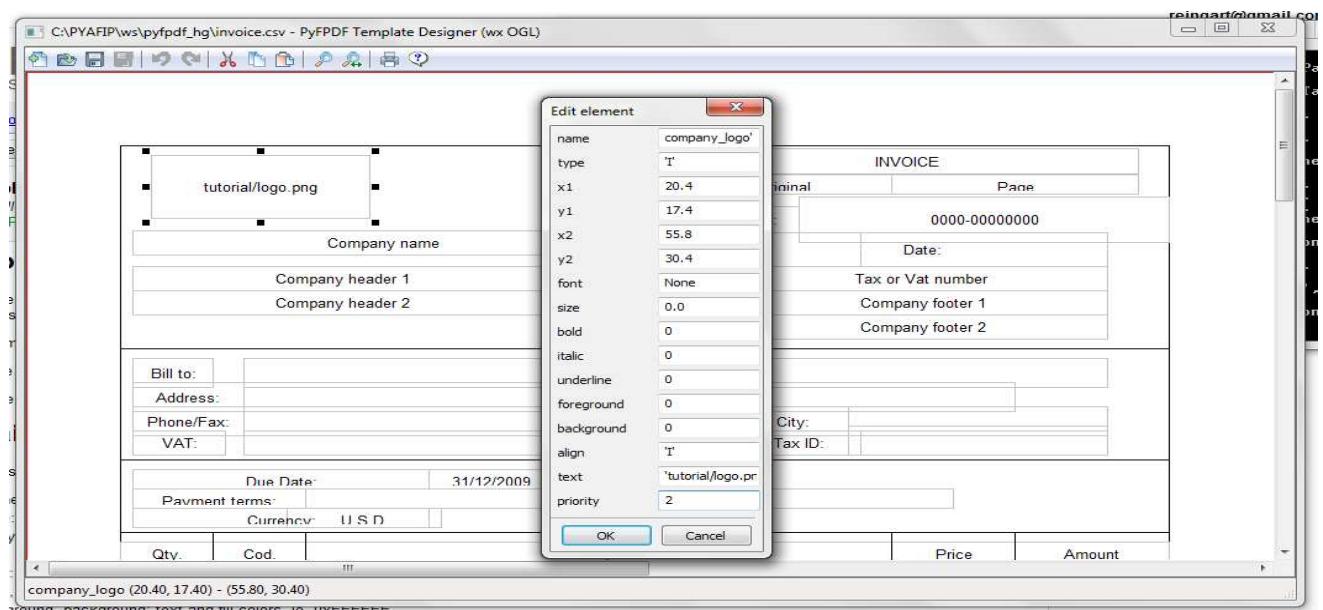
| Título | Idioma | Género | Director |
|----------------------|--------|----------|--------------------|
| Los Padres de Ella | Inglés | Comedia | Alex de la Iglesia |
| La Bella y la Bestia | Inglés | Infantil | Gary Trousdale |

Página 1 de 1

→ Numeración de líneas, recuentos e totales. Valores calculados.

Cuando calculamos un **resumen y/o totalizamos** los datos, hay que **dividir en grupos** y realizar la operación especificada sobre los valores de cada grupo. Todo lo que hay que hacer es especificar lo siguiente:

- el **campo** que deseamos resumir o totalizar,
- el tipo de operación que va a realizar con ese campo: **media, unidades x precio-unidad, subtotal, iva x subtotal, iva + subtotal....**
- el campo que va a desencadenar un nuevo grupo siempre que cambie su valor, por ejemplo, **modificaciones del iva, descuentos pronto-pago...**
- el orden de clasificación.



Plantilla de Web2Py para realizar informes con PyFPDF

Para insertar un subtotal (sólo en campos numéricos), o totales determinadas herramientas como **CrystalReports** tienen asistentes que facilitan dichos cálculos. En otros casos, como en Python, debemos generar mediante código la obtención de dichos resultados para incluirlos en el informe final. De hecho el lenguaje SQL permite que determinadas operaciones sean obtenidas en las sentencia de consulta, por ejemplo, el count, la media...

➔ **Librerías para la generación de informes: clases, métodos y atributos**

Generalmente la mayoría de los API para creación de informes se han utilizando herramientas integradas en el caso de IDE integrados. Se suele aconsejar reducir el uso de **sql + programación** para la generación de informes por alto coste de trabajo aunque es cierto que tiene la ventaja de reducir código redundante.

Existen en el mercado herramientas que integran las librerías necesarias para generar informes:

- *Microsoft SQL Server Reporting Services*
- *Crystal Reports*
- *BEA Portal Reporting Solutions*
- *MicroStrategy Report Services*
- *Actuate BIRT (Business Intelligence and Reporting Tools)*

Y otras que son Open Source:

- *Pentaho Report Designer*
- *OpenRPT para PostgreSQL*
- *Eclipse BIRT*
- *JasperReports*

La mayoría de las librerías que realizan la generación de informes establecen la siguiente arquitectura en su elaboración:

- **modelo de datos**
- **acceso a los datos**
- **presentación de los datos**

Para estudiar la estructura de clases de una herramientas de elaboración de informes utilizaremos **JasperReports**. Dicha herramienta elabora informes predefinidos en XML. Su estructura de clases es la siguiente:

- *net.sr.jasperreports.engine.jasperexportManager*
- *net.sr.jasperreports.engine.jasperprintManager*
- *net.sr.jasperreports.engine.jasperfillManager*
- *net.sr.jasperreports.engine.jaspercompileManager*
- *net.sr.jasperreports.engine.jasperdesignViewer*
- *net.sr.jasperreports.engine.jasperViewer*

Las cuatro primeras permiten manejar el **motor de generación del informe**, y las dos últimas se encargan de la **presentación o visualización** del mismo.

En primer lugar llevamos a cabo el diseño del informe que no es más que una “plantilla” que utilizará el motor del generación del informe. Como es un documento XML este **lleva un DTD** asociado que se encuentra en las librerías de JasperReport. Su estructura es similar a la de cualquier documento de texto:

| | |
|---------------------|-------------------------------|
| title | Título del reporte |
| pageHeader | Encabezado del documento |
| columnHeader | Encabezado de las columnas |
| detail | Detalle del documento. Cuerpo |
| columnFooter | Pie de la columna |
| pageFooter | Pie del documento |
| sumary | Cierre del documento |

Una vez el diseño está listo llega el momento de la compilación que se encarga el método *compileReport* de la clase *compileManager* para posteriormente completar su contenido con el método *fillReport* de la clase *jasperFillManager* obteniendo los datos de la base de datos o con cálculos intermedios.

La clase *jasperManager* se encarga de llamar a la API adecuado sea ODBC, JDBC para obtener los datos. Finalmente *jasperPrint* se encarga de su preparación para imprimir, *jasperViewer* para visualizarlo en pantalla o *exportManager* para exportarlo a otro formato.

```

Map parametrosReporte;           // Mapa de parámetros del reporte
JasperDesign disenioReporte;    // diseño del reporte
JRDesignQuery queryReporte;    // consulta a la base para
                                // armar el reporte

// Armamos el MAP de parámetros adicionales a pasar al reporte
// (los nombres están determinados por el XML diseñado)
parametrosReporte.put("Identificacion", "121");
parametrosReporte.put("NumeroSecuencia", "01");

// Cargamos el diseño desde el archivo xml, la extensión por
// defecto: .jrxml
disenioReporte = JRXmlLoader.load("DisenoXML.jrxml");

```

Un ejemplo sería:

```

Map parametrosReporte;           // Mapa de parámetros del reporte
JasperDesign disenioReporte;    // diseño del reporte
JRDesignQuery queryReporte;    // consulta a la base para
                                // armar el reporte

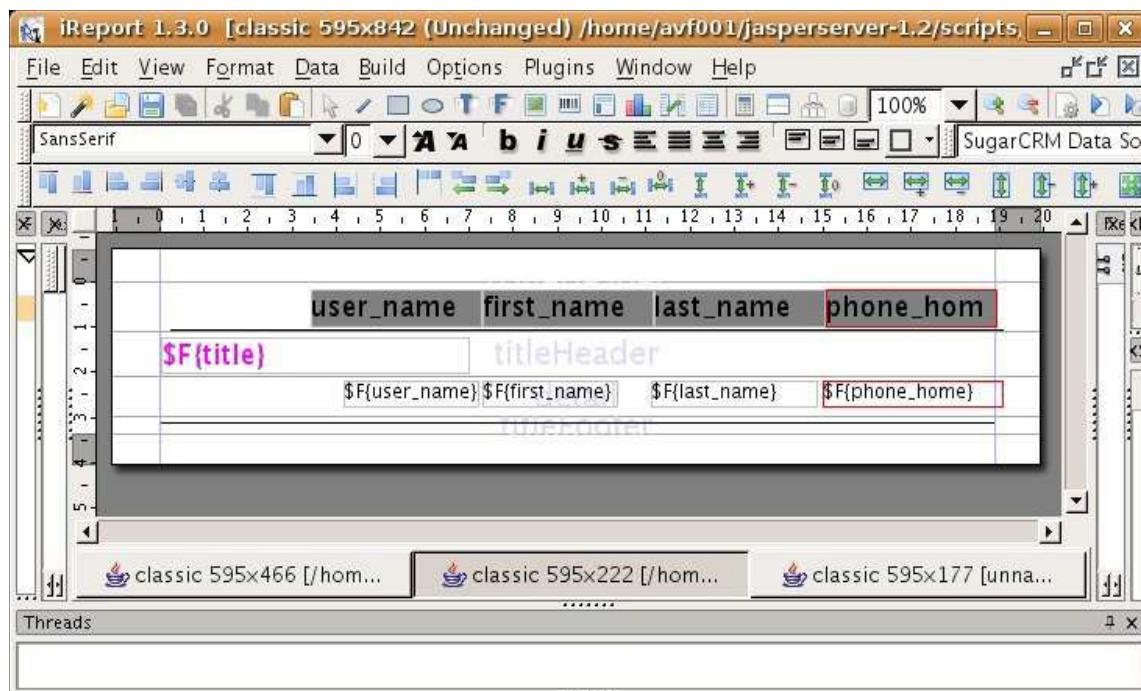
// Armamos el MAP de parámetros adicionales a pasar al reporte
// (los nombres están determinados por el XML diseñado)
parametrosReporte.put("Identificacion", "121");
parametrosReporte.put("NumeroSecuencia", "01");

// Cargamos el diseño desde el archivo xml, la extensión por
// defecto: .jrxml
disenioReporte = JRXmlLoader.load("DisenoXML.jrxml");

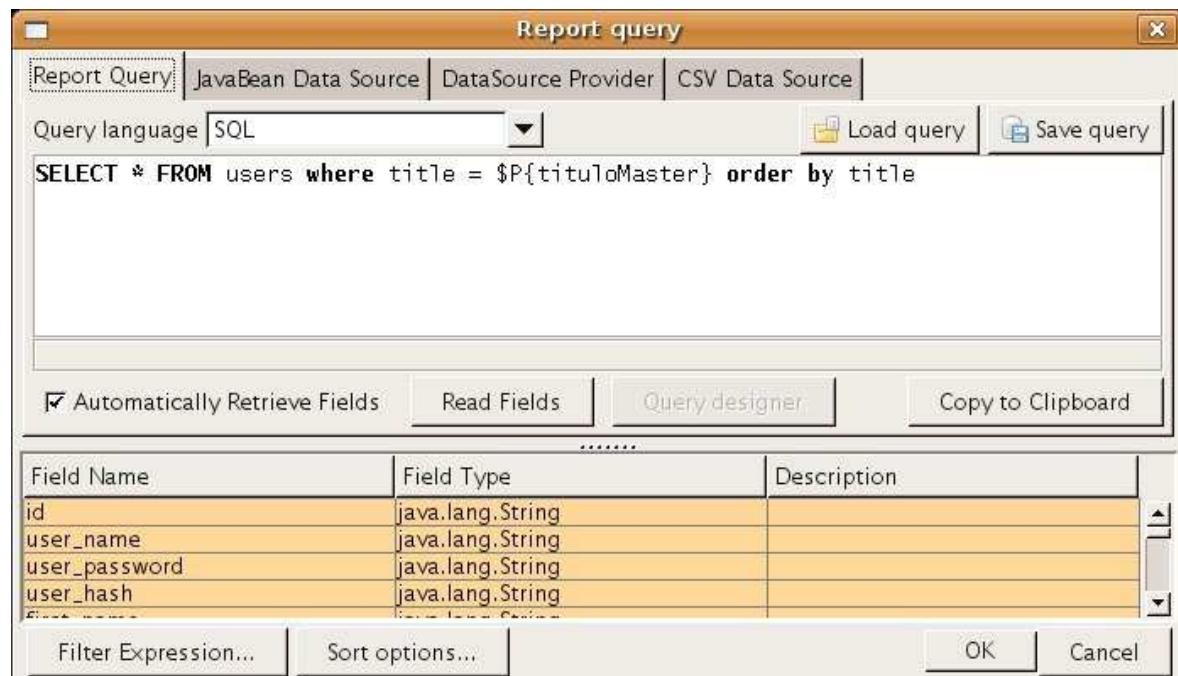
```

→ Informes con parámetros

Las consultas que dan lugar a informes son útiles para poder trabajar sólo con los campos de una tabla que corresponden a una tarea determinada. Cuando se desea limitar aún más los datos con los que se va a trabajar, basándose en el valor de un campo, se pueden usar **criterios en la consulta**. Los criterios son reglas que se incluyen en el diseño de una consulta. Estas reglas especifican valores o modelos con los que los campos deben coincidir o que los campos deben contener para que la consulta los devuelva.



Ejemplo de consulta basada en el parámetro título



Ejemplo de informe basada en el parámetro título de la consulta anterior

| user_name | first_name | last_name | phone_ho |
|--------------------|------------|-----------|-----------|
| Sales Manager East | | | |
| will | Will | Westin | Sin telf. |
| kristen | Kristen | Verona | Sin telf. |

Informante resultante

Cuando se desea que una consulta pida un valor o un modelo cada vez que se ejecuta, se puede crear una consulta con parámetros. La manera más sencilla de crear un informe que acepte parámetros **es usar una consulta de parámetros como origen de registros del informe**. Por ejemplo, puede crear un informe de ingresos mensuales basado en una consulta de parámetros que solicite un valor para el mes.

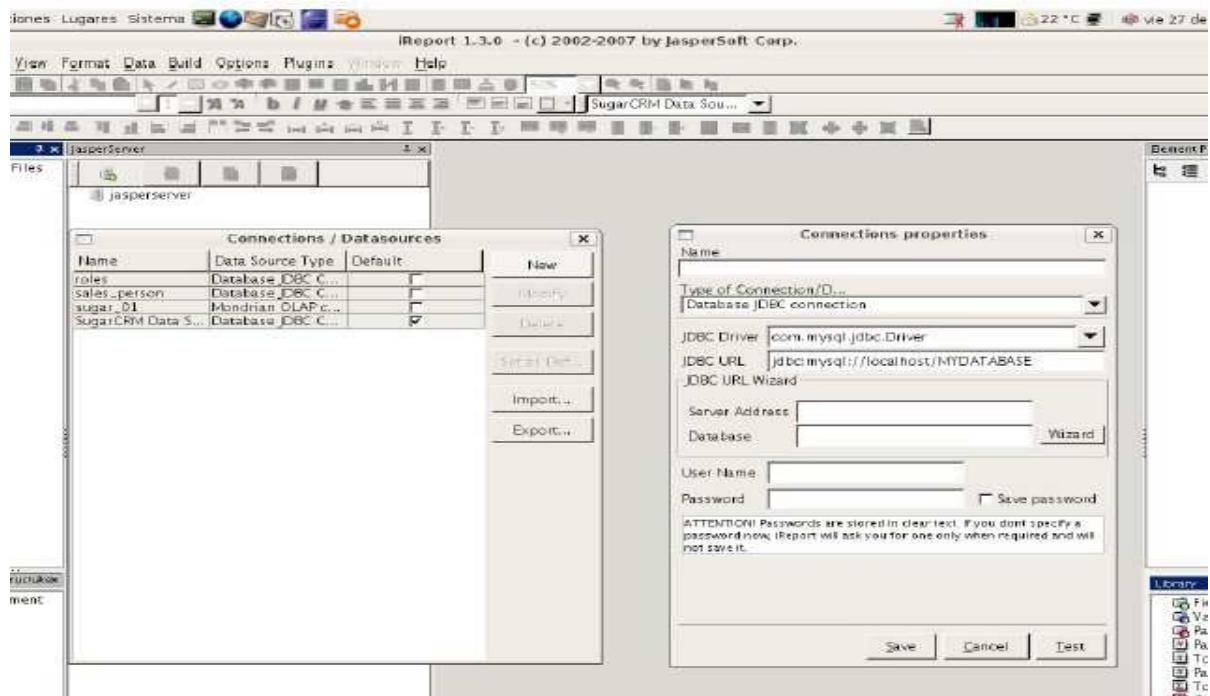
→ Conexión con las fuentes de datos y ejecución de consultas.

Las conexiones con las fuentes de datos son las mismas con las que se realizan las consultas. La diferencia de las diferentes herramientas de generación de informes radican en la automatización de dichas tareas.

Mientras en VB.NET un asistente puede guiar al programador en la conexión de datos, en Python, la conexión más habitual es la manual. Eso no implica que en el primer caso la podamos hacer manual, y en el segundo existan herramientas que lo permitan hacer de forma automática. En resumen, todos los lenguajes de programación, o casi todos, permiten ambas formas de conectar la aplicación la base o fuente de datos para la elaboración de consultas e informes detallados respectivamente.

En la siguiente figura podemos observar como JaserServer realiza una conexión a una base de datos de forma automatizada monstrándonos la herramienta utilizada para ello:

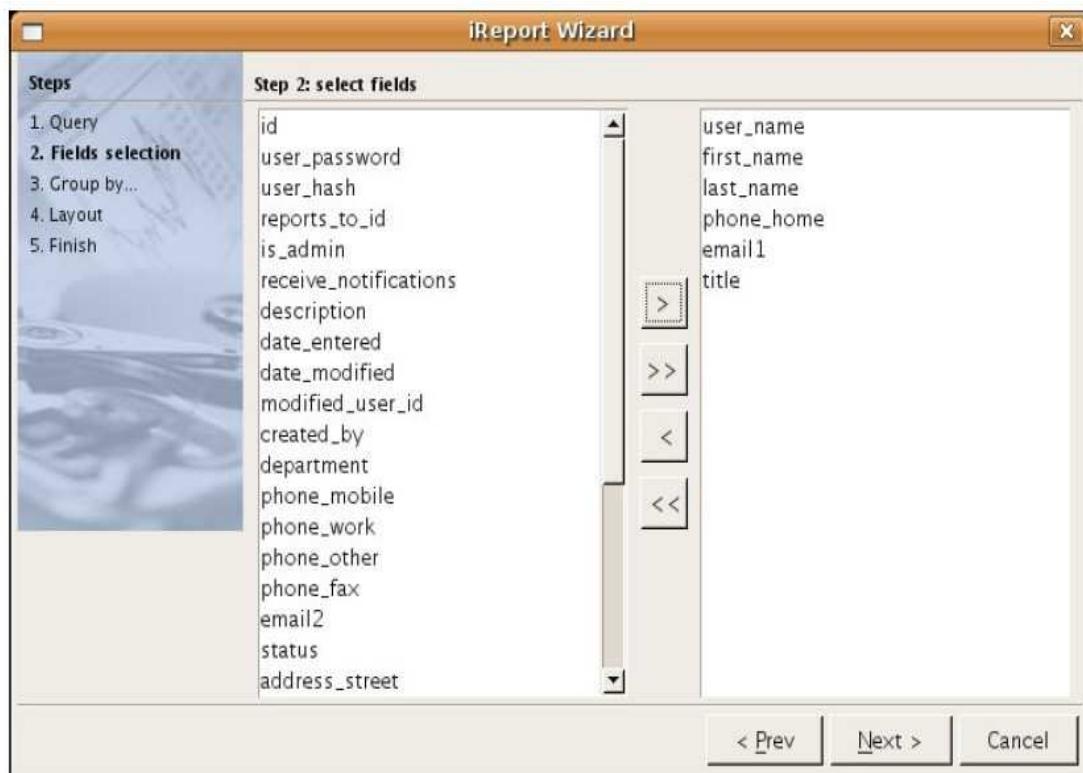
1. Conexión con la base de datos



2. Inicio del asistente



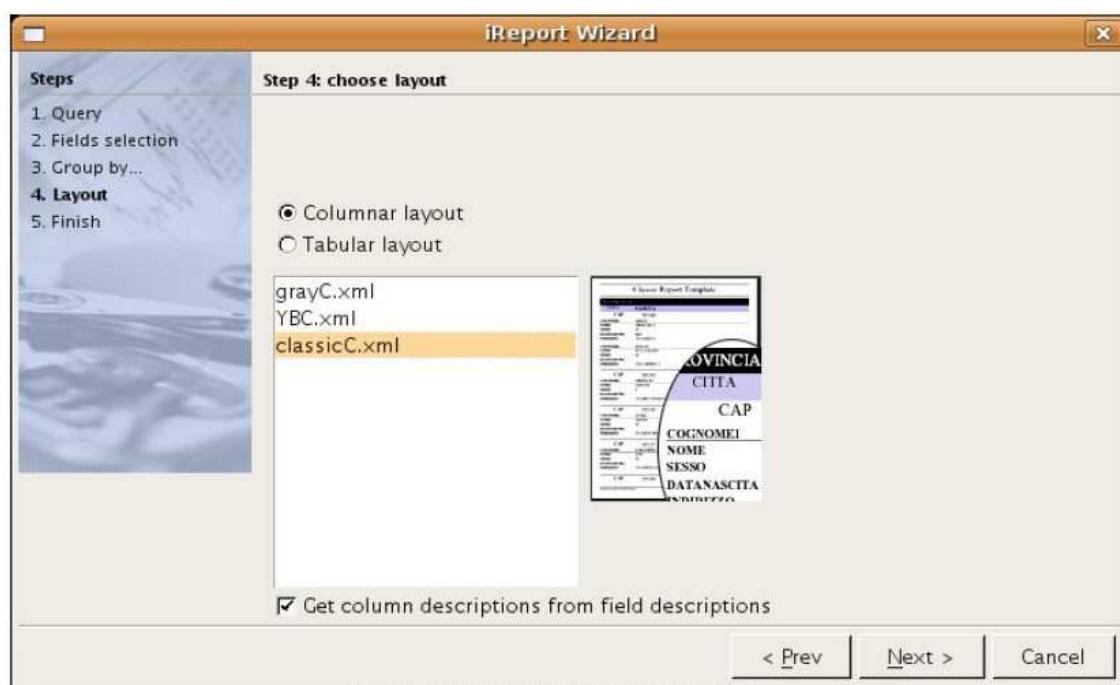
3. Seleccionamos los campos



4. Seleccionamos algún criterio o parámetro



5. Elegimos la presentación



La mayoría de los generadores de informes automatizados funcionan de una manera similar a esta.

ACTIVIDADES

Ejemplo. La siguiente actividad permite la instalación de **pypdf** una librería que permite la elaboración de informes con python. En el siguiente enlace de su creador podremos analizar las posibilidades que presenta dicha librería.

<https://code.google.com/p/pyfpdf/>

Su sencillez permite incluso añadir funcionalidades nuevas ya que su código es visible.

- **Instalación**

La instalación es manual. .

- **Descarga del paquete**

```
# wget http://pyfpdf.googlecode.com/files/fpdf-1.7.zip
```

- Instalar la herramienta *pip* que facilita a su vez la instalación de nuevas librerías en python.

```
# apt-get install python-distribute  
# apt-get install python-pip
```

- Finalmente instalamos la nueva librería. Esta se aloja en /usr/local/lib/python. Echar un vistazo par asegurarnos de que se instaló allí tras la siguiente ejecución:

```
# pip install fpdf-1.7.zip
```

- **Aplicación ejemplo**

A continuación hay dos códigos.

Uno para comprobar que todo va bien obtenido de la web del desarrollado del módulo.

```
from fpdf import FPDF  
pdf=FPDF()  
pdf.add_page()  
pdf.set_font('Arial','B',16)  
pdf.cell(40,10,'Hola Mundo!')  
pdf.output('tuto1.pdf','F')
```

Y el segundo código básico, pero que permite comprobar su funcionamiento.

```

import os
import sqlite3 as lite

# en los tutoriales de la web hacen llamada a pyfpdf pero si os vais a las librerías # en
realidad los módulos se guardan en un directorio llamados fpdf

from fpdf import FPDF

pdf=FPDF()

def reporte(pdf):
    try:
        pdf=FPDF()                      #pdf es la variable que guarda el informe
        pdf.add_page()
        pdf.set_font('Arial', size=12)
        #clientes.db es una base de datos ejemplo de tres campos
        conexion = lite.connect('clientes.db')
        cur = conexion.cursor()
        cur.execute('SELECT * FROM DATOS')
        rows = cur.fetchall()
        for row in rows:
            pdf.cell(20,10,str(row),0,1)   #vuelca los datos en el informe
        pdf.output('tutorial.pdf')         # crea el informe
        conexion.close()
    except lite.Error, e:
        print("Error %s: ")

    reporte(pdf)

os.system('/usr/bin/evince tutorial.pdf')      #evince es un lector pdf

```

Propuesta 1.

A continuación crearemos una sencilla aplicación que nos permite hacer una **listado de compras y una suma final** del gasto total:

A. Creamos una base de datos con una única tabla y los campos siguientes:

Código, Fecha, Concepto, Precio

B. Creamos una aplicación que nos permite insertar, modificar y borrar registros en dicha tabla.

C. Añadimos un botón nuevo en la aplicación que nos genere un informe que liste todas las compras y nos haga una suma final de lo gastado.

UD6. Documentación de las aplicaciones

RA6. Documenta aplicaciones, para lo que selecciona e utiliza herramientas específicas.

- CA61. Identificar sistemas de generación de ayudas
- CA6.2. Se generaron ayudas en formato habitual
- CA6.3. Se generan ayudas sensibles al contexto
- CA6.4. Se documenta la estructura de la información persistente.
- CA6.5. Se confecciona el manual de usuario e la guía de referencia.
- CA6.6. Se confecciona los manuales de instalación, configuración y administración.
- CA6.7. Se confeccionan tutoriales.

BC6. Documentación de aplicaciones

- Ficheros de ayuda: formatos.
- Herramientas de generación de ayudas. Ayudas genéricas y sensibles al contexto.
- Tablas de contenidos, índices, sistemas de búsqueda, etc. Integración da ayuda na aplicación.
- Tipos de manuales: manual de usuario, guía de referencia, guías rápidas, e manuales de instalación, configuración e administración. Destinatarios e estructura.
- Elaboración de tutoriales.

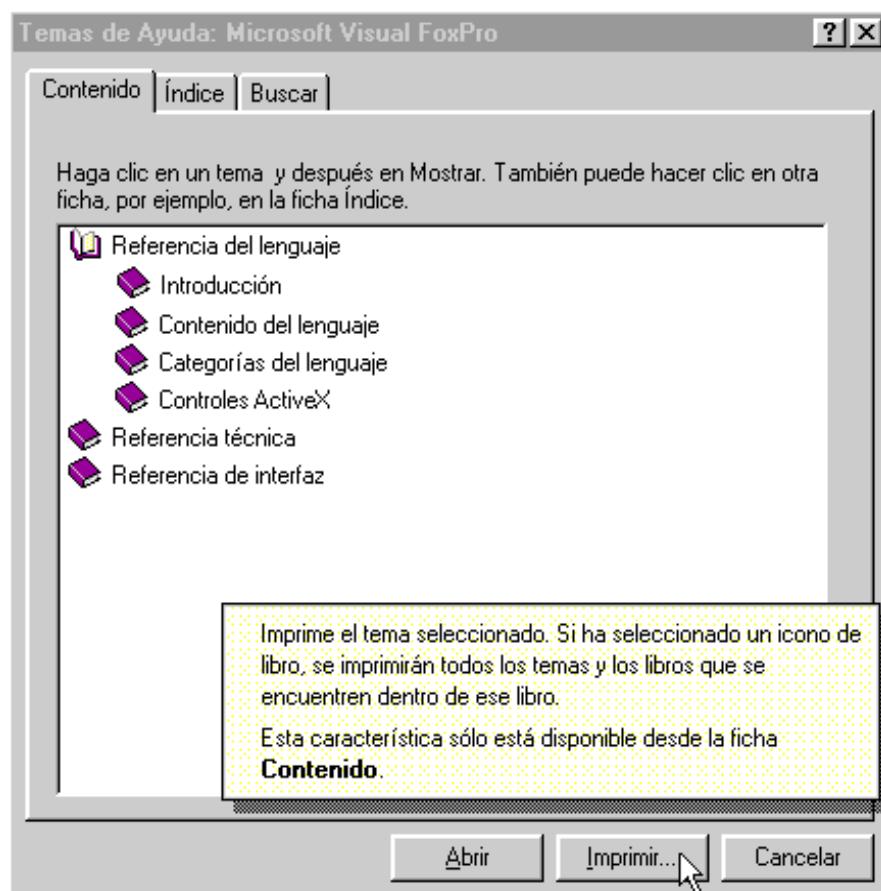
→ Ficheros de Ayuda. Formatos

Un **fichero de ayuda**, como su nombre indica, es un documento sobre **papel o digital** cuya finalidad es servir de guía de referencia, manual o ayuda a los diferentes usuarios de una aplicación.

Es importantes tener en cuenta que cuando nos referimos a futuros usuarios, también incluimos a posibles **desarrolladores** que puedan en un futuro modificar dicha aplicación con lo cual deberán en principio conocer el mecanismo de su uso.

Los formatos de ayuda más destacados, ciñéndonos a soporte digital, pueden ser:

- **pdf o cualquier formato ofimático:** son los más habituales y fáciles de realizar. Pueden ser abiertos por cualquier visor pdf en el primer caso o bien herramientas ofimáticas de carácter general en el segundo caso.
- **Winhelp (.hlp) :** Los archivos con extensión hlp son los archivos de **ayuda de Windows (el nombre de la extensión proviene de Help) u otros programas**. Al hacer doble click sobre un archivo con extensión hlp éste se abrirá con la Ayuda de Windows gracias al programa **winhlp32.exe**.

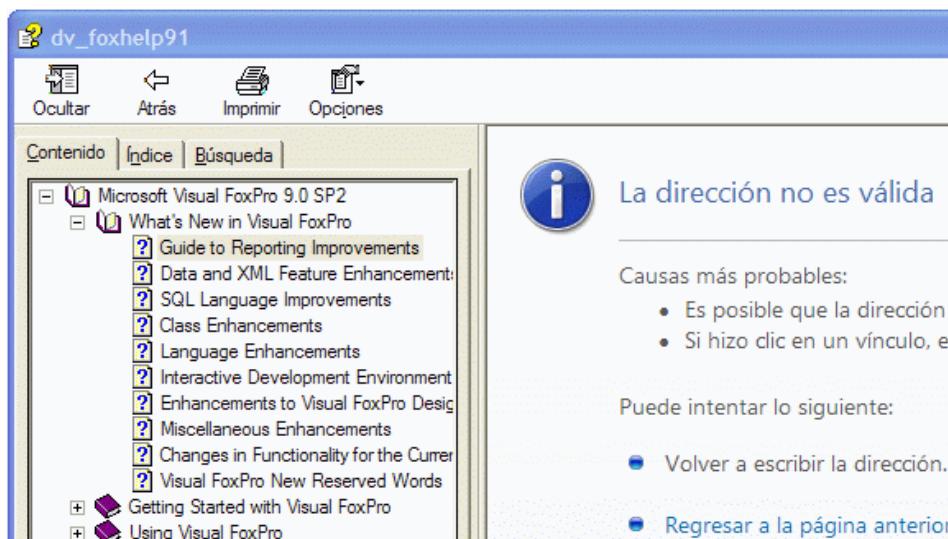


El formato de archivo se basa en el **formato de texto enriquecido (RTF)**. Fue la

plataforma de ayuda más popular desde Windows 3.0 a Windows XP. Posteriormente WinHelp fue eliminado en Windows Vista para fomentar el uso de formatos de ayuda más recientes como **.chm**.

Un archivo de WinHelp tiene como **".hlp"** sufijo. Puede ir acompañado por una tabla de contenido opcional (**.cnt**). Cuando Windows abre un archivo WinHelp, se crea un archivo **.gid** en el mismo directorio, que contiene información sobre el archivo **.hlp** entre la cual está el tamaño de la ventana y la ubicación en la pantalla. Si el usuario hace clic en la ficha "Buscar" permite la indexación de palabras clave.

- **chm: archivo de Ayuda de HTML Compilado** (Microsoft Compiled HTML Help en inglés) es un formato privativo de ayuda en línea desarrollado por Microsoft. Se publicó por primera vez en 1997 como sucesor del sistema de ayuda **winhelp**. Se popularizó con **Windows 98**, pero sobre todo se usó considerablemente hasta el sistema operativo **Windows XP**.



El archivo **.chm** consiste en un índice, una tabla de contenidos y un conjunto de páginas en **HMTL hiperenlazadas** a la tabla, que se compilan para generar el archivo de ayuda. Aplicaciones como **HTML Help Workshop**, de Microsoft, permiten compilar estos archivos.

En 2003, Microsoft anunció que debido a **fallos de seguridad que presentaba**, no lo iba a usar a partir de **Windows Vista** en adelante; sin embargo, aún aparece en muchas aplicaciones que corren en **Windows7**.

El archivo **.chm** consiste en un **índice, una tabla de contenidos y un conjunto de páginas** en **HTM hiperenlazadas** a la tabla, que se compilan para generar el archivo de ayuda. Aplicaciones como **HTML Help Workshop**, de Microsoft, permiten compilar estos

archivos. Los Archivos de Ayuda de HTML Compilado pueden contener páginas web con **código malicioso y ejecutarlas posteriormente, por lo que representan una amenaza a la seguridad**. El formato de archivo de Microsot Reader, .lit, es una derivación del .chm. Los archivos .chm a veces se utilizan como **e-books**.

- **Microsoft Asistencia Markup Language (AML Microsoft, generalmente se conoce como MAML)** es un lenguaje de marcado basado en **XML** desarrollado para proporcionar asistencia al usuario ("ayuda en línea") para el sistema operativo Microsoft Windows Vista y sucesivos. **MAML** también se utiliza para proporcionar información de ayuda para los cmdlets de PowerShell, módulos y funciones avanzadas. Fue sustituto para los archivos de ayuda de Windows desde la versión Vista.

MAML se aparta de todos los tipos anteriores de asistencia a los usuarios de los sistemas operativos de Windows. Anteriormente, la asistencia al usuario para los sistemas operativos Windows utiliza los archivos creados con una línea de comandos de compilador (**hhc.exe**).

Este compilador se utiliza para compilar una **.hhp** (proyecto) de archivos, **.hhc** (tabla de contenido) de archivos, **.hhk** archivo (índice), y una colección de archivos **HTML** y los temas relacionados con los recursos (CSS, JavaScript y archivos de imagen) en uno solo archivo **.chm**.

El aspecto más significativo de MAML es que cambia la producción de la ayuda al usuario para el concepto de autoría estructurada (algo similar a DITA o DocBook). Con MAML, **se hace hincapié en el contenido y las tareas que un usuario realiza con un ordenador**, no las características del software. La presentación se gestiona como parte del motor de renderizado cuando un usuario solicita un tema (como un motor de búsqueda).

La estructura de autoría MAML se divide en segmentos relacionados con un tipo de contenido:

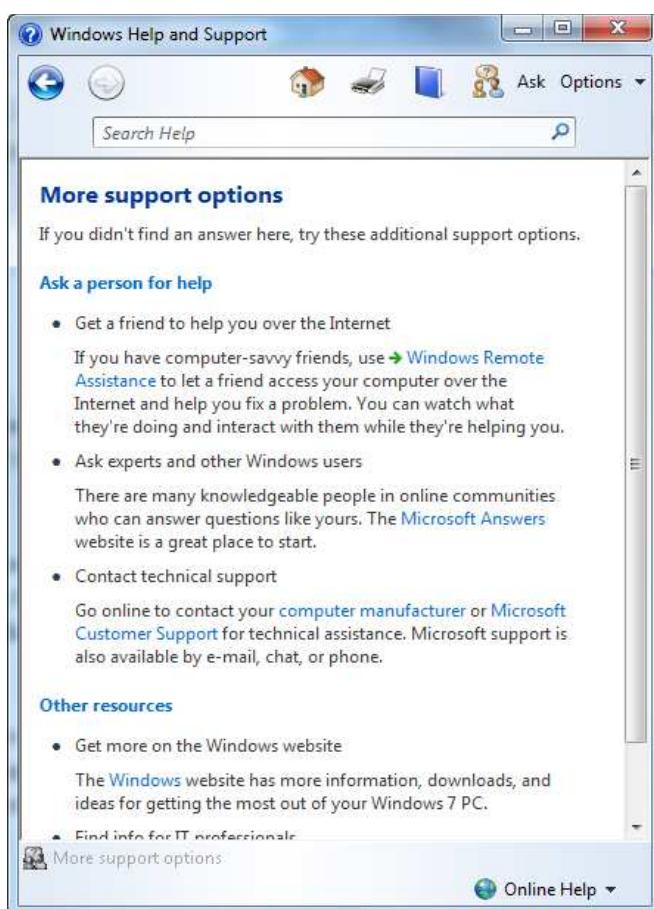
- Preguntas frecuentes conceptua
- Glosario
- Procedimiento de referencia
- Contenido reutilizable
- Tarea y solución de problemas
- Tutorial.

Hay tres niveles de transformación se producen cuando un tema se muestra: **estructura**,

presentación y representación:

- La **estructura** contiene contenido reutilizable y aplica la lógica condicional para determinar la estructura que debe tener el contenido cuando se muestra, y el contenido del texto en sí.
- La **presentación** permite que el contenido creado en MAML utilizar muchos formatos diferentes, incluyendo DHTML, XAML, RTF y material impreso.
- La **representación** se aplica hojas de estilo y visualiza el contenido final a los usuarios.

Desafortunadamente aún no existe una herramienta que permita la automatización total de la generación de las ayudas co MAML



- En cuanto a **Linux** los primeros ficheros de ayuda fueron los generados con el comando **man** para conocer el uso y opciones de los comandos.

Por otro lado están los comandos **whatis** que permite una breve descripción de los comandos y **apropos** que nos permite conocer los comandos relacionados con un determinado tema.

Por ejemplo, si escribimos *apropos password* nos mostrará en línea una serie de

comandos relacionados con el término `password`. Además la opción `comando --help` también nos permite acceder a las características de dicho comando.

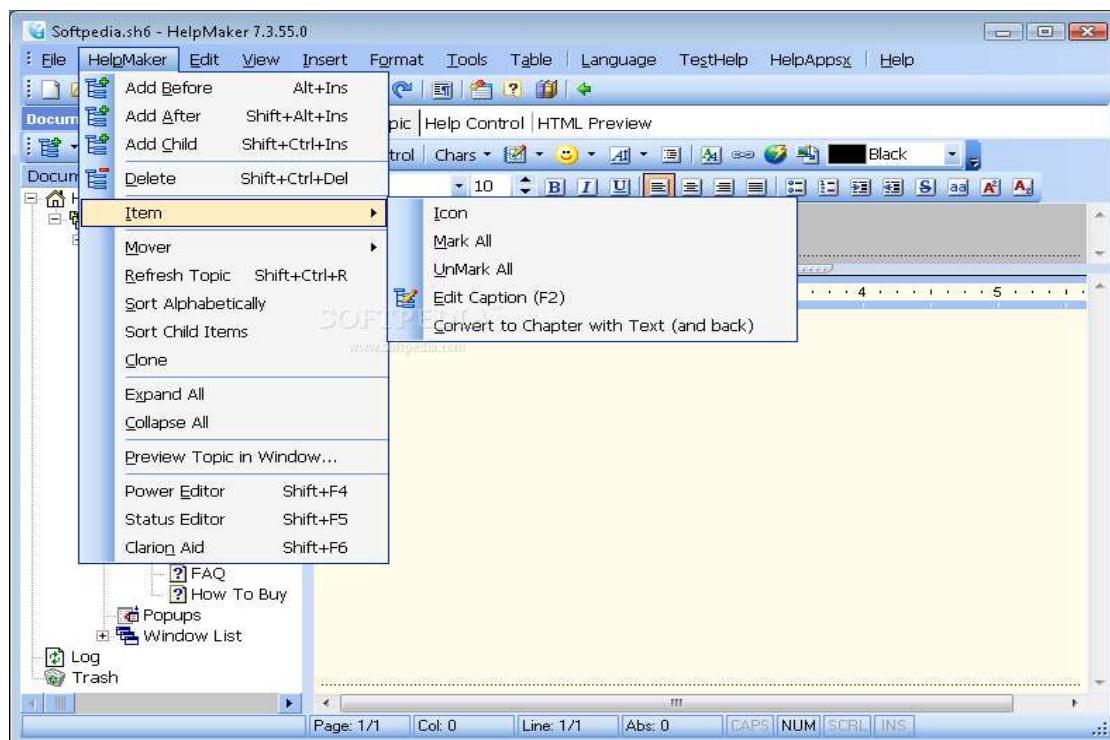
Finalmente podemos navegar en nuestro sistema de archivos hasta `/usr/share/doc` y buscar mas información como, changelogs, readmes específicos de distribución, archivos de ejemplo, etc, de la aplicación y/o utilería que estemos estudiando. Es en esta sección donde la mayoría de los programas utilizados en Linux guardan sus documentos de ayuda.

→ Herramientas de generación de ayudas. Ayudas genéricas y sensibles al contexto.

Si las ayudas están generadas en formatos `.pdf` u otro formato ofimático se utilizarían los editores de texto o documentos habituales.

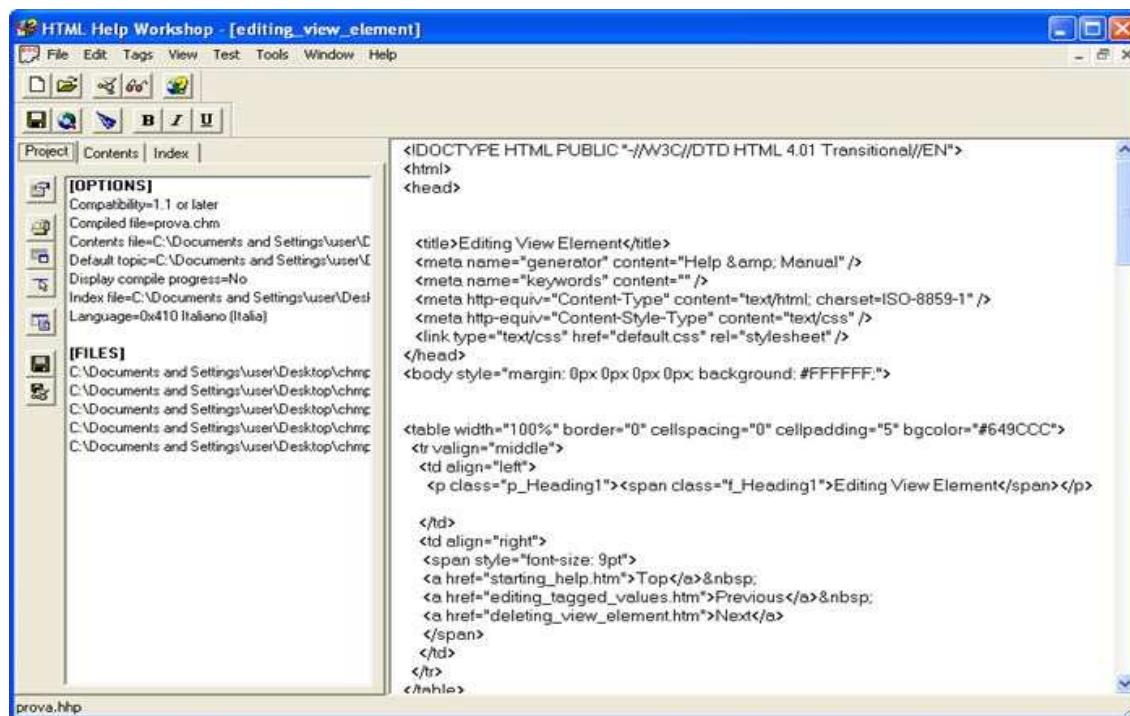
En el caso de que los formatos sean otros entonces tenemos:

- **HelpMaker:** es una aplicación para crear archivos de ayuda para programas. Permite la **creación de archivos de ayuda enteros**; en diferentes formatos, tales como: WinHelp, RTF (texto enriquecido) y HTML-Help. El programa permite crear la estructura de la ayuda, pudiendo cambiarla, ampliarla, editarla, añadir vínculos (incluso de una zona a otra de la misma ayuda), etc. También cuenta con un corrector ortográfico, que te vendrá muy bien si haces las ayudas también en inglés.



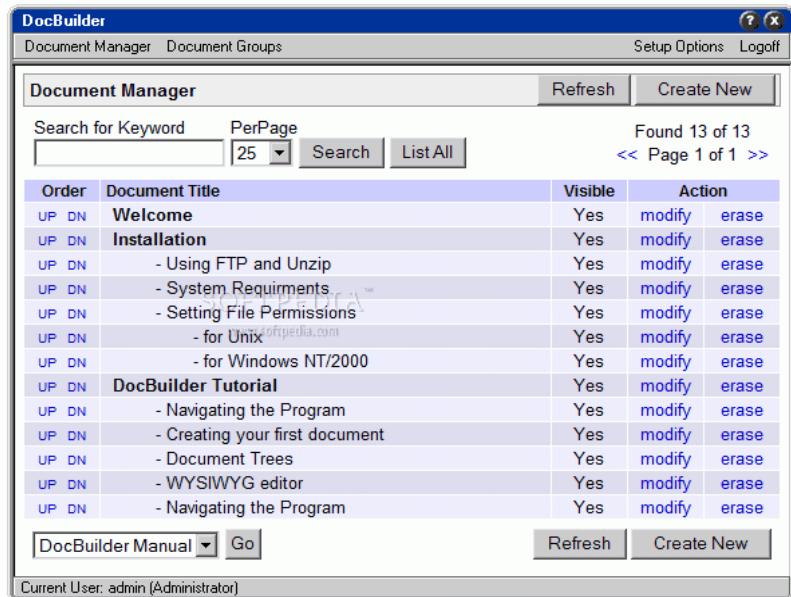
- **Microsoft HTML Help Workshop:** permite **crear ficheros de ayuda de windows (HLP)** y **páginas web que utilicen controles de navegación.** Crea estos ficheros y los distribuye con las aplicaciones. Incluye un administrador de proyectos, un compilador de ayuda y un editor de imágenes. HTML Help Workshop ofrece algunas ventajas sobre el estándar HTML, incluyendo la habilidad de implementar una tabla de elementos combinada y un índice, así como el uso de palabras clave para capacidades avanzadas de hiperenlazado. El compilador permite comprimir HTML, gráficos y otros ficheros en un fichero compilado **chm relativamente pequeño**, que puede ser distribuido junto a la aplicación o bien descargado desde Internet.

También se incluyen un control ActiveX y un applet de Java. El primero puede ser usado en cualquier navegador con soporte ActiveX, o en HTML Help Workshop, y el applet de Java puede ser utilizado en cualquier navegador que soporte Java. Con ambos controles puedes crear páginas web con soporte de ayuda HTML, incluyendo tablas de contenido, índices, y temas relacionados. Asimismo, crea ficheros compilados de ayuda para ser ejecutados directamente desde el web.

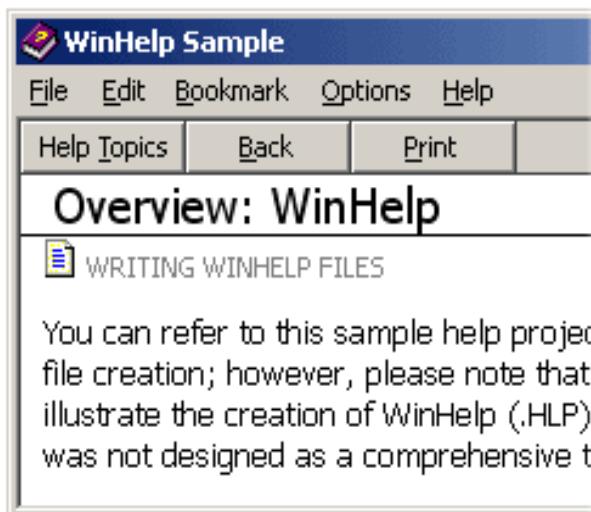


- **DocBuilder** es una aplicación capaz de generar archivos de documentación y ayuda en diversos formatos para el software desarrollado por ti mismo. El programa reconoce varios lenguajes de programación: C/C++, Pascal, Delphi. Y puede generar documentación en **RTF, HTML y archivos Windows de ayuda.**

Esta herramienta examina la estructura del código fuente del programa, distinguiendo los comandos de los comentarios sobre el software, analizando estos últimos y distribuyéndolos de acuerdo con el formato elegido. Es de las más antiguas.



- **WinHelp Compiler** ayuda a crear archivos de ayuda. Este compilador de Microsoft crea ficheros de ayuda Windows .**hlp** que sólo se pueden ver en Windows 95/98/NT4 y superiores. WinHelp Compiler. Este paquete incluye "Segmented Hypergraphics Editor" (SHED.EXE) y "Multi-Resolution Bitmap Compiler" (MRBC.EXE).



En la red podréis encontrar algunos más pero en general estos son los más utilizados.

Para poder visualizar un fichero **chm** en Linux existen diferentes herramientas. En Gnome por ejemplo, está **xchm** que es un visor de archivos propietarios **chm** de Microsoft.

→ Tablas de contenidos, índices, sistemas de procura, etc. Integración en la aplicación.

Para explicar esto utilizaremos el popular sistema de ayudas basado en archivos **chm**.

CHM utiliza un lenguaje de marcas HTML que es un lenguaje muy probado y nos ofrece mas posibilidades y versatilidad que los derivados en RTF.

Una vez creados los diferentes capítulos de nuestra ayuda en diferentes archivos HTML, estos son **ensamblados en un archivo de proyecto con extensión HHP**. A este proyecto se le añaden :

- **Archivos H** : Añaden un ID para identificador de capítulo
- **Archivos HHK** : Indices.
- **Archivos HHC** : Tablas de contenido.

Con todo esto, compilamos el proyecto y obtenemos un archivo de ayuda con extensión **chm**. La estructura básica para crear dicho proyecto de ayuda CHM está basada un archivo de extensión HHP que aglutina todos los demás archivos que conforman la ayuda CHM.

Para crear este y los demás archivos podemos utilizar una aplicación como, por ejemplo, **Help WorkShop** o un editor de textos sencillo como el bloc de notas.

Un archivo de **proyecto HHP** es un archivo de texto estructurado por secciones que se referencian con los caracteres “[“ y “]”. Las secciones mas importantes son :

- **OPTIONS** : Opciones de compilación. En esta sección damos a conocer al compilador de ayudas informaciones tales como :
 - Title : Titulo de nuestra ayuda.
 - Language : ID del idioma utilizado para nuestra ayuda.
 - Index file : Archivo de indice, (HHK)
 - Contents file : Archivo de contenido, (HHC)
 - Compiled file : Nombre del archivo de ayuda compilado, (CHM)

Un ejemplo para esta sección seria :

```
[OPTIONS]
Compatibility=1.1 or later
Compiled file=PruebaHTML.chm
Contents file=PruebaHTML.hhc
Default topic=PruebaHTML1.htm
Display compile progress=No
Index file=PruebaHTML.hhk
```

Language=0xc0a Español (España)

Title=Prueba ayudas HTML

- **ALIAS** : Identificador de Nombre del archivo HTM. Cada archivo HTM de nuestra ayuda puede identificarse por un alias, (nombre o variable), que lo identifica y relaciona con un ID que es utilizado posteriormente. Por ejemplo :

[ALIAS]

IDH_top1=PruebaHTML1.htm

IDH_Top2=PruebaHTML2.htm

- **MAP** : Archivo con la definición de los nombres de alias y un ID para cada uno que luego es utilizado en la aplicación bajo Windows. Esta relación de archivos H en el que se definen los ALIAS descritos en la sección anterior se relacionan con un ID que se utilizará posteriormente. Por ejemplo :

[MAP]

#include PruebaHTML.h

- **Archivos HHC, (Contenido):** En un archivo HHC se define la estructura del libro de contenidos de nuestra ayuda. El formato de un archivo de contenido es HTML. Un ejemplo de archivo HHC seria :

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft® HTML Help Workshop
4.1">
<!-- Sitemap 1.0 -->
</HEAD><BODY>
<OBJECT type="text/site properties">
<param name="Window Styles" value="0x800025">
<param name="Image Type" value="Folder">
</OBJECT>
<UL>
<LI> <OBJECT type="text/sitemap">
<param name="Name" value="Principal">
```

```

</OBJECT>
<UL>
<LI> <OBJECT type="text/sitemap">
    <param name="Name" value="Principal">
    <param name="Local" value="PruebaHTML1.htm">
    <param name="Comment" value="Comentario">
    <param name="ImageNumber" value="10">
</OBJECT>
<LI> <OBJECT type="text/sitemap">
    <param name="Name" value="Capítulo I">
    <param name="Local" value="PruebaHTML2.htm">
</OBJECT>
</UL>
</UL>
</BODY></HTML>

```

- **Archivos HHK, (Índice):** Un archivo de indices también está descrito en formato HTML. En él se hace referencia todas las cadenas de búsqueda para cada capítulo. Un ejemplo de archivo de indice seria :

```

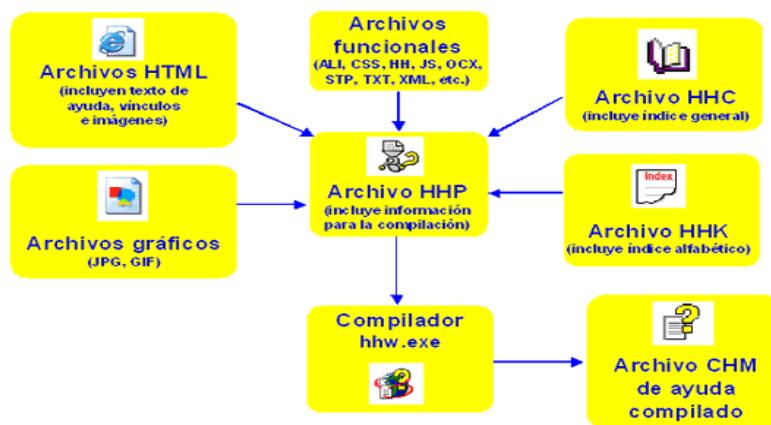
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<meta name="GENERATOR" content="Microsoft® HTML Help Workshop
4.1">
<!-- Sitemap 1.0 -->
</HEAD><BODY>
<UL>
<LI> <OBJECT type="text/sitemap">
    <param name="Name" value="Capítulo I">
    <param name="Name" value="Capítulo I">
    <param name="Local" value="PruebaHTML2.htm">
    <param name="URL" value="PruebaHTML2.htm">
</OBJECT>
<LI> <OBJECT type="text/sitemap">

```

```

<param name="Name" value="Principal">
<param name="Name" value="Capítulo I">
<param name="Local" value="PruebaHTML2.htm">
<param name="Name" value="Principal">
<param name="Local" value="PruebaHTML1.htm">
</OBJECT>
</UL>
</BODY></HTML>

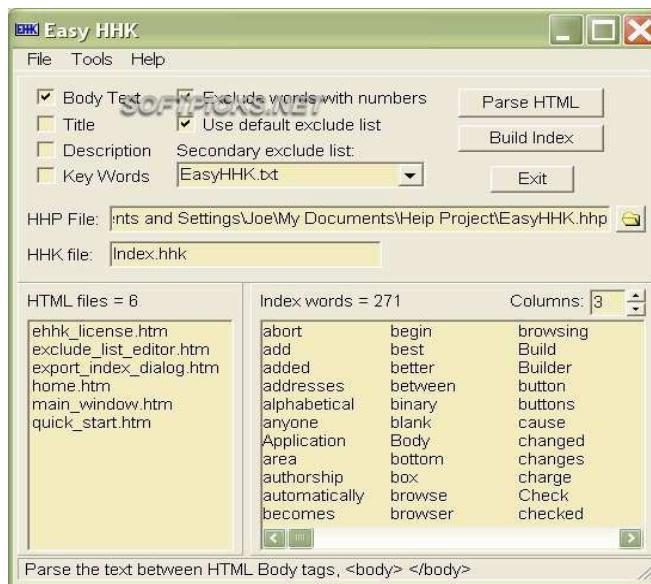
```



- Archivos H, (Declaración de ID de contexto):** En un archivo H definimos, (con el comando `#define`), los alias que referenciaremos a cada capítulo, (archivos HTM), de nuestra ayuda. Un ejemplo de definición de estos alias seria :

```
#define IDH_Top1 1
```

```
#define IDH_Top2 2
```



- **Archivos HTM, (Cuerpos de la ayuda):** Los archivos HTM son archivos en formato HTML en los que **relatamos toda nuestra ayuda**. Cada capítulo o tema de nuestra ayuda conforma un archivo HTM.

➔ **Tipos de manuales:** **manual de usuario, guía de referencia, guías rápidas, e manuales de instalación, configuración e administración. Destinatarios e estructura.**

La documentación de los programas es un aspecto sumamente importante, tanto en el desarrollo de la aplicación como en el mantenimiento de la misma. Mucha gente no da la suficiente importancia a este parte del desarrollo con lo que se pierde, muchas veces, la posibilidad de la reutilización de código en otras aplicaciones o la dificultad posterior de modificaciones del mismo. La documentación de un programa empieza a la vez que la construcción del mismo y finaliza justo antes de la entrega del programa o aplicación al cliente. Así mismo, la documentación que se entrega al cliente tendrá que coincidir con la versión final de los programas que componen la aplicación.

Una vez concluido el programa, los documentos que se deben entregar son una **guía técnica**, una **guía de uso** y **guía de instalación**.

- **Tipos de documentación**

La documentación que se entrega al cliente se divide claramente en dos categorías, interna y externa:

- **Interna:** Es aquella que se crea **en el mismo código**, ya puede ser en forma de comentarios o de archivos de información dentro de la aplicación.
- **Externa:** Es aquella que se escribe en **cuadernos o libros**, totalmente ajena a la aplicación en si. Dentro de esta categoría también se encuentra la **ayuda electrónica**.

- **La guía técnica**

En la guía técnica o manual técnico se reflejan el **diseño del proyecto, la codificación de la aplicación y las pruebas realizadas** para su correcto funcionamiento. Generalmente este documento esta diseñado para personas con conocimientos de informática, generalmente programadores. El **principal objetivo** es el de facilitar el desarrollo, corrección y futuro mantenimiento de la aplicación de una forma rápida y fácil.

Esta guía esta compuesta por tres apartados claramente diferenciados:

- **Cuaderno de carga:** Es donde queda reflejada la **solución o diseño de la aplicación**. Esta parte de la guía es únicamente destinada a los programadores. Debe

- estar realizado de tal forma que permita la división del trabajo
- **Programa fuente:** Es donde se **incluye la codificación** realizada por los programadores. Este documento puede tener, a su vez, otra documentación para su mejor comprensión y puede ser de gran ayuda para el mantenimiento o desarrollo mejorado de la aplicación. Este documento debe tener una gran claridad en su escritura para su fácil comprensión.
 - **Pruebas:** es el documento donde se especifican el tipo de pruebas realizadas a lo largo de todo el proyecto y los resultados obtenidos.
- **La guía de usuario**

Es lo que comúnmente llamamos el **manual del usuario**. Contiene la información necesaria para que los usuarios utilicen correctamente la aplicación.

Este documento se hace desde la guía técnica pero se **suprime los tecnicismos** y se presenta de forma que sea comprensible **para el usuario que no sea experto en informática**.

Un punto a tener en cuenta en su creación es que no debe hacer referencia a ningún apartado de la guía técnica y en el caso de que se haga uso de algún tecnicismo debe ir acompañado de un **glosario al final** de la misma para su fácil comprensión.

Las secciones de un manual de usuario a menudo incluyen:

- Una página de portada.
- Una página de título.
- Una página de derechos de autor.
- Un prefacio, que contiene detalles de los documentos relacionados y la información sobre cómo navegar por la guía del usuario.
- Una página de contenido.
- Una guía sobre cómo utilizar al menos las principales funciones del sistema, es decir, sus **funciones básicas**.
- Una sección de solución de problemas que detalla los posibles errores o problemas que pueden surgir, junto con la forma de solucionarlos.
- Una sección de **preguntas frecuentes. (FAQ)**
- Dónde encontrar más ayuda, y datos de contacto.
- Un Glosario y, para documentos más grandes, un índice.

- **La guía de instalación**

Es la guía que contiene la información necesaria para implementar dicha aplicación. Dentro de este documento **se encuentran las instrucciones para la puesta en marcha del sistema** y las normas de utilización del mismo. Dentro de las normas de utilización se incluyen también las normas de seguridad, tanto las físicas como las referentes al acceso a la información.

ACTIVIDADES

Propuesta 1. Tomando como base el *ejercicio 1* de la unidad 4 elabora una ayuda para dicha aplicación.

- A. Con la aplicación Windows Helper Maker
- B. De forma manual en el formulario Glade creado en su momento.

UD7. Distribución de las aplicaciones

RA7. Preparar aplicaciones para su distribución, para lo que utiliza y evalúan herramientas específicas.

- CA2.1. Se empaquetan los componentes que requiere la aplicación.
- CA2.2. Se personaliza el asistente de la aplicación.
- CA2.3. Se empaqueta la aplicación para su instalación típica, completa o personalizada.
- CA2.4. Se generan los paquetes de la instalación usando un entorno de desarrollo.
- CA2.5. Se generan paquetes de instalación usando herramientas externas.
- CA2.6. Se generan paquetes de instalación de forma desatendida.
- CA2.7. Se prepara el paquete de instalación para que la aplicación se pueda instalar de forma correcta.
- CA2.8. Se prepara la aplicación para descargarla desde un servidor web y ejecutarla.

BC7. Distribución de aplicaciones

- ➔ Componentes de una aplicación. Empaquetado.
- ➔ Instaladores. Paquetes autoinstalables. Herramientas para crear paquetes de instalación.
- ➔ Parámetros de la instalación. Personalización de la instalación: logotipos, fondos, diálogos, botones, idioma, etc.
- ➔ Asistentes de instalación e desinstalación.
- ➔ Tecnologías para la automatización de la descarga y ejecución de aplicaciones desde servidores web de aplicaciones.

→ Componentes de una aplicación. Empaquetado

El **empaquetado de aplicaciones** consiste en proporcionar a los futuros usuarios las aplicaciones en forma de paquetes, a los que se suele llamar en inglés **software bundle o application bundle**. Estos paquetes están formados por:

- los **programas ejecutables** de la aplicación,
- las **bibliotecas necesarias** de las que depende
- otros tipo de **ficheros** (como imágenes, bases de datos, ficheros de audio, traducciones y localizaciones, etc.).

Todos en ellos dentro del mismo fichero forman un todo o un conjunto.

Las bibliotecas de las que depende el programa pueden haber sido enlazadas, como ya hemos visto en unidades anteriores, tanto de **forma dinámica** como **también estática**. Independientemente de ello, el usuario percibe que el paquete como un conjunto que representa al programa en sí, cuando en realidad incluye varios ficheros.

Una de las mayores ventajas de un correcto empaquetado de aplicaciones es que permite **evitar los problemas de las dependencias** tanto a la hora de instalar la aplicación como a la hora de usarla, ya que cada paquete lleva consigo sus dependencias, y la instalación o desinstalación de otro software no va a afectar a las dependencias de dicho paquete. De ahí radica su principal **objetivo evitar la problemática de las dependencias**, y que la aplicación se puede trasladar de un computador a otro sin necesidad de reinstalarla, ya que el paquete de la aplicación contiene todos los ficheros necesarios para ejecutarla.

Sin embargo, como **desventaja** se presenta que estos paquetes **ocupan mucho más espacio** en el disco, especialmente si el paquete incluye bibliotecas.

Por lo general cada distribución tiene su propia forma de empaquetar sus aplicaciones:

→ Linux: tenemos dos tipos de paquetes que sobresalen del resto:

- **rpm**: (Redhat Package Manager) de la familia de RedHat (RHEL, Fedora,CentOS), Mandriva, Suse
- **deb**: de la familia de Debian (Debian,Ubuntu, y derivados)

→ Windows: tomando como base su IDE Visual Studio el formato de empaquetado es:

- **msi**: se definen como **instaladores de Microsoft**, a saber, aquellos paquetes de software que contienen la información necesaria para automatizar su instalación, minimizando la intervención manual del usuario, ya que toda la información iría contenida en el propio fichero **msi**. La información de instalación, y a menudo los archivos mismos, son empaquetados en **paquetes de instalación**, bases de datos

estructuradas como **OLE Structuree Storage** (almacenamiento estructurado de ficheros) y comúnmente conocido como "MSI files" por su extensión de archivo. El sistema de archivos **msi** es muy útil cuando queremos distribuir aplicaciones en equipos pertenecientes a un dominio **Windows**.

➔ **Java:** el principal es jar.

- **jar:** es un tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java. Las siglas están deliberadamente escogidas para que coincidan con la palabra inglesa "jar" (tarro). Los archivos JAR están comprimidos con el formato **zip** y cambiada su extensión a .jar. Es el más popular entre los dispositivos móviles y en los SO más importantes.

➔ **Instaladores. Paquetes autoinstalables. Herramientas para crear paquetes de instalación.**

WINDOWS

En equipos **WINDOS** existen los ficheros que comúnmente son llamados ejecutables. Los primeros fueron los llamados **bat** (extensión de un fichero formado por un lote de órdenes Dos, **que ya no se usan**), los **exe** y los **msi**, éstos dos últimos son utilizados con mayor frecuencia en el entorno gráfico de Windows.

La herramienta encargada de llevar a cabo la instalación es **Windows Installer** que es un motor para la instalación, mantenimiento y eliminación de programas en plataformas **Windows**.

Los paquetes **MSI (Microsoft Installer)** se definen como **instaladores de Microsoft** o paquetes de software que contienen la información necesaria para automatizar su instalación, minimizando la intervención manual del usuario, ya que toda la información iría contenida en el propio fichero "msi".

Una de las ventajas de los paquetes msi es su facilidad para **la distribución de software** desde servidores **Windows Server**. Estos paquetes se pueden crear incluso con *software de terceros* para luego instalarlos en equipos en red pertenecientes a un dominio.

Un paquete describe la instalación completa de un producto ya que Windows Installer no maneja dependencias entre productos y está identificado por un **GUID**. El paquete está compuesto de **componentes** agrupados dentro de sus **características**.

Un **componente** es la mínima parte de un producto. Cada componente es tratado por Windows Installer como una unidad. Los componentes pueden contener *archivos, grupos de archivos,*

directorios, componentes COM, claves del registro de Windows, accesos directos y otro tipo de datos. El usuario final no interviene directamente con los componentes.

También los componentes al estar identificados globalmente por GUID's, ello permite que un mismo componente sea compartido entre varios del mismo paquete o de múltiples paquetes, idealmente a través del uso de la unión de módulos (aunque para trabajar correctamente, diferentes componentes no deberían compartir ningún sub-componente).

Una **ruta maestra** es un fichero específico, clave de registro, o fuente de datos ODBC que el autor del paquete especifica como crítico para un componente dado. Como las rutas maestras más utilizadas son en forma de fichero, se suele utilizar el término *fichero maestro (key file)*. Un componente puede contener, a lo sumo, una ruta maestra; si un componente no tiene establecida de manera explícita una ruta maestra, el directorio destino del componente es tomado como la ruta maestra. Cuando se ejecuta una aplicación basada en MSI, Windows Installer comprueba la existencia de estos **fichero críticos o claves de registro** (es decir, las rutas maestras). Si existe un desajuste entre el estado actual del sistema y el valor especificado en el paquete MSI (e.g., un fichero maestro desaparecido), entonces la característica asociada **es reinstalada**. Este proceso es también conocido como *auto-reparación*. Dos componentes no pueden utilizar la misma ruta maestra u otra.

InstallShield es una herramienta de software para crear instaladores. InstallShield se utiliza sobre todo para instalar software del escritorio y las plataformas de servidor de Windows, pero también se puede usar para administrar aplicaciones y paquetes de software en una amplia gama de móviles y portátiles. Fue desplazado por **Windows Installer** aunque aún se utiliza en la actualidad.



Por otro lado para la creación del instalador **msi de Windows**, Visual Studio tiene su propia herramienta **VSI (Visual Studio Installer)**. Con el Visual Studio Installer, se puede desarrollar un proyecto de instalación creando un archivo **.wip** como una parte integral de su solución. El archivo de paquete de instalación de Windows (.msi), se construye a partir del proyecto de Visual Studio Installer (.wip) que contiene todos los datos e instrucciones necesarias para instalar la aplicación.

En la red hay diferentes manuales que explican su uso, cosa no complicada ya que se trata de un asistente.

LINUX

Hay 3 formas de instalar paquetes en GNU/Linux:

- **Compilar el paquete:** Esta es la forma clásica, y antigua, de instalar paquetes. Consiste en bajar el código fuente, comprimido en un archivo **.tar.gz o .tar.bz2**.

Una vez bajado, entramos en la consola (shell) y nos movemos hasta el directorio donde tengamos el paquete. Si el paquete está en formato .tar.gz escribimos:

```
# tar -xzvf archivo.tar.gz      //para descomprimir el paquete
```

Si está en .tar.bz2 escribimos:

```
# bzip2 -dc archivo.tar.bz2 | tar -xv
```

Una vez hecho esto, hay que entrar en el directorio creado y compilar el código para obtener la aplicación funcional y que consiste en escribir en la siguiente línea de comandos lo siguiente:

```
# ./configure  
# make  
# make install
```

Lo que estamos haciendo es compilando el programa a partir de código fuente.

Uno de los principales problemas de este método es si el paquete tiene dependencias, es decir, si depende de algún otro paquete para que funcione correctamente. En ese caso, habrá que instalarlos manualmente.

- **Paquetes .deb y .rpm:** Los paquetes .deb y .rpm son un método de instalación muy efectivos para sus respectivas distribuciones.

Los paquetes **.deb** son paquetes que se pueden instalar en la distribución Debian y derivados (Ubuntu, Kubuntu...).

Los **.rpm** (RedHat Package Manager) son los de la distribución Red Hat y derivados (OpenSuse, Mandriva, Fedora...).

Un paquete **.rpm** no lo podemos instalar en la distro Debian o derivados, y un **.deb** tampoco en RedHat y derivados. No obstante, existen programas como por ejemplo el denominado '**Alien**' que permite convertir un paquete **.rpm** a **.deb** y viceversa.

Para instalar un paquete **.deb** entramos la siguiente línea de comandos en la consola (Situándonos en el directorio donde está el paquete):

```
# dpkg -i nombredelpaquete.deb
```

Para **crear un paquete deb** ya hay asistentes que facilitan bastante su creación. Uno de ellos es **checkinstall**. Los pasos son los siguientes:

Para empezar, tenemos que **instalar checkinstall**. Así que hacemos (*como root*):

```
# apt-get install checkinstall
```

Lo siguiente es ir a la **carpeta** en la que tenemos el **código de la aplicación**, y abrir una **terminal**. Ejecutamos los siguientes comandos, uno a uno:

```
# ./configure
```

```
# make
```

Con "*./configure*" se configuran los paquetes para nuestra distribución y se crea un "*Makefile*" (*un archivo que contiene instrucciones de compilación*), y con "*make*" se compila el código y deja los binarios, librerías, etc en la carpeta "*src*". Ahora, antes de continuar, es recomendable **no tener instalada** la aplicación de la que se hará el paquete. Si lo está:

```
# make uninstall
```

Y entonces es momento de comenzar con la parte importante, el uso de **checkinstall**. En esa misma terminal, escribimos:

```
# checkinstall
```

Y se abrirá el “asistente” de *checkinstall*. En él podemos modificar **la información** que tendrá el paquete que crearemos. Las opciones que podemos modificar son:

- **Maintainer**: el desarrollador principal del paquete.
- **Summary**: una descripción del paquete.
- **Name**: nombre que quieras darle al paquete.
- **Versión**: versión del paquete.
- **Release**: viene siendo la versión principal del paquete, podemos dejarlo como venga.
- **License**: licencia de la aplicación, es preferible no tocarlo.
- **Group**: grupo por el cuál fué creado, podemos dejarlo como está.
- **Architecture**: arquitectura de procesador del paquete.
- **Source location**: nombre de la carpeta (solo la carpeta, no la ruta entera) en la que está el código del paquete.
- **Alternate source location**: no es necesario modificarlo.
- **Requires**: dependencias que deben ser instaladas para su correcto funcionamiento.
- **Provides**: nombre del paquete que provee, no es necesario modificarlo.
- **Conflicts**: paquetes con los que entra en conflicto.
- **Replaces**: paquetes a los que reemplaza.



Cada una tiene **un número** a su izquierda, así que para editarla solo **escibimos su número**. No es necesario modificar todas, pero si las más importantes como *Maintainer*, *Summary*, *Name* y *Version*.

Una vez hayamos modificado lo que queremos, presionamos **[Enter]** (*sin ningún número previo*) y comenzará a **compilar e instalar** el paquete. Cuando haya terminado, en el directorio donde compilamos habrá aparecido un **paquete .deb** de la aplicación, listo para instalar. Asegurarse en "*Version*", **no** haya letras. Eso suele evitar que se cree el paquete. Es posible que "*Requires*" dé un fallo, con dejar el espacio **en blanco**.

En RedHat paraa instalar un paquete **-rpm** introducimos:

```
# rpm --install nombredelpaquete.rpm
```

Para crear un paquete rpm:

```
# yum groupinstall "Development Tools"
# yum install rpmdevtools
# yum install rpmlint
# rpmdev-setuptree
```

Ésto nos creará un entorno de trabajo para crear los RPM's o "Árbol del Proyecto". Necesitaremos dos cosas: un Archivo **".spec"** y el **código fuente del programa** para instalar. El archivo ".spec" lo puedes conseguir de cualquier "src.rpm" creado para tu versión de Fedora (con una versión anterior del código fuente por ejemplo) o con un "src.rpm" para una versión antigua de fedora, una fuente para dicho tipo de paquetes está en: <http://www.rpmfind.net/>

Debemos abrir el paquete "src.rpm" que hayas descargado con un gestor de archivadores, ahí dentro encontrarás el ".spec". Debido a que lo que haremos será construir un Nuevo RPM desde código fuente más reciente, generalmente sólo debemos cambiar el campo "Version" y agregar un "Changelog" y fijarse en el campo "Release". Por ejemplo usar el paquete de **Kmess 2.0.5-2** de **Fedora 14** para crear el **2.0.6.1-0** usable en el mismo Fedora, en ése caso, sólo modificarímos el apartado *Version* por el nuevo, quitando el **2** de *Release* y poniendo un **0** ("Release" es el número que aparecerá después del guión) y agregando un *changelog* copiando uno de los ejemplos anteriores y modificándolo según aplicó con el *changelog* oficial.

Otra cosa que es chequear el archivo ".spec" en su apartado de #BuildRequires ya que todo lo que esté en éste apartado son dependencias sin las cuales el paquete no podrá ser creado. Instalamos todos los paquetes requeridos con un:

```
# yum -y install paquete1 paquete2 paquete3
```

Vamos a la carpeta "rpmbuild" y luego en la carpeta de SOURCES debemos colocar el código fuente de nuestra aplicación, el "tarball" para explicarme mejor. Por otro lado, debemos copiar nuestro ".spec" modificado/recentemente creado en SPECS y continuamos con el siguiente paso.

Usamos una pequeña herramienta llamada *rpmlint* para nuestro cometido. Ésta herramienta nos avisará de errores en la estructura del archivo que acabamos de modificar/escribir y su uso es bastante sencillo:

```
# rpmlint program.spec
```

Se nos mostrará un Output donde nos dará información acerca de nuestro archivo. Si hay errores o advertencias, simplemente tenemos que hacer caso y modificar según se necesite.

Ya tenemos todo listo para la construcción:

```
# rpmbuild -ba program.spec
```

Tras un tiempo de compilado y trabajo, tendremos un paquete RPM listo para usar, su "src.rpm" y también los paquetes adicionales que se hayan creado.

Terminada la creación del paquete, si todo salió bien, podremos irnos al directorio *rpmbuild*

y en el subdirectorio RPMS encontraremos los binarios de nuestra aplicación y los extras que la acompañen, mientras que en el directorio SRPMS encontraremos los paquetes "src.rpm" para la misma aplicación. Tener a la mano éstos dos tipos de paquetes es lo que debería importarte... Ahora simplemente se chequea el RPM final generado (el que usaremos para la instalación de la aplicación) con rpmlint:

```
# rpmlint *.rpm
```

El último paso tanto en **deb** como **rpm** es subirlos a una web de repositorios para que puedan ser descargados con **aptitude** o **apt-get** en el primer caso o con **yum** en el segundo caso.

➔ Asistentes de instalación y desinstalación

WINDOWS

Una de las herramientas “ocultas” de Windows es **iexpress**.



IExpress es una utilidad de Microsoft incluido con varias ediciones de los sistemas operativos Windows (32-bits y 64-bits), Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7 y Windows 8. También fue incluido como parte de todos los de Internet Explorer Administration Kit Lanzamientos 4, 5 y 6, y era parte de toda la instalación de Internet Explorer 6. Iexpress.exe se utiliza para crear auto-paquetes de un conjunto de archivos. Estos paquetes pueden utilizarse para instalar aplicaciones, el ejecutable, los paths, otros componentes del sistema, o bootstrappers de configuración.

De esta forma se puede utilizar para la distribución de paquetes de instalación a múltiples ordenadores con Windows locales o remotos. Crea **ejecutable autoextraíble (.exe)** o un **archivo contenedor comprimido (.CAB)** usando la interfaz proporcionada que facilita la automatización.

Los archivos pueden modificarse con cualquier texto plano / editor ASCII, como el Bloc de notas.

Todos los archivos autoextraíbles creados por el uso de Iexpress se comprimen con el **MakeCab** (MAKECAB.EXE) y se extraen mediante el Wextract (wextract.exe) de Microsoft.

Iexpress.exe se encuentra en la carpeta System32 de Windows. La interfaz frontal (IExpress Wizard) se puede iniciar manualmente escribiendo IExpress en la ventana Ejecutar del menú Inicio. También se puede utilizar desde la línea de comandos (o archivo por lotes en la consola) para crear paquetes de instalación personalizada, con el tiempo desatendida (funcionamiento automático):

c:> IEXPRESS /N letraDeUnidad directory_name file_name.SED

A través de la creación de interfaz del Asistente de IExpress le permite al usuario especificar un **título para los paquetes, añadir la solicitud de la licencia de usuario final** que debe ser aceptado como una orden para permitir la extracción, seleccione los archivos que se archivan, la visualización avanzada ventana de opciones, y finalmente, especificar un mensaje que se mostrará al finalizar.

La **desinstalación de software** es el proceso de revertir los cambios producidos en un sistema por la instalación del mismo. Por ello no solo deben ser borrados los archivos, sino también cambios en otras aspectos del software, como por ejemplo, eliminar usuarios que hayan sido creados, retirar derechos concedidos, borrar directorios creados hasta llevar la contabilidad en un sistema de gestión del sistema, en el caso de Windows el Registro.

Debido a la creciente complejidad de sistemas operativos y sus interfaces (API), la desinstalación de software puede ser no solo contraproducente sino también poner en peligro la estabilidad del sistema. Por esta razón **la calidad de un software** no solo depende de sus efectos productivos o creativos sino también de su **capacidad de integración en el sistema operativo y compatibilidad con otros programas**. El desarrollador del software debe ofrecer una función para desinstalar su software sin dañar o desestabilizar el sistema.

Cada vez es más difícil la desinstalación, dado que muchas bibliotecas se comparten entre aplicaciones de diferentes productores de software que utilizan enlaces duros o simbólicos a través del directorio.

En sistemas de alta complejidad, el esfuerzo para desinstalar un programa puede ser mayor que el de la instalación.

En Windows la desinstalación mejora si utilizamos software que además de eliminar los archivos del software eliminan las entradas en los registros. Esto es debido a que el desinstalador que lleva Windows (*Agregar y Quitar Programas*) no es muy fiable en ese aspecto. Existen varias herramientas como por ejemplo es **Ccleaner**, **RegCleaner**, **Revo Uninstaller** o **Iobit Uninstaller** entre otras, que además de desinstalar hacen búsquedas en la base de datos del Registro para **eliminar aquellas entradas** que no están asociadas a ningún software instalado.

En cuanto a Linux los comandos para desinstalar aplicaciones son los siguientes:

```
# apt-get remove "nombre-del-paquete"      //desinstalamos el paquete
# apt-get purge "nombre-del-paquete"        //borra los archivos de configuración
# apt-get clean "nombre-del-paquete" // borra los archivos descargados con la aplicación
```

→ **Tecnologías para la automatización de la descarga y ejecución de aplicaciones desde servidores web.**

Un **sistema de gestión de paquetes**, también conocido como gestor de paquetes, es una colección de herramientas que sirven para automatizar el proceso de **descarga, instalación, actualización, configuración y eliminación de paquetes de software**. El término se usa comúnmente para referirse a los gestores de paquetes en sistemas UNIX, especialmente GNU/Linux, ya que se apoyan considerablemente en estos sistemas de gestión de paquetes.

En estos sistemas, el software se distribuye en forma de paquetes, frecuentemente encapsulado en un solo fichero. Estos paquetes incluyen otra información importante, además del software mismo, como pueden ser el nombre completo, una descripción de su funcionalidad, el número de versión, el distribuidor del software, la suma de verificación y una lista de otros paquetes requeridos para el correcto funcionamiento del software. Esta metainformación se introduce normalmente en una base de datos de paquetes local.

| Sistema de Gestión de Paquetes | Instalador |
|--|---|
| Forma parte del sistema operativo. | Cada producto viene unido a su propio instalador. |
| Usa una única base de datos de instalación. | Rastrea su propia instalación |
| Puede verificar y administrar todos los paquetes sobre el sistema. | Sólo trabaja con su propio producto. |
| Un único vendedor de sistema de administración de paquetes. | Múltiples vendedores de instalador. |
| Un único formato de paquetes. | Múltiples formatos de instalación |

Los sistemas de gestión de paquetes tienen la tarea de **organizar todos los paquetes instalados** en el sistema y se encargan de mantener su usabilidad. Esto se consigue combinando las siguientes técnicas:

- Comprobación de la suma de verificación para evitar que haya diferencias entre la versión local de un paquete y la versión oficial.
- Comprobación de la firma digital.
- Instalación, actualización y eliminación simple de paquetes.
- Resolución de dependencias para garantizar que el software funcione correctamente.
- Búsqueda de actualizaciones para proveer la última versión de un paquete, ya que normalmente solucionan *bugs* y proporcionan actualizaciones de seguridad.
- Agrupamiento de paquetes según su función para evitar la confusión al instalarlos o mantenerlos.

Muchos de los sistemas de gestión de paquetes ampliamente utilizados utilizan backends simples para instalar los paquetes. Por ejemplo, YUM utiliza RPM como backend y APT utiliza dpkg.

En los sistemas donde las aplicaciones comparten módulos, como en la mayor parte de las distribuciones de GNU/Linux, **la resolución de dependencias al instalar y desinstalar software se convierte en una necesidad**. Algunos de los sistemas de gestión de paquetes más avanzados tienen la capacidad de desinstalar los paquetes recursivamente o en cascada, de forma que se eliminan todos los paquetes que dependen del paquete a desinstalar y todos los paquetes de los que el paquete a desinstalar depende, respectivamente.

Es común que un administrador **instale software que no está disponible en los repositorios provistos**. Algunos ejemplos pueden ser una nueva versión de una aplicación que todavía no está disponible en la distribución o una alternativa distinta de la elegida por la distribución. Si este software adicional sólo se distribuye en forma de código fuente, la instalación requerirá **la compilación del código**. Sin embargo, la instalación de este software adicional en el sistema **ocasionará que el estado del sistema y la base de datos del gestor de paquetes no estén sincronizados**, por lo que el administrador deberá tomar medidas adicionales para asegurar que el sistema de gestión de paquetes se mantenga al día, puesto que éste no es capaz de hacerlo automáticamente.

Alien es un programa que convierte entre los diferentes formatos de paquetes de GNU/Linux. Soporta la conversión entre Linux Standard Base, RPM, deb, Stampede (.slp) y paquetes de Slackware (.tgz).

Otra problemática aparte de la **actualización de software es la actualización de ficheros de configuración**. Ya que los sistemas de gestión de paquetes sólo son capaces de sobrescribir o retener los ficheros de configuración, en lugar de poder aplicarles reglas de modificación. Sin embargo, hay excepciones, que normalmente se aplica al proceso de configuración del núcleo, ya que si estos son incorrectos pueden ocasionar fallos al reiniciar el sistema, pudiendo incluso hacer que el sistema no arranque. Estos problemas pueden ocasionarse cuando el formato de los **ficheros de configuración cambia**. Por ejemplo, cuando el antiguo fichero de configuración no deshabilita nuevas opciones que deberían ser deshabilitadas. Algunos sistemas de gestión de paquetes, como el dpkg de Debian, permiten configurar el software durante la instalación. En cualquier otra situación es preferible instalar los paquetes con la configuración por defecto y sobrescribirla posteriormente.

El software normalmente **se pone a disposición de los usuarios en los repositorios**, con el fin de proporcionar a los usuarios de un sencillo control sobre los diferentes tipos de software que van a instalar en su sistema y, en ocasiones, debido a razones legales o conveniencias por parte de los distribuidores.

```
GNU nano 2.2.2           File: /etc/apt/sources.list

# deb cdrom:[Ubuntu 10.04 LTS _Lucid Lynx_ - Release amd64 (20100429)]/ l$#
# See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
# newer versions of the distribution.

deb http://gb.archive.ubuntu.com/ubuntu/ lucid main

## Major bug fix updates produced after the final release of the
## distribution.
deb http://gb.archive.ubuntu.com/ubuntu/ lucid-updates main

## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubun$#
## team. Also, please note that software in universe WILL NOT receive any
## review or updates from the Ubuntu security team.
deb http://gb.archive.ubuntu.com/ubuntu/ lucid universe multiverse
deb http://gb.archive.ubuntu.com/ubuntu/ lucid-updates universe multiverse

^G Get Help ^O WriteOut ^R Read File^Y Prev Page^K Cut Text ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

Cuando un usuario interactúa con el gestor de paquetes para realizar una actualización, éste suele mostrar una lista de las tareas a realizar (normalmente la lista de paquetes a actualizar y, posiblemente, también los números de versión) y también es probable que permita realizar una actualización completa o bien seleccionar los paquetes que se desea actualizar. Algunos gestores de paquetes permiten indicar los paquetes que no se desea actualizar nunca o solamente cuando **estos corren errores importantes en la versión anterior**. A este proceso se lo suele denominar **version pinning**. Por ejemplo, yum permite esto mediante la sintaxis

exclude=openoffice*, pacman con IgnorePkg=openoffice (en ambos casos para evitar la actualización de OpenOffice), mientras que las herramientas de Debian poseen una sintaxis más compleja y potente.

Sistemas basados en paquetes binarios

- **dpkg**, usado originalmente por Debian y ahora también por otros sistemas, usa el formato **.deb** y fue el primero en poseer una herramienta de resolución de dependencias ampliamente conocida, **APT**.

Instalar un paquete:

```
# apt-get install <paquete>
```

Desinstalar un paquete:

```
# apt-get remove <paquete>
```

Eliminar un paquete incluidos sus ficheros de configuración:

```
# apt-get purge <paquete>
```

Eliminar de forma automática aquellos paquetes que no se estén utilizando:

```
# apt-get autoremove
```

Actualizar un paquete a la última versión disponible en el repositorio:

```
# apt-get update <paquete>
```

Actualizar el sistema. Actualizará todos los paquetes que dispongan de una versión superior dentro de la rama instalada de la distribución:

```
# apt-get upgrade
```

Actualizar la distribución completa. Actualizará nuestro sistema a la siguiente versión disponible de la distribución:

```
# apt-get dist-upgrade
```

Descargar únicamente las fuentes de un paquete para manipularlas de forma manual:

```
# apt-get source <paquete>
```

Limpiar cachés, paquetes descargados, etc:

```
# apt-get clean
```

```
# apt-get autoclean
```

Verificar que no tenemos ninguna dependencia incumplida:

```
# apt-get check
```

Buscar un paquete en los repositorios, se puede especificar un patrón, expresión regular, el nombre exacto del paquete, etc:

```
# apt-cache search <paquete>
```

Mostrar información sobre un paquete específico (nombre del paquete, versión, dependencias...):

```
# apt-cache showpkg <paquete>
```

Mostrar información del paquete incluyendo la descripción, información del paquete como su sitio web, página de bugs...

```
# apt-cache show <paquete>
```

Mostrar dependencias de un paquete:

```
# apt-cache depends <paquete>
```

Mostrar los nombres de todos los paquetes instalados en el sistema:

```
# apt-cache pkgnames
```

- **fink**, para Mac OS X, deriva parcialmente de dpkg/apt y de ports. Esta herramienta pretende hacer más sencilla la instalación de programas libres en Mac OS X
- **el sistema RPM**, creado por Red Hat y usado por un gran número de distribuciones de GNU/Linux, es el formato de paquetes del Linux Standard Base. Para trabajar con este sistema de paquetes existen muy diversas herramientas como apt4rpm, up2date (de Red Hat), urpmi (de Mandriva), YaST (de SuSE) y YUM (usado por Fedora Yellow Dog Linux).

`yum -y install paquete`

Instala la última versión del paquete indicado. Instala sin pedir confirmación.

`yum -y install paquete1 paquete2`

Instala la última versión de los paquetes indicados, no hay límite de cuantos paquetes se pueden indicar. Instala sin pedir confirmación.

`yum -y install paquete.arch`

Instala la última versión del paquete indicado con la arquitectura indicada, por ejemplo: **yum install mysql.i386**.

`yum -y update`

Actualiza todos los paquetes en el sistema.

`yum -y update --exclude=sendmail`

Actualiza todos los paquetes del sistema, excepto sendmail.

`yum -y update httpd`

Actualiza solo el paquete indicado, en este caso el servidor Web Apache.

`yum -y update opera firefox`

Actualiza los paquetes indicados.

`yum -y update --enablerepo=centosplus`

Además de los repositorios que se tengan se habilita otro, en este caso 'centosplus', esta opción también aplica para 'install'.

`yum -y upgrade`

Actualiza los paquetes indicados, pero tomando en cuenta paquetes obsoletos en el cálculo de la actualización. Esta opción es idéntica a **yum -y --obsoletes update** y solo es realmente útil cuando se actualizan paquetes a través de distintas versiones de la distribución, por ejemplo de centos4 a centos5.

`yum check-update`

Muestra una lista de paquetes que necesitan ser actualizados sin instalarlos.

`yum info paquete`

Descripción completa del paquete indicado. Ejemplo:**yum info samba**

| | |
|---|---|
| <code>yum info recent</code> | Muestra información resumida de los últimos paquetes instalados o actualizados. |
| <code>yum info available</code> | Muestra información resumida de los paquetes disponibles a actualizarse. |
| <code>yum list</code> | Lista de todos los paquetes disponibles para instalación, actualización o ya instalados. |
| <code>yum list grep mysql</code> | Muestra solo los paquetes disponibles o ya instalados de mysql. |
| <code>yum list installed</code> | Lista de todos los paquetes instalados en el sistema. |
| <code>yum list available</code> | Lista de todos los paquetes disponibles para ser instalados. |
| <code>yum list updates</code> | Lista de todos los paquetes disponibles para ser actualizados. |
| <code>yum remove telnet</code> | Remueve el paquete indicado. |
| <code>yum -y remove telnet vncserver</code> | Remueve los paquetes indicados sin pedir confirmación. |
| <code>yum search paquete</code> | Busca el 'paquete' en la base de datos de paquetes instalados o para instalar. 'paquete' puede ser una palabra parcial del paquete a buscar. |
| <code>yum clean headers</code> | Elimina todos los archivos de encabezados que yum utiliza para resolver dependencias. |
| <code>yum clean packages</code> | Cuando utilizas la opción 'update' o 'install' el paquete que se desacarga e instala o actualiza no se elimina del sistema, ocupando espacio, con esta opción eliminas esos paquetes. |
| <code>yum clean all</code> | Limpia tanto archivos de encabezados como paquetes, como utilizar las dos opciones previas, pero al mismo tiempo. |
| <code>yum repolist</code> | Lista los repositorios que se tengan de yum. |

- El sistema **tgz**, usado por Slackware, empaqueta el software usando tar y gzip. Pero, además, hay algunas herramientas de más alto nivel para tratar con este formato: slapt-get, slackpkg and swaret.
- **Pacman**, para Arch Linux usa binarios precompilados distribuidos en un fichero pkg.tar.xz.

Sistemas de metapaquetes

Los siguientes sistemas unifican la gestión de paquetes para muchas o todas las distribuciones de GNU/Linux y otras variantes de Unix basándose también en el concepto de ficheros-receta:

- **klik** proporciona una forma sencilla de instalar paquetes de software para la mayor parte de distribuciones sin los problemas de dependencias tan comunes en otros formatos de paquetes.

- **Autopackage** usa fichero .package.
- **epm**, desarrollado por Easy Software Products (creadores de CUPS), es un metaempaquetador que permite crear paquetes nativos para todas las distribuciones de GNU/Linux y otros sistemas operativos basados en Unix (.deb, .rpm, .tgz para GNU/Linux; .pkg para Solaris y *BSD, .dmg para Mac OS X, ...) a partir de un único fichero .list.

Uno de los sistemas para **la distribución de aplicaciones en Linux**, aparte de los repositorios **SVN** de las aplicaciones es la web **sourceforge.net** que es una central de desarrollos de software que controla y gestiona varios proyectos de software libre y actúa como un repositorio de códigos fuente. Fue creado en 1999. En el año 2013 ha girado hacia posiciones más mercantilistas, mediante inserciones **adware**, aunque sigue manteniendo su esencia. Es la web de mayor influencia en la distribución de software.

Su funcionamiento se basa en la sincronización de tu proyecto con sourceForge además de gestionar las versiones de tu código, pudiendo crear tickets, tareas pendientes, dar de alta varios desarrolladores para ese proyecto, etc.. El proceso es el siguiente:

1. Se crea una cuenta en la web de sourceforge
2. Registros un nuevo proyecto (nombre, la ruta de la web del proyecto en source...)
3. Subir el código de la versión que tengas del proyecto

Según el tipo de lenguaje e IDE que utilizas la sincronización de tu versión del proyecto con el que has distribuido o es manual, o como en Java hay **addons** en Eclipse y en Netbeans que automatizan la sincronización de las sucesivas versiones que vas obteniendo.

Sistemas propietarios

En la actualidad, una gran variedad de sistemas de gestión de paquetes es usada por algunos sistemas operativos propietarios para tratar con la instalación tanto de paquetes propietarios como libres. De los que nos interesan tenemos el framework .NET de Microsoft, un ensamblado es una biblioteca de código parcialmente compilado destinado al uso en deployment, versioning y seguridad.

Gestión de paquetes incrustada en aplicaciones

Algunos sistemas de gestión de paquetes no forman parte nativa del sistema operativo, como pueden ser fink en Mac OS X o el entorno Unix-like de Cygwin (para Windows).

Algunos lenguajes de programación interpretados tienen su propio sistema de gestión de paquetes para manejar módulos del lenguaje, como pasa con los lenguajes de programación Perl (CPAN), Python (pip), PHP (PEAR) o Ruby (RubyGems). Otros programas pueden venir con su propio sistema para gestionar módulos.

ACTIVIDADES

Propuesta 1. Utilizando la aplicación realizada en el ejercicio 1 de la unidad 5 vamos a crear dos ejecutables para distribuir nuestra aplicación.

El primero que debeís crear es un ejecutable .exe para instalar en **Windows** y posteriormente un paquete .deb para Linux basadas en Debian.

Para Windows en el manual *Python para todos* en el capítulo *Distribuir aplicaciones de Python*, página 152, explica como obtener un paquete de instalación para Windows.

En cuanto a un paquete .deb en dicho manual en la página 147 hace referencia a una herramienta denominada **stdeb**. Hay otras técnicas más elaboradas, con el problema de que son técnicas hechas “a mano”. Aquí dejo un resumen de su utilización.

stdeb (paquete debian *python-stdeb*) se encarga de generar un paquete debian con todas las de la ley que podrás instalar con dpkg.

Veamos el siguiente ejemplo: vamos a instalar un módulo python que no está en debian: *python-gnupg* (verificar antes la versión)

```
$ wget http://python-gnupg.googlecode.com/files/python-gnupg-0.2.7.tar.gz
$ unp python-gnupg-0.2.7.tar.gz
$ cd python-gnupg-0.2.7
```

Aquí entra en juego stdeb:

```
$ python setup.py --command-packages=stdeb.command bdist_deb
```

Y esto te deja el paquete debian en el directorio deb_dist

```
$ sudo dpkg -i deb_dist/python-gnupg_0.2.7-1_all.deb
```

Obviamente si esto fuera tan bueno, nadie haría los paquetes debian “a mano”. stdeb no puede detectar conflictos entre paquetes, dependencias complejas y en general cualquier aspecto de la política de Debian que no sea automatizable (y hay bastante). Por ejemplo en este caso, el paquete *python-gnupg_0.2.7-1_all.deb*, es decir, le ha puesto el prefijo «python» por ser un módulo Python a pesar de que el nombre original ya tenía ese prefijo.

En cualquier caso, es útil para probar un módulo Python sin “ensuciar” mucho el sistema. Si el paquete está en PyPI puedes descargar la fuente, crear el paquete e instalarlo con el comando:

```
$ pip install gnupg
```

La siguiente web puede ser de ayuda para crear paquetes deb:

<http://mundogeek.net/archivos/2008/09/23/distribuir-aplicaciones-python/>

UD8. Realización de pruebas

RA8. Evalúa el funcionamiento de las aplicaciones para lo que diseña y ejecuta pruebas.

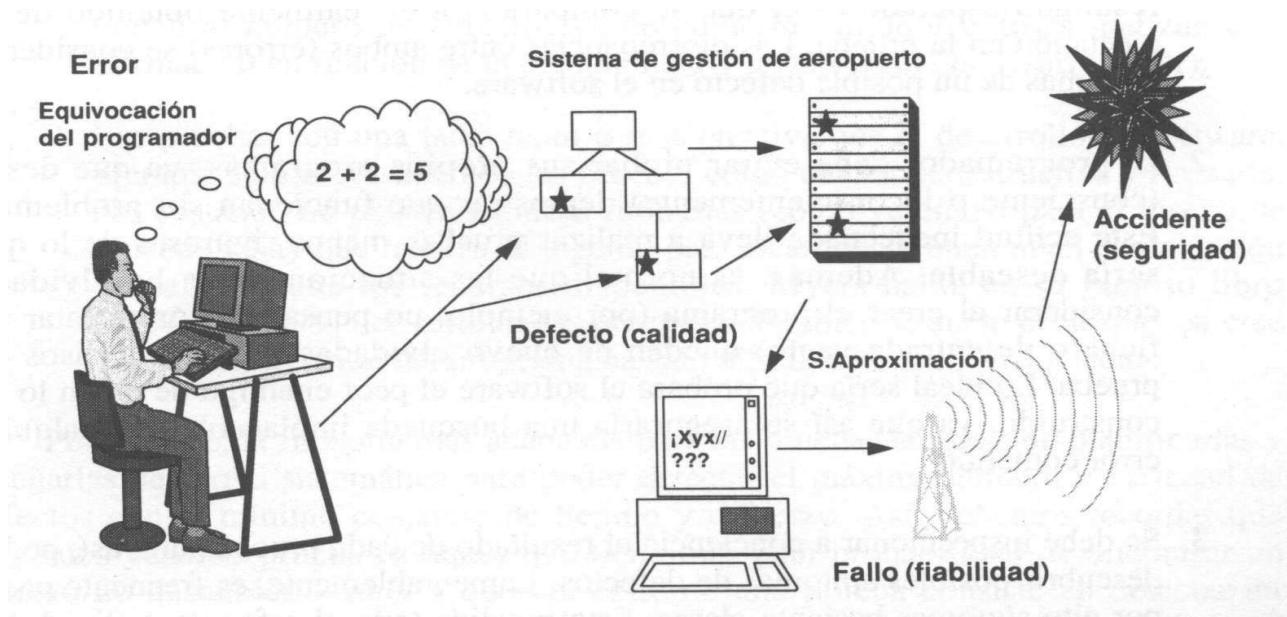
- CA3.1. Se estableció una estrategia de pruebas.
- CA3.2. Se realizaron pruebas de integración de los elementos.
- CA3.3. Se realizaron pruebas de regresión
- CA3.4. Se realizaron pruebas de volumen y estrés.
- CA3.5. Se realizaron pruebas de seguridad.
- CA3.6. Se realizaron pruebas de uso de recursos por parte de la aplicación.
- CA3.7. Se documentó una estrategia de pruebas y se analizaron los resultados obtenidos.

BC8. Realización de pruebas

- Objetivo, importancia y limitaciones del proceso de prueba. Estrategias.
- Pruebas de integración: ascendentes y descendentes.
- Pruebas de sistema: configuración, recuperación, etc.
- Pruebas de regresión.
- Pruebas de uso de recursos.
- Pruebas de seguridad.
- Pruebas manuales y automáticas. Herramientas de software para la realización de pruebas.
- Pruebas de aceptación. Versiones alfa e beta.

→ Objetivo, importancia y limitaciones del proceso de prueba. Estrategias.

Las **pruebas de software** son estudios empíricos y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto desarrollado al futuro usuario y/o comprador. Son una actividad más en el proceso de **desarrollo de software**. Básicamente son un conjunto de actividades como una etapa más dentro del desarrollo de software.



Los errores en el software pueden causar víctimas

La **prueba exhaustiva del software es impracticable** (no se pueden probar todas las posibilidades de su funcionamiento ni siquiera en programas sencillos. Por ejemplo, en un programa sencillo de suma de dos números es imposible probar todos los casos (todos los números) a sumar. El objetivo de las pruebas **no es asegurar la ausencia de defectos** en un software, únicamente puede demostrar que existen defectos en el software.

El objetivo de las pruebas es la **detección de defectos** en el software (descubrir un error es el éxito de una prueba). El hallazgo de un error o defecto no implica que seamos malos profesionales ya que todo el mundo comete errores. Es más, **el descubrimiento de un defecto significa un éxito para la mejora de la calidad del producto.**

Por otro lado, el proceso de **verificación** es el proceso de evaluación de un sistema (o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase, es decir, responde a la pregunta: "*¿estamos construyendo el producto correctamente?*"

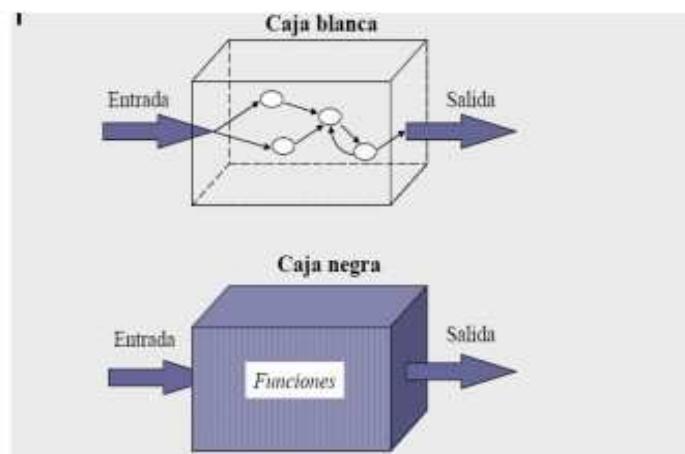
Finalmente esta el proceso de **validación** o proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos marcados por el usuario, es decir, responde a la pregunta: "*¿estamos construyendo el*

producto correcto?"

Veamos a continuación una serie de definiciones importantes en el campo de las pruebas de software algunas de las cuales mencionaremos en apartados posteriores:

- **Pruebas (test):** actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de uno o varios aspectos concretos.
- **Caso de prueba (test case):** conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular.
- **Defecto (defect, fault, «bug»):** un defecto en el software como, por ejemplo, un proceso, una definición de datos o un paso de procesamiento incorrectos en un programa. Por ejemplo: “el software es incapaz de cargar los sumandos”
- **Fallo (failure):** La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados. Por ejemplo: “carga los datos pero no los suma”.
- **Error (error):** La diferencia entre un valor calculado, observado o medio y el valor verdadero, especificado o teóricamente correcto. Por ejemplo: “la suma es incorrecta”.

Las **pruebas de caja negra o integración** también conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba independientemente de su diseño interno o “*o de como lo haga*”.



El componente se ve como una “Caja Negra” cuyo comportamiento **sólo puede ser determinado estudiando sus entradas y las salidas obtenidas a partir de ellas**.

Las **pruebas de Caja Blanca o Estructurales** a este tipo de técnicas se le conoce también como Técnicas de Caja Transparente o de Cristal. Este método se centra en cómo diseñar los casos de prueba atendiendo al **comportamiento interno y la estructura del programa**, es decir, interesa . Se examina así la lógica interna del programa sin considerar los aspectos de rendimiento.

El objetivo de la técnica es **diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa**, y todas las condiciones tanto en su vertiente verdadera como falsa. De estas pruebas se obtiene la **complejidad ciclomática de los algoritmos**.

Veremos un ejemplo sencillo:

La complejidad ciclomática se basa en la teoría gráfica y se calcula de tres maneras:

1. **Número de regiones**
2. **Complejidad ciclomática es igual a número de aristas, menos el número de nodos más**

$$V(G) = E - N + 2$$
3. **Complejidad ciclomática es igual al número de nodos predicable más uno**

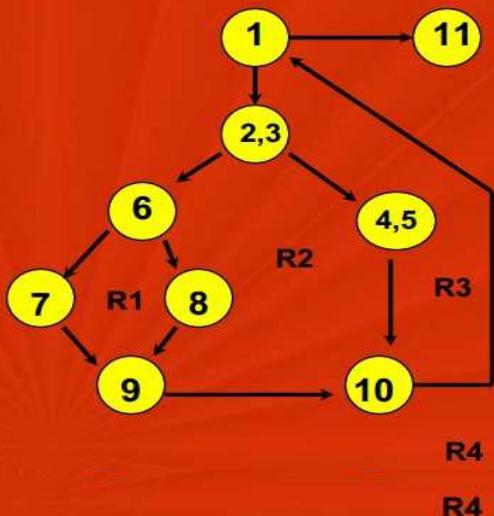
$$V(G) = P + 1$$

Formas de Calcular la complejidad ciclomática



Calculamos las diferentes rutas.

Calculamos el número de regiones

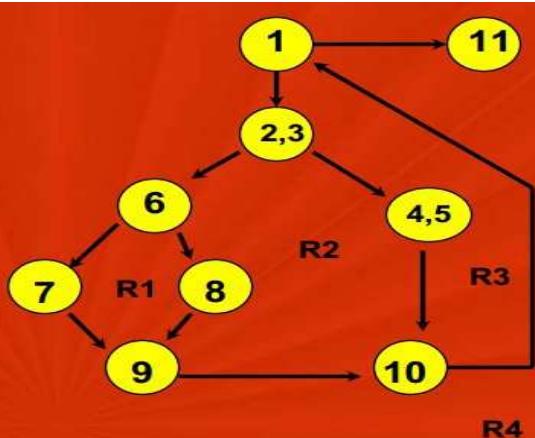


2. Complejidad ciclomática es igual a número de aristas, menos el número de nodos más 2

$$V(G) = E - N + 2$$

$$V(G) = 11 - 9 + 2 = 4$$

Forma de Calcular la $V(G)$ (I)



3. Complejidad ciclomática es igual al número de nodos predicado más uno

$$V(G) = P + 1$$

$$V(G) = 3 + 1 = 4$$

Forma de Calcular la $V(G)$ (II)

Para la realización con éxito de unas pruebas sobre el software he aquí una serie de consejos o recomendaciones:

- Cada caso de prueba debe definir el **resultado de salida esperado** que se comparará con el realmente obtenido.
- El programador **debe evitar probar sus propios programas**, ya que desea (consciente o inconscientemente) demostrar que funcionan sin problemas. Además, es normal que las situaciones que olvidó considerar al crear el programa queden de nuevo olvidados al crear los casos de prueba.
- Se debe **inspeccionar a conciencia** el resultado de cada prueba, así, poder descubrir posibles síntomas de defectos.
- Al generar casos de prueba, se deben incluir tanto **datos de entrada válidos y esperados**

como no válidos e inesperados.

- Las pruebas deben centrarse en dos objetivos:
 - Se deben **evitar los casos desecharables**, es decir, los no documentados ni diseñados con cuidado.
 - No deben hacerse planes de prueba suponiendo que, prácticamente, no hay defectos en los programas y, por lo tanto, dedicando pocos recursos a las pruebas siempre hay defectos.
- Probar **si el software no hace lo que debe hacer**.
- Probar **si el software hace lo que debe hacer**, es decir, si provoca efectos secundarios adversos.
- La experiencia parece indicar que **donde hay un defecto hay otros**, es decir, la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al número de defectos ya descubierto.
- Las pruebas **son una tarea tanto o más creativa** que el desarrollo de software. Siempre se han considerado las pruebas como una tarea destructiva y rutinaria.
- Es interesante planificar y diseñar las pruebas para poder detectar el máximo número y variedad de defectos con el **mínimo consumo de tiempo y esfuerzo**.

Las tareas a realizar para probar un software según el orden establecido son:

- **Diseño de las pruebas.** Esto es, identificación de la técnica o técnicas de pruebas que se utilizarán para probar el software. Distintas técnicas de prueba ejercitan diferentes criterios como guía para realizar las pruebas. Seguidamente veremos algunas de estas técnicas.
- **Generación de los casos de prueba.** Los casos de prueba representan los datos que se utilizarán como entrada para ejecutar el software a probar. Más concretamente los casos de prueba determinan un conjunto de entradas, condiciones de ejecución y resultados esperados para un objetivo particular. Como veremos posteriormente, cada técnica de pruebas proporciona unos criterios distintos para generar estos casos o datos de prueba.
- **Definición de los procedimientos de la prueba.** Esto es, especificación de cómo se va a llevar a cabo el proceso, quién lo va a realizar, cuándo, ...
- **Ejecución de la prueba,** aplicando los casos de prueba generados previamente e identificando los posibles fallos producidos al comparar los resultados esperados con los resultados obtenidos.
- **Realización de un informe de la prueba,** con el resultado de la ejecución de las pruebas, qué casos de prueba pasaron satisfactoriamente, cuáles no, y qué fallos se detectaron.

→ Pruebas de integración: ascendentes y descendentes.

Aún cuando los módulos de un programa funcionen correctamente por separado es necesario probarlos conjuntamente. Un módulo puede tener un efecto adverso o inadvertido sobre otro módulo; las subfunciones, cuando se combinan, pueden producir la función principal o un resultado indeseado; la imprecisión aceptada individualmente puede crecer hasta niveles inaceptables al combinar los modulos.

Por lo tanto, es necesario probar el software ensamblando todos los módulos probados previamente. Y este **es el objetivo de la pruebas de integración**.

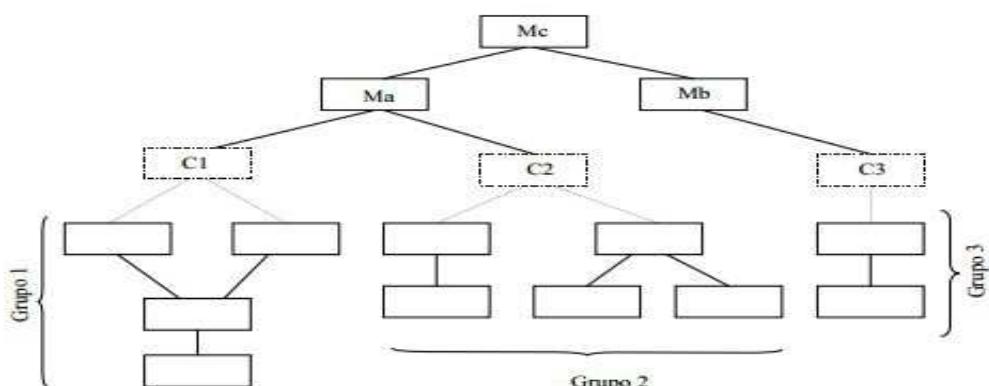
A menudo hay una tendencia a intentar una **integración no incremental**; es decir, a combinar todos los módulos y probar todo el programa en su conjunto. El resultado puede ser un poco caótico con un gran conjunto de fallos y la consiguiente dificultad para identificar el módulo (o módulos) que los provocó.

Por ello y, a veces, se puede aplicar la **integración incremental** en la que el programa se prueba en pequeñas porciones en las que los fallos son más fáciles de detectar. Existen dos tipos de integraciones incrementales, la denominada **ascendente y descendente**.

- **Integración incremental ascendente:**

- a. Se combinan los módulos de bajo nivel en grupos que hagan una subfunción específica.
- b. Se escribe un controlador (un programa de control de la prueba) para coordinar la entrada y salida de los casos de prueba.
- c. Se prueba el grupo
- d. Se eliminan los controladores y se combinan los grupos moviéndose hacia arriba por la estructura del programa.

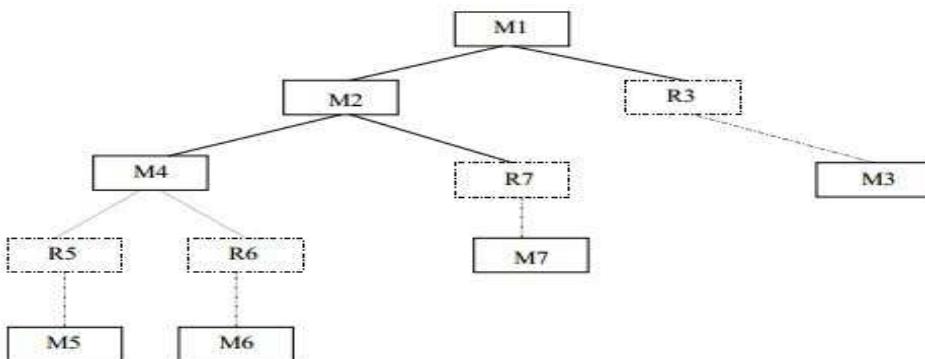
En la figura de ejemplo, se forman los grupos 1, 2 y 3 de módulos relacionados, y cada uno de estos grupos se prueba con el controlador C1, C2 y C3 respectivamente. Seguidamente, los grupos 1 y 2 son subordinados de Ma, luego se eliminan los controladores correspondientes y se prueban los grupos directamente con Ma. Análogamente se procede con el grupo 3 eliminando el controlador C3 y probando el grupo directamente con Mb. Tanto Ma y Mb se integran finalmente con el módulo Mc y así sucesivamente.



- **Integración incremental descendente**

- a. Se usa el **módulo de control principal** como controlador de la prueba, creando resguardos (módulos que simulan el funcionamiento de los módulos que utiliza el que está probando) para todos los módulos directamente subordinados al módulo de control principal.
- b. Dependiendo del enfoque e integración elegido (es decir, primero-en-profundidad, o primero-en-anchura) se van sustituyendo uno a uno los resguardos subordinados por los módulos reales.
- c. Se llevan a cabo pruebas cada vez que se integra un nuevo módulo.
- d. Tras terminar cada conjunto de pruebas, se reemplaza otro resguardo con el módulo real.

En la figura siguiente vemos una prueba de integración descendente. Supongamos que se selecciona una integración descendiente por profundidad, y que por ejemplo se prueba M1, M2 y M4. Sería entonces necesario preparar resguardos para M5 y M6, y para M7 y M3. Estos resguardos se han representado en la figura como R5, R6, R7 y R4 respectivamente. Una vez realizada esta primera prueba se sustituiría R5 por M5, seguidamente R6 por M6, y así sucesivamente hasta probar todos los módulos.



➔ Pruebas de sistema

Este tipo de pruebas tiene como propósito probar profundamente el sistema para verificar que se han integrado adecuadamente todos los elementos del sistema (hardware, otro software, etc.) y que realizan las funciones adecuadas. Concretamente se debe comprobar que:

- Se cumplen los requisitos funcionales establecidos.
- El funcionamiento y rendimiento de las interfaces hardware, software y de usuario.
- La adecuación de la documentación de usuario.
- Rendimiento y respuesta en condiciones límite y de sobrecarga.

Para la generación de casos de prueba de sistema se utilizan técnicas de **caja negra**. Este tipo de pruebas se suelen hacer inicialmente en el **entorno del desarrollador**, denominadas **Pruebas**

Alfa, y seguidamente en el **entorno del cliente** denominadas **Pruebas Beta**.

Se distinguen los siguientes tipos de pruebas (algunas solo se mencionaran porque su nombre es autoexplicativo):

- **Pruebas de comunicaciones.** Comprueba que las interfaces tanto locales como remotas funcionan adecudamente.
- **Pruebas de rendimiento.** Comprueba los tiempos de respuesta de la aplicación.
- **Pruebas de recuperación.** Se fuerza el fallo del software para comprobar su capacidad de recuperación.
- **Pruebas de volumen.** Se comprueba su funcionamiento con cantidades próximas a su capacidad total de procesamiento
- **Pruebas de sobrecarga.** Similar al anterior pero en el límite de su capacidad.
- **Pruebas de tensión.** Es similar a la prueba de volumen pero se restringe el tiempo disponible (algo como determinar la velocidad de trabajo)
- **Pruebas de disponibilidad de datos.** Comprueba si tras la recuperación el sistema mantuvo la integridad de los datos.
- **Pruebas de facilidad de uso.** Refiriéndose al usuario final.
- **Pruebas de operación.** Comprueba la correcta implementación de los procedimientos de operación, incluyendo la planificación y control de trabajos, arranque y rearranque del sistema, etc...
- **Pruebas de entorno.** Comprueba la interacción con otros sistemas.
- **Pruebas de seguridad.** Comprobación de los mecanismos de control de acceso al sistema.
- **Pruebas de usabilidad.** Ya visto en la UD 4.
- **Pruebas de almacenamiento.** Comprobación de la cantidad de memoria principal y secundaria que el programa usa y el tamaño de los archivos temporales.
- **Pruebas de configuración.** Aunque a veces resulta imposible se debe comprobar el programa con cada tipo de hardware, sistema operativo, antivirus....
- **Pruebas de instalación.** En especial la automatización para facilidad del usuario final.
- **Pruebas de la documentación.** Hace referencia tanto a la documentación técnica para desarrolladores futuros como a la documentación para el usuario.

→ Pruebas de regresión

Se denominan **pruebas de regresión** a cualquier tipo de pruebas que intentan descubrir las causas de nuevos errores o bugs, carencias de funcionalidad, o divergencias funcionales con **respecto al comportamiento esperado del software**, inducidos por **cambios recientemente realizados en partes de la aplicación** que anteriormente al citado cambio no eran propensas a este tipo de error. Esto implica que el error tratado se reproduce como consecuencia inesperada del citado cambio o modificación realizada en el programa para su mejor o adaptación a nuevas necesidades del cliente.

Este tipo de cambio puede ser debido a prácticas no adecuadas del **control de las versiones**, falta de consideración acerca del ámbito o contexto de producción final y extensibilidad del error que fue corregido (fragilidad de la corrección), o simplemente una consecuencia del rediseño de la aplicación. El **objetivo de las pruebas de regresión es eliminar el efecto onda**, es decir, comprobar que los cambios sobre un componente de un sistema de información, no introducen un comportamiento no deseado o errores adicionales en otros componentes no modificados

Por lo tanto, en la mayoría de las situaciones del desarrollo de software se considera una buena práctica que cuando se localiza y corrige un bug, se grabe una prueba que exponga el bug y se vuelvan a probar regularmente después de los cambios subsiguientes que experimente el programa.

Existen herramientas de software que permiten detectar este tipo de errores de manera parcial o totalmente automatizada. La práctica habitual en programación es que este tipo de pruebas se ejecuten en cada uno de los pasos del **ciclo de vida del desarrollo del software**.

Para mitigar los riesgos en las modificaciones realizadas en los módulos del programa ya desarrollados:

- **Repetición completa** y habitual de la **batería de pruebas**, manual o mediante automatización.
- **Repetición parcial** basada en trazabilidad y análisis de los posibles riesgos.
- **Pruebas de cliente** o usuario:
 - **Beta** - distribución a clientes potenciales y actuales de las **versiones Beta**.
 - **Pilot** - distribución a un subconjunto bien definido y localizado.
 - **Paralela** - simultaneando uso de ambos sistemas. Usar releases (**software candidato a definitivo**) mayores. Probar nuevas funciones a menudo cubre las funciones existentes. Cuantas más nuevas características haya en un release, habrá mayor nivel

de pruebas de regresión "accidental".

- **Parches de emergencia** - estos parches se publican inmediatamente, y serán incluidos en releases de mantenimiento futuras.

Las Pruebas de Regresión pueden usarse no solo para probar la **corrección** de un programa, sino a menudo usarse para rastrear la calidad de su salida. Por ejemplo, en el diseño de un compilador, las pruebas de regresión deben rastrear el tamaño del código, tiempo de simulación, y el tiempo de compilación de las suites de prueba. Cuando quiera que aparece un nuevo build, el proceso de regresión aparece.

→ Pruebas de uso de recursos

Esta relacionado con las **pruebas de rendimiento**. Estas son las pruebas que se realizan para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. Dentro de ellas están las pruebas del uso de los recursos preferentemente hardware, red, sistema operativo...

Por ejemplo, puede ser el número de sesiones que un servidor de aplicaciones puede soportar o el número de acceso que un servidor de datos es capaz de transferir llegándose al máximo o **pruebas de stress** con el objetivo de que la aplicación falle en esas condiciones.

→ Pruebas de seguridad

La prueba de seguridad intenta verificar que los mecanismos de protección incorporados en el sistema. Estas pruebas van desde la introducción de datos incorrectos o inapropiados, intentos de acceso no autorizados, pruebas de control de la integridad de los datos, etc...

→ Pruebas manuales y automáticas. Herramientas de software para la realización de pruebas.

La **prueba manual** se realiza ejecutando el software, introduciendo datos de entrada e inspeccionando los de salida. El proceso es sencillo y no hace falta aprendizaje previo, ademas ejecutamos las pruebas exactamente como el usuario final pero tiene algunos inconvenientes.

- **Repetitiva.** Cada vez que se cambia o añade una nueva funcionalidad, o se corrige un error, necesitamos "reprobar" las aplicaciones para asegurarnos de que no se ha producido un error por causa de este cambio.
- **Susceptible de error.** Precisamente por ser repetitiva, es posible que pasemos por alto detalles a la hora de probar, o que no se prueben funcionalidades aparentemente no

relacionadas con el cambio, pero realmente afectadas. Es imposible conocer todas las interrelaciones existentes entre las funcionalidades de nuestro software.

- **Solo pruebas lo que ves.** Esto significa que solo pruebas indirectamente lo que no ves, y es especialmente el caso de las aplicaciones de la parte servidor. Necesitamos que funcionen correctamente la capa de presentación y la capa de negocio, pero especialmente la capa de negocio residente en el servidor.
- **Con la prueba manual,** otros no pueden verificar el correcto funcionamiento de nuestra aplicación. Otras personas deben aceptar que nuestra prueba manual ha sido correcta y satisfactoria, o realizar ellos mismos una nueva prueba manual, que quizás no están preparados para realizar por no conocer la lógica de nuestro software.

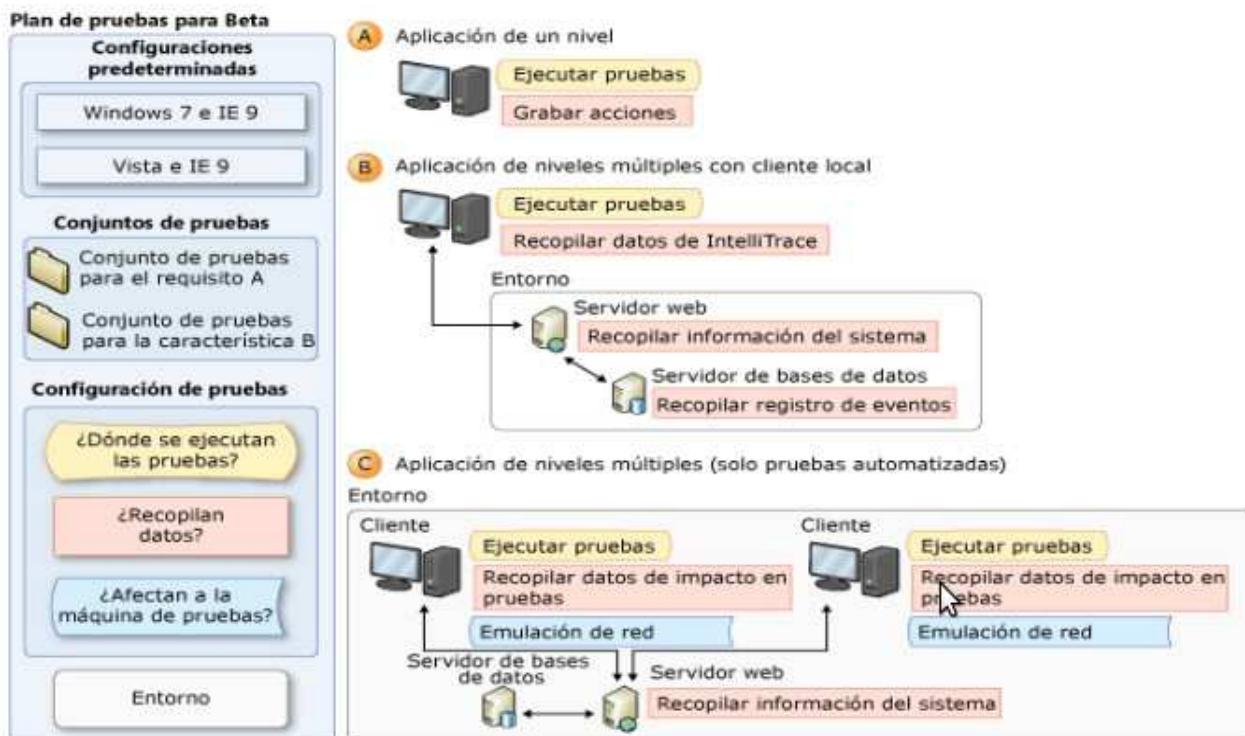
La **prueba automatizada** se integra en las metodologías modernas de una manera integral en el desarrollo del software. La prueba automatizada es repetible y portable. Repetible significa que la prueba se puede repetir indefinidamente produciendo los mismos resultados siempre, y portable se refiere a que otras personas pueden ejecutar la prueba y verificar que el software pasa todas las pruebas. Por otro lado, la prueba evoluciona con el software, de forma que si los requisitos y funcionalidad del software cambian, la prueba debe cambiar también. No es muy adecuada una prueba que no verifica el correcto funcionamiento del software en su estado actual.

Realizando pruebas automáticas, podemos tener dos tipos: **prueba funcional y prueba unitaria**.

La **prueba funcional** es de tipo “caja negra” realizada sin conocimiento interno de la aplicación, a alto nivel, simulando la actuación del usuario.

La **prueba unitaria**, por el contrario, implica un conocimiento del software a bajo nivel, no solo las entradas y las salidas. Se deben probar todos los métodos y clases importantes, e implica simular las entradas de información. Proporciona también soluciones a los problemas antes planteados:

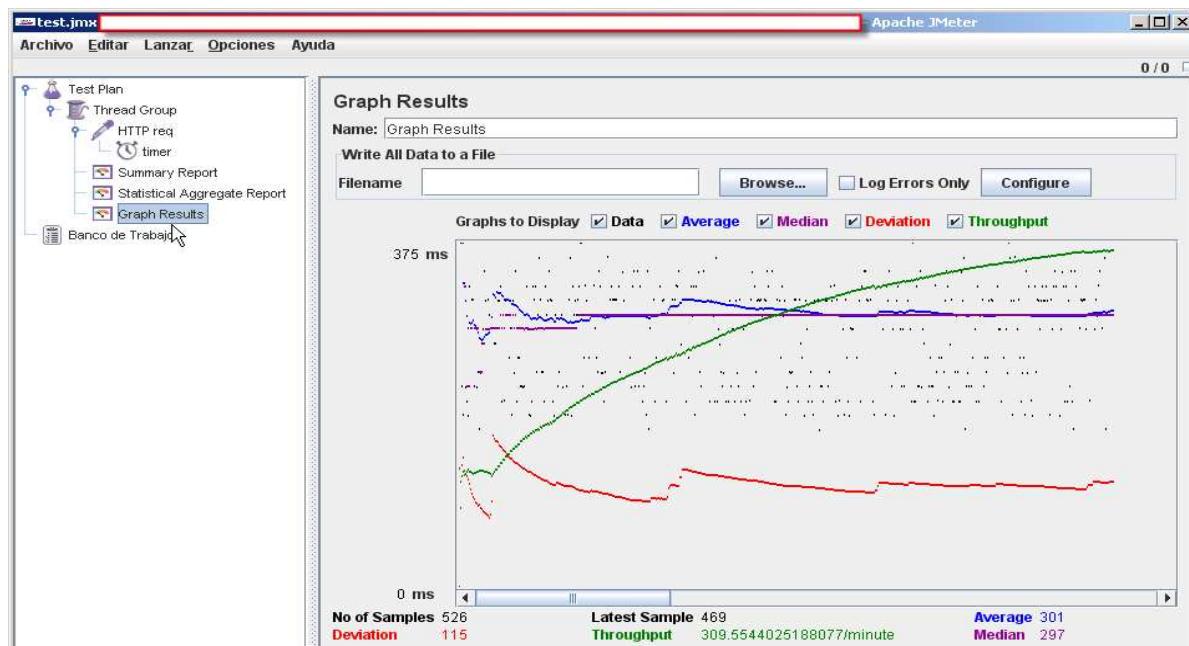
- **Elimina el componente repetitivo.** Se transmite el trabajo repetitivo al ordenador (para eso está), que seguirá exactamente la secuencia de pruebas, cada vez que lo solicitemos.
- **Reduce los errores.** Cada repetición será exactamente igual cada vez que se ejecute, de forma que se puede probar fácilmente todas las funcionalidades ante cualquier cambio sin temor a olvidar alguna parte.
- **Permite probar las partes no visibles de código,** el corazón de la aplicación realmente. El nivel de detalle depende de la programación de la prueba.
- **Permite a los usuarios finales** verificar que el software funciona como debe.



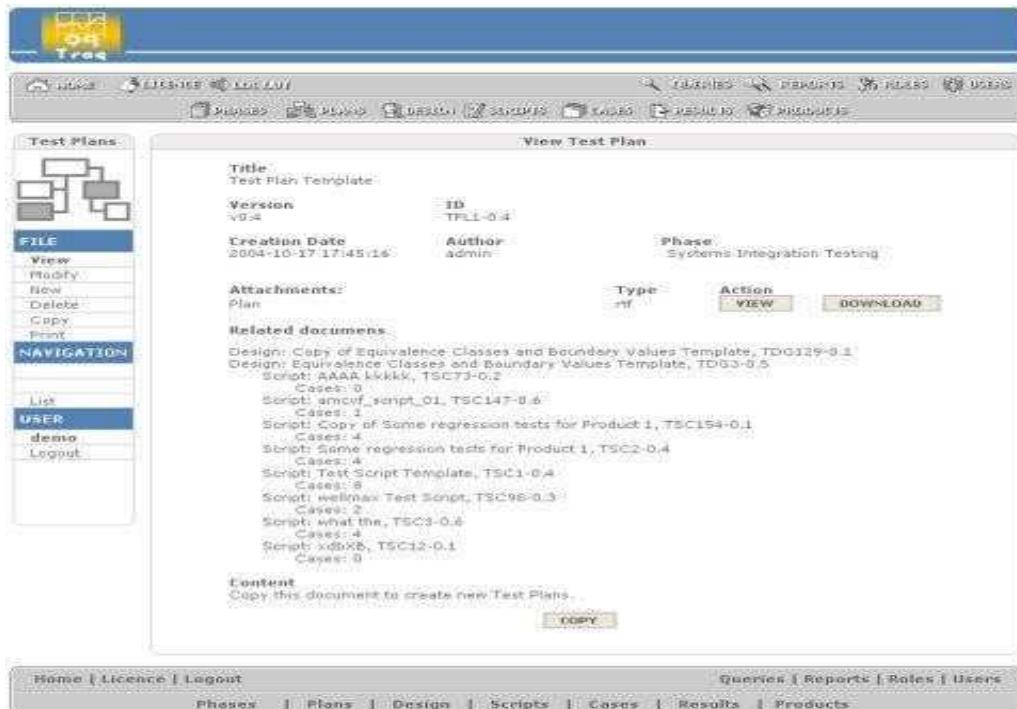
Ejemplo de esquema de prueba para aplicación en Microsoft Visual Studio

Entre las herramientas que tenemos para las pruebas automatizadas de software (solo freeware):

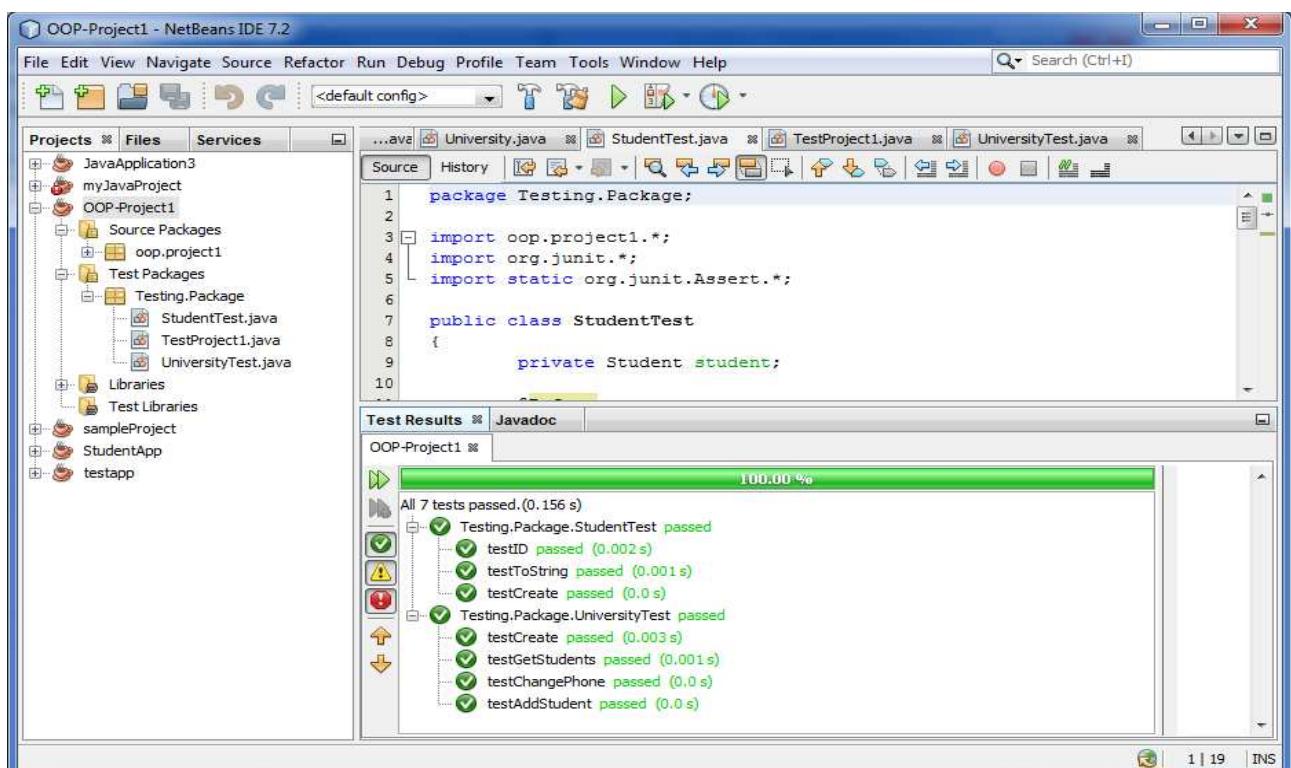
- **Para el seguimiento de defectos:** *Bugzilla*, *BugRat*. Realmente son bases de datos que van almacenando los defectos encontrados en las diferentes versiones
- **Para evaluación de pruebas de carga y rendimiento:** *Jmeter* (específica para aplicaciones web creada por Apache Foundation)



- **Para la gestión y manejo de pruebas:** *rth*, *QaTraq*. Ambas almacenan los resultados de las diferentes tipos de pruebas.



- **Para pruebas de unidad:** *JUnit*. Muy enfocada a las pruebas de regresión.



IBM tiene el **Rational Software de IBM** que contiene diferentes módulos según el tipo de pruebas que puede considerarse como la herramienta más avanzada para este campo.

➔ Pruebas de aceptación. Versiones alfa y beta.

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y **permitir al usuario de dicho sistema que determine su aceptación**, desde el punto de vista de su funcionalidad y rendimiento.

Las pruebas de aceptación son definidas por el usuario del sistema y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación final corresponden al usuario.

Cuando se construye software a medida para un cliente, se lleva a cabo una serie de **pruebas de aceptación** para permitir que el cliente valide todos los requisitos. La mayoría de los desarrolladores de productos de software llevan a cabo un proceso denominado pruebas alfa y beta para descubrir errores que parezca que sólo el usuario final puede descubrir.

- **Prueba alfa:** (**versión alfa**) se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y problemas de uso. Las pruebas alfa se llevan a cabo en un entorno controlado.
- **Prueba beta:** (**versión beta**) se llevan a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador. El cliente registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.

ACTIVIDADES

Ejercicio 1. Dado el siguiente pseudocódigo:

```

var num1= 0,
    num2 = 5;

if( num1 == 0 ){

    if( num2 == 5){
        console.log( 'num1 = 0');
        console.log ('num2 = 5');
    }

    else{
        console.log('Only num1 = 0');
    }
}

```

- Establece el grafo de flujo del mismo.
- Determina la complejidad ciclomática de las tres formas descritas.

Ejercicio 2. Dado el siguiente algoritmo:

```

1 Begin {programa}
2 Print ("Ilmo.Sr.Director General:");
3 Read (registro_FICH);
4 Prima = 0;
5 While (no FF de FICH) do
6     Begin {while}
7         If (meses_FICH>=12)
8             Then
9                 If (directivo_FICH="+")
10                    then Prima=1000;
11                    else Prima=75;
12             Else
13                 If (directivo_FICH="+")
14                     then Prima=500;
15                     else;
16             Print (num_FICH, nombre_FICH, Prima);
17             Read (registro_FICH);
18     End {while};
19     Print ("S.e.u.o.");
20 End {programa}

```

- Establece el grafo de flujo del mismo.
- Determina la complejidad ciclomática de las tres formas descritas.

ANEXO

Si observamos los contenidos del módulo *Desarrollo de Interfaces* nos percatamos de que, a grandes rasgos, representan las diferentes etapas del desarrollo de una aplicación. En mi opinión dicho módulo debería complementarse con un proyecto global que el alumno/alumna desarrolle a lo largo del curso poniendo en práctica todos los aspectos vistos en el presente manual. Ello le permitirá tener una visión global de lo que se pretende en dicha materia.

Así propongo los siguientes proyectos en los cuales se aconseja tratar los siguientes puntos:

- Diagrama Entidad Relación. Estructura de datos.
- Diagrama de Flujo de Datos.
- Diagrama de Componentes (diseño modular).
- Diagrama de Casos de Uso.
- Código debidamente comentado.
- Diseño de pruebas.

Proyecto 1. Elaboración de un proyecto de software de Contabilidad Doméstica para usuarios particulares. Los requisitos que te han pedido son los siguientes:

- El programa configurará cuantas cuentas bancarias desee el usuario.
- En dichas cuentas se introducirán los ingresos, gastos de dinero y en los cuales deberá constar el concepto de una lista predefinida (alimentación, gasolina, sueldo...), así como la fecha y el valor.
- Dicha lista, aunque predefinida tras la instalación, el usuario podrá introducir nuevos conceptos o añadir subcategorías dentro de los existentes, por ejemplo, en alimentación, incluir supermercado o salir a cenar.
- También podremos registrar transacciones periódicas que se efectúen de forma manual por parte del usuario o bien de forma automática llegada la fecha en una determinada cuenta (recibos de luz, sueldos....). Dicha elección la configurará el usuario.
- También podrá registrar transferencias entre las cuentas configuradas en el programa.
- En principio se utilizará como moneda el euro e idioma castellano.
- El programa deberá realizar diferentes gráficos, listados,... Por ejemplo, relación de gastos e ingresos por intervalos de tiempo, listados de categorías de gastos e ingresos por valor...
- El programa podrá acceder a diferentes herramientas del sistema como la calculadora.
- Además deberá realizar a petición del usuario backups sencillos.
- Deberá contener una ayuda o manual de uso para el usuario.

Proyecto 2. El proyecto CEB consiste en la gestión de una empresa de empeños y compraventa de artículos usados. Los módulos principales del proyecto son:

- Gestión de vendedores, es decir aquellos clientes que llevan un artículo para empeñarlo o venderlo
- Gestión de artículos. Pueden ser productos empeñados, es decir, el antiguo propietario puede recuperarlos pagando el dinero prestado y un interés y artículos vendidos que son directamente puestos a la venta.
- Gestión de ventas. En algunos casos se puede requerir los datos personales en determinados artículos, es decir, el programa permite dar de alta un usuario como vendedor o comprador ambos.
- Gestión de empleados. Sobre todo de aquellos que llevan gestiones de compra de artículos o venta de productos de alta gama. Para simplificar, cada compraventa debe llevar asociado un vendedor.
- Gestión de informes. Van desde listados, facturas, documento de empeños...

Como mencionamos antes el empeño consiste en dejar en depósito un artículo por un tiempo estipulado como aval de un préstamo de dinero. Transcurrido ese tiempo el artículo puede ser puesto a la venta en la zona pública ya que pasa a ser propiedad de la casa de empeños. Por el contrario el propietario si quiere recuperarlo necesitará abonar la cantidad recibida más un interés mensual (para simplificar usaremos como unidad el mes) por dicha cantidad.

Proyecto 3. Desarrollo de un software para la gestión de una Nave Alquiler de Trasteros.

El programa tiene los siguientes módulos:

- Gestión de los Trasteros: en este módulo permite dar de alta un trastero dentro de la nave. Para ello deberemos aparte de un número identificativo que lo localice en un croquis. Consideraremos todos los trasteros de igual tamaño y precio de alquiler.
- Gestión de Clientes: precisamos aquellos datos que permitan además de localizar el trastero pasar la cuota por banco del alquiler del mismo.
- Gestión de Alquileres: asignamos a un cliente uno o más trasteros. Los alquileres serán mensuales o anuales.
- Gestión de Impagos: tras un período de tiempo determinado por el contrato de falta de pago de cuotas, el programa lanzará un aviso para enviar el correspondiente burofax o llamada al cliente para hacer el pago. En el caso de que no lo afrontara al cabo de otro tiempo se llevará a cabo la subasta del contenido del trastero.
- Gestión de Subastas: aquellos trasteros de los que no se pagaron sus cuotas tal como se estableció en el contrato pasará a ser subastado su contenido. La gestión de subastas indicará el trastero, fecha de la subasta, precio pujado por su contenido y nombre pujador.
- Gestión de informes: aquí van desde la elaboración de un documento-tipo para el contrato, listado de trasteros, croquis de su distribución, listado de clientes.... y todos aquellos que se consideren oportunos.

Proyecto 4. Una ONG desea llevar a cabo un proyecto para actuar como intermediaria entre particulares para la donación y/o intercambio de libros de texto de sus hijos.

El software de gestión tendrá los siguientes módulos.

- Gestión de Solicitantes: solo darán datos básicos y entre ellos un teléfono de contacto. Entre los datos incluirán el título del libro o libros que necesitan, editorial, curso del hijo o hijos, colegio e ISBN
- Gestión de Libros: recogerá los datos de libros que se donan. Además de los anteriores su estado entre 3 o 4 posibles (excelente, bueno...) y algún comentario al respecto. Asimismo la localización del libro en las estanterías.
- Gestión de Donantes: aunque no es obligatorio se permitirá la recogida de estos datos que serán básicos.
- Gestión de Vales o Entregas: conforme los libros con los datos correspondientes fueron entregados a un posible receptor.

Proyecto 5. El sector de la Geriatría es considerado como uno de los de mayor futuro. La gestión de un centro geriátrico puede exigir los siguientes módulos:

- Gestión de Residentes. No se debe usar pacientes ya que no son enfermos. Los datos a incluir podrían ser datos personales, datos familiares y datos de cuenta si la RG es privada
- Gestión de Sanitaria-Alimentación. Un residente geriátrico suele precisar de una alimentación determinada y, seguramente medicación. Todo ello debe ser registrado para que el personal de cocina sepa que tipo de alimentos a preparar y el personal sanitario que medicación y con que pauta.
- Gestión de Alojamiento: cada residente debe estar inscrito en una habitación determinada. A veces puede ser necesario un registro de sus pertenencias personales ...
- Gestión de Actividades: como su nombre indica podemos realizar actividades de diferente tipo y asignar residentes a ellas a los residentes.
- Gestión de Empleados: una RG puede tener diferentes tipos de empleados que van desde personal sanitario, asuntos sociales, dirección, administración, cocina, lavandería....

**Manual de apoyo para la docencia del módulo
profesional Desarrollo de Interfaces perteneciente
al Ciclo Formativo de Desarrollo de Aplicaciones
Multiplataforma.**