MÓDULO PROFESIONAL

DESENVOLVEMENTO DE INTERFACES

Profesor: Juan Carlos Pérez González

Introducción

No fue hasta finales de la década de los sesenta cuando aparecieron las primeras interfaces gráficas de usuario tal como las entendemos hoy en día. Anteriormente, las interfaces de usuario eran simples **CLI (Interfaces Command Line)** donde se introducían las órdenes mediante comandos con sus respectivas **opciones.**

Posteriormente aparecieron los primeros interfaces de usuario (IU) con **menús jerárquicos** como, por ejemplo, *Windows 3.1 o Apple Lisa*. A principios de los 90, con el OS2 /2.0 surgen términos como **usabilidad** donde se le da más importancia a la experiencia de usuario dando como producto las interfaces **WIMP** (*Windows, Icons, Menus and Pointing device*), es decir, **ventanas, iconos, menus y punteros**. Más tarde se introducen elementos como la simulación 3D, mejoras en el aspecto visual y, en especial, la **programación orientada a eventos** que no es más que una versión de la programación orientada a objetos dirigida al manejo de las interfaces gráficas. En el siguiente **enlace** se puede observar la evolución de las interfaces (hasta 2012) en lo que hace referencia a diferentes sistemas operativos. En el futuro, presente ya, se va hacia la generación **eventos mediante voz o vista e incluso** que el ordenador tome sus decisiones y acciones en función de la observación del usuario, machine learning y AI.

- **La Metáfora del escritorio**

Un elemento que se circunscribe a los sistemas operativos y está presente en la mayoría de ellos es la denomina **metáfora del escritorio.**

La **metáfora de escritorio** introducida por primera vez en 1970 por Alan Kay en la empresa Xerox PARC no es más que es un conjunto de conceptos unificadores usados por las interfaces gráficas de usuario para ayudar a los usuarios a interactuar más fácilmente con la máquina. Esta 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.

Un nuevo elemento que ha surgido con los dispositivos móviles es **la interfaz natural de usuario o NUI** es el tipo de interfaz de usuario en las que se interactúa con un sistema, aplicación, etcétera, sin utilizar sistemas de mando o dispositivos de entrada (ratón, teclado alfanumérico, panel táctil, joystick, etcétera), sino que, **se hace uso de movimientos gestuales del cuerpo o de alguna de sus partes** tales como las manos.

En el caso de pantallas **capacitivas multitáctiles**, la operación o control es por medio de las yemas de los dedos, como en el caso de los móviles, en uno o varios contactos o también el control cercano a la pantalla, pero sin tocarla.

Recordemos la película **Matrix**

Como comentamos también existe control de sistemas operativos por medio de **la voz humana**, denominado control por reconocimiento del habla o reconocimiento de voz, como por ejemplo Siri, Alexa o Cortana.

Desarrollo de interfaces de usuario

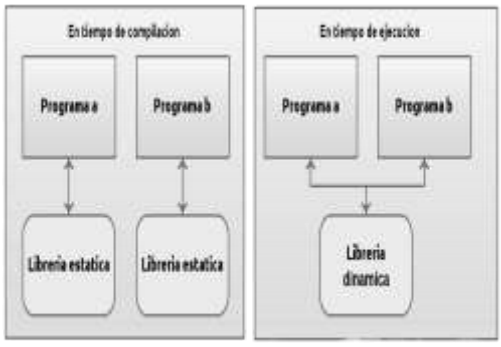
La complejidad en el desarrollo de una interfaz hizo necesario que las diferentes utilidades estuviesen contenidas en diferentes **módulos.** Una **biblioteca** es un **conjunto de implementaciones o subprogramas** para ser invocados y que son utilizados para desarrollar software. Las bibliotecas contienen **código y datos**, que **proporcionan servicios a programas independientes sin relación entre ellos** y que pasan a formar parte de estos de manera que el código y los datos se compartan y puedan modificarse de forma modular. En la ejecución de un programa, ejecutables y bibliotecas hacen **referencias o enlaces** entre sí a través de un software denominado **enlazador.**

A diferencia de un **programa ejecutable,** el comportamiento que implementa una biblioteca no es ser utilizada de forma autónoma, sino que **su fin es ser utilizada por otros programas**, independientes y de forma simultánea, eso sí, sin dejar de ser también un programa. Incluso unas bibliotecas pueden requerir de otras para funcionar, pues a veces es necesario refinar o alterar el comportamiento de la biblioteca original. Por ejemplo, el módulo de impresión en Windows que es común a todos los programas que necesiten imprimir.

El término que utilizamos es el de ***librería,*** mala traducción del concepto inglés *library* que en realidad significa **biblioteca**. La principal ventaja de las bibliotecas o *librerías* es que descarga al desarrollador de una aplicación el de tener que diseñar nuevos módulos.

Un componente de una interfaz de usuario, como puede ser un botón o un panel no deja de ser un elemento gráfico, elemento que debe ser "dibujado" o renderizado en la pantalla por el sistema operativo utilizando para ello las **librerías gráficas.**

Los sistemas operativos modernos **proporcionan bibliotecas que implementan los servicios** del sistema. De esta manera, estos servicios son usados por cualquier programa o aplicación.



Podemos dividir las librerías en dos tipos:

- **Librerías estáticas:** son aquellas que se enlazan en **tiempo de compilación**. La principal ventaja de este tipo de enlace es que hace **que un programa no dependa de ninguna biblioteca** (puesto que las **enlazó al compilar**), haciendo más fácil su distribución al **liberar de dependencias.** Su inconveniente es que los programas son **más pesados y menos flexibles** a la hora de modificar su código porque exige volver a compilar. El enlace estático da como resultado, un archivo ejecutable con todos los símbolos y módulos respectivos incluidos en dicho archivo
- **Librerías dinámicas**: son aquellas enlazadas **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 (/lib en Linux o C:\Windows),** 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 en tiempo de ejecución**. Esto ocasiona algunos problemas de dependencias, principalmente entre diferentes versiones de una misma biblioteca. En Windows las librerías dinámicas se denominan **DLL (Dinamic-Link Library).**
  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 ejecutar.

Las librerías que mayor interés para este módulo profesional son las **librerías gráficas,** de las cuales podemos citar:

Para aplicaciones web:

- **React**: React es una biblioteca de JavaScript muy popular para la construcción de interfaces de usuario. Tiene una gran comunidad y una amplia gama de bibliotecas complementarias, como Material-UI, Ant Design y Bootstrap, que te proporcionan componentes y estilos predefinidos.

- **Vue.js:** Vue.js es otro framework de JavaScript que facilita la creación de interfaces de usuario interactivas. Tiene su propio ecosistema de librerías como Vuetify y Element para construir UIs atractivas.
- **Angular:** Angular es un framework de JavaScript desarrollado por Google. Viene con su propia librería de componentes llamada Angular Material que te permite construir interfaces de usuario con un diseño coherente.
- **Ember.js:** Ember.js es otro framework de JavaScript que ofrece Ember Bootstrap y Ember Material como librerías de UI.
- **Tailwind CSS:** Tailwind CSS es un framework de CSS altamente personalizable que te permite diseñar interfaces de usuario de manera rápida y eficiente utilizando clases predefinidas.

Para aplicaciones móviles:

1. **Flutter:** Flutter es un framework de desarrollo de aplicaciones móviles de código abierto creado por Google. Te permite crear aplicaciones para Android y iOS con un solo código base. Tiene su propio conjunto de widgets personalizables para diseñar interfaces de usuario.
2. **React Native:** React Native es una biblioteca de JavaScript para construir aplicaciones móviles nativas tanto para Android como para iOS. Puedes utilizar librerías como React Native Paper y NativeBase para crear interfaces de usuario elegantes.

Para aplicaciones de escritorio:

1. **Electron:** Electron es un framework que permite crear aplicaciones de escritorio multiplataforma utilizando HTML, CSS y JavaScript. Puedes utilizar librerías de UI como Material-UI y Photon para diseñar la interfaz de usuario.
2. **JavaFX:** Si estás desarrollando aplicaciones de escritorio en Java, JavaFX es una librería gráfica que proporciona componentes y herramientas para crear interfaces de usuario modernas.
3. **GTK:** El Toolkit de GIMP (GTK) es una librería de desarrollo de interfaz de usuario utilizada para aplicaciones de Linux y Unix. GTK+ y su binding para varios lenguajes, como PyGTK y Gtk# (C#), son opciones populares.
4. **PyQT:** es una biblioteca de Python que te permite crear aplicaciones de escritorio con interfaces de usuario gráficas utilizando el framework Qt. Qt es una biblioteca de C++

ampliamente utilizada para el desarrollo de aplicaciones de escritorio, móviles y embebidas. PyQt proporciona enlaces entre Python y Qt, lo que te permite aprovechar toda la potencia y flexibilidad de Qt para crear interfaces de usuario personalizadas y ricas en características en Python.

Entornos de programación: IDE

Un entorno de **desarrollo integrado o entorno de desarrollo interactivo,** en inglés *Integrated Development Environment (IDE),* es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

Normalmente, consiste en un **editor de código fuente**, **herramientas de construcción automáticas y un depurador**. La mayoría de los IDE tienen **auto-completado inteligente** de código ***(IntelliSense).*** Algunos contienen un compilador, un intérprete, o ambos, tales como NetBeans, Eclipse o Visual Net.

Muchas veces, a los efectos de simplificar la construcción de la interfaz gráfica de usuario (GUI) se integran un **sistema controlador de versión** (GIT) y otras varias herramientas. Muchos IDE modernos también cuentan con un navegador de clases, un buscador de objetos y un diagrama de jerarquía de clases, para su uso con el desarrollo de software orientado a objetos.

La integración de todos estos procesos de desarrollo hace posible mejorar la productividad. Por ejemplo, el código puede ser continuamente reprogramado, mientras es editado, previendo retroalimentación instantánea, como cuando hay errores de sintaxis.

Algunos IDE están dedicados específicamente a un lenguaje de programación, permitiendo que las características sean lo más cercanas al paradigma de programación de dicho lenguaje. Por otro lado, existen muchos IDE de múltiples lenguajes tales como Eclipse, ActiveState Komodo, IntelliJ IDEA, MyEclipse, Oracle JDeveloper, NetBeans, Qt Creator, Codenvy y Microsoft Visual Studio, entre otros.

Con el advenimiento de la computación en nube, algunos IDE están disponibles en línea y se ejecutan dentro de los navegadores web.

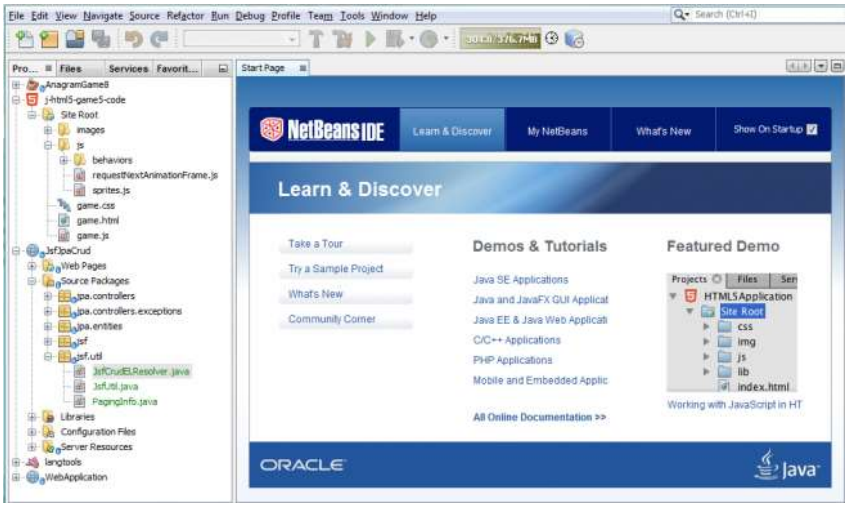
Algunos de los IDES de mayor implantación son:

- **Eclipse** es uno de los entornos más conocidos y utilizados por los programadores, ya que se trata de un entorno de programación de **código abierto y multiplataforma.** Está soportado por una comunidad de usuarios lo que hace que tenga muchos *plugins* de modo que hacen que nos sirva para casi cualquier lenguaje, en este aspecto es de lo mejores. Sirve para Java, C++, PHP, Perl, Python y un largo etcétera. También nos permite realizar aplicaciones de escritorio y aplicaciones web por lo que nos brinda una gran versatilidad.

Además, está en constante evolución y hay muchos tutoriales por la red que nos guían en su instalación y utilización.

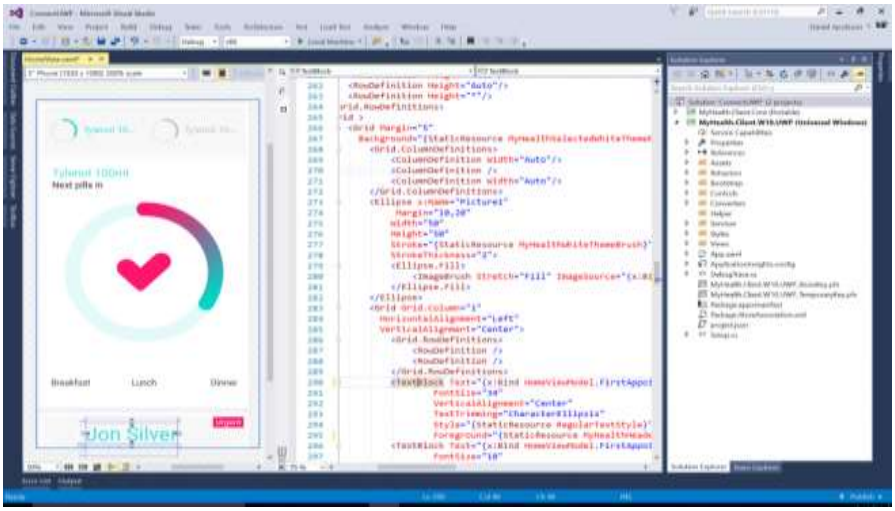
- **Netbeans** también es un entorno de programación muy utilizado por los programadores. Se trata de otro entorno multilenguaje y multiplataforma en el cual podemos desarrollar software de calidad. Con él podemos crear aplicaciones web y de escritorio, además de contar con plugins para trabajar en Android.

  El lenguaje que mejor soporta es Java, ya que fue creado por Oracle y su creación fue para ser el IDE de Java. Aunque como hemos dicho, es multilenguaje debido a que soporta JavaScript, HTML5, PHP, C/C++ y otros.



- **Visual Studio** fue diseñado por Microsoft y es uno de los mejores entornos de programación actualmente soporta todo tipo de lenguajes.

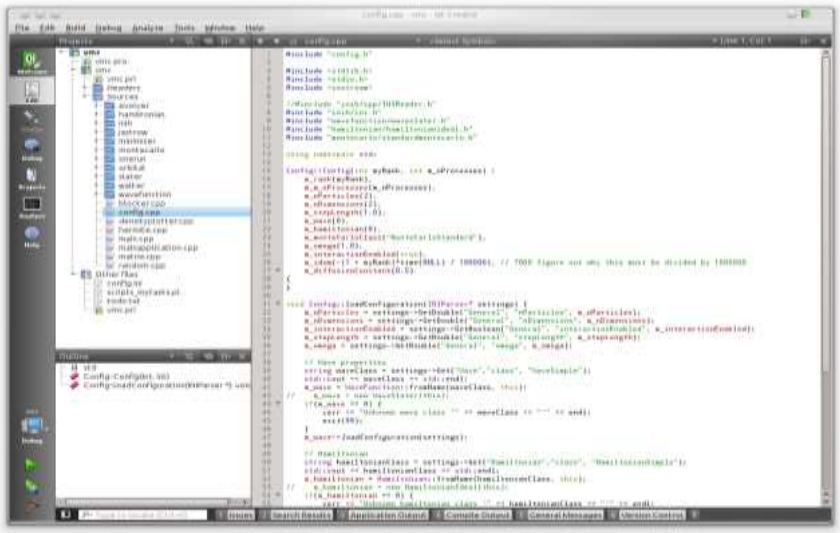
  Microsoft ha creado también un *Visual Studio Comunity* que es muy parecido al Visual Studio de pago, sólo que este está soportado por la comunidad. Este entorno nos permite hacer aplicaciones web y de escritorio y ayuda mucho al programador.



- **JetBrain** no es un entorno concreto, es una compañía que crea entornos de programación, es software libre y desarrollan entornos para multitud de lenguajes como son Java, Ruby, Python, PHP, SQL, Objective-C, C++ y JavaScript. También están desarrollando IDE's para C# y GO. Carece de diseñador de interfaces gráficas.

  Durante la parte práctica del curso con Python que es el **Pycharm**, y ha facilitado mucho su programación por tratarse una herramienta muy completa.

- **QtCreator** es un entorno de programación para C++ usan el framework de QT, es un entorno amigable. También es un entorno multiplataforma programado en C++, JavaScript y QML. Este IDE está diseñado específicamente para utilizar el framework de QT, que por otra parte es un muy interesante ya que nos permite hacer aplicaciones multiplataforma de una manera sencilla y rápida incluyendo diseñador gráfico, **Qt- Designer.**



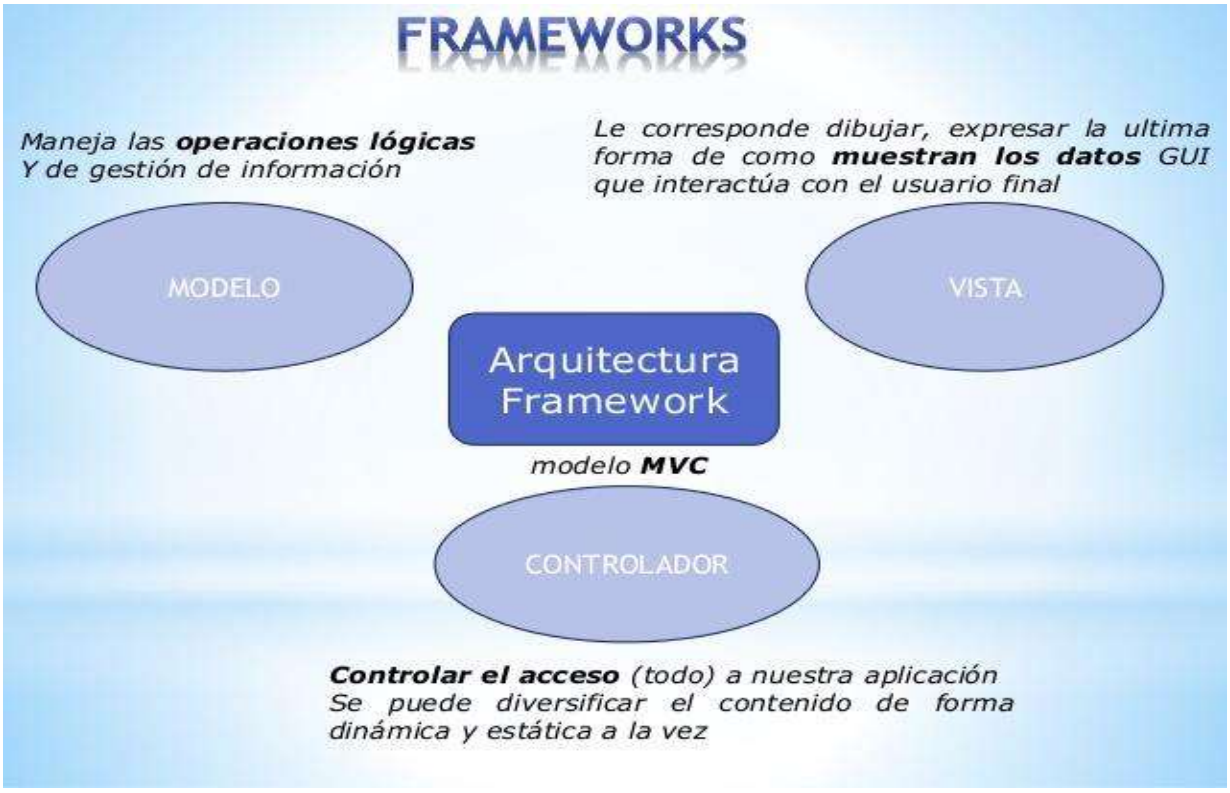
Existe bastantes más entornos de desarrollo, pero los citados quizás son los más influyentes que hay hoy en día. Por mencionar otros, **Aptana Studio, Gambas, Mono Develop, Anjuta, Android Studio.** También existen los denominados IDE-Editores de texto, que como su nombre indica no son más que un editor de texto y compilador: **Sublime-Text, Notepad++, Vi o nano.**

Framework

Un concepto importante y más amplio son los **framework** herramienta que trabaja con la arquitectura basada en la metodología **modelo, vista y controlador.**

Un **framework** es una estructura conceptual y tecnológica de soporte definido, normalmente con módulos de software concretos, que puede servir de base para la organización y desarrollo de software. Básicamente, es un marcho de trabajo con un conjunto estandarizado de **conceptos, prácticas y criterios** para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. Puede incluir soporte de **programas, bibliotecas, y un lenguaje interpretado**, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto.

En el desarrollo web, donde los framework tienen su mayor influencia, es hoy en día la arquitectura más utilizada por la enorme productividad que puede alcanzarse en los proyectos a realizar.


FRAMEWORKS
Maneja las operaciones lógicas
Y de gestión de información
Le corresponde dibujar, expresar la ultima forma de como muestran los datos GUI que interactúa con el usuario final
MODELO
VISTA
Arquitectura Framework
modelo MVC
CONTROLADOR
Controlar el acceso (todo) a nuestra aplicación
Se puede diversificar el contenido de forma dinámica y estática a la vez


FRAMEWORKS WEB MAS NOTABLES
PHP
laravel
yii framework
akePHP
Code Igniter
ZEND FRAMEWORK
Symfony

Cada uno de ellos con sus ventajas e inconvenientes.

Interfaz gráfica. Características y componentes

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

Existen una serie **principios generales** basados en los principios formulados por Jakob Nielsen, uno de los expertos más importantes en **usabilidad**, que deben acompañar al diseño e implementación de Interfaces de Usuario (IU), ya sea para las IU gráficas de Escritorio, como para la Web y que son:

- **Sencilla**
- **Intuitiva**
- **Coherente**
- **Clara**
- **Predecible**
- **Flexible**
- **Consistente**

Dentro del diseño de la interacción usuario-computador en la actualidad se tienen en cuenta una serie de **diferentes disciplinas** que van desde la psicología, filosofía, ciencia cognitiva, ergonomía, ingeniería, sociología, antropología hasta la lingüística.

Los elementos básicos de una interfaz gráfica son:

- **Componentes GUI (*widgets*)** y que a su vez se componen:
    - Objetos visuales del interfaz
    - Un programa gráfico es un 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. Este elemento es fundamental. Los *layout managers* gestionan la organización de los componentes gráficos de la interfaz.

Dependiendo de cómo sea la interfaz que queramos realizar tendremos la posibilidad de utilizar un tipo u otro, o combinar varios elementos entre sí para poder aprovechar la funcionalidad de todos ellos. Tomando como base el diseño de IU en *Android* los principales layouts son:

- ***Linear Layout*:** Como su nombre indica maneja el espacio de *forma lineal, bien sea en horizontal o en vertical.* Es decir, coloca todos aquellos elementos que están situados dentro del mismo de izquierda a derecha o de arriba abajo.
- ***Relative Layout***: se trata de un gestor del espacio que coloca los elementos con respecto al superior.
- ***Frame Layout:*** es un gestor del espacio que nos permite superponer capas una encima de otra con la posibilidad de mostrar todas al mismo tiempo o de mostrar una sola.
- ***TableLayout*** es una especialización de *Linear Layout*. Al igual que pasa en HTML, un *Table Layout* necesita la declaración de un Table Row para poder añadir una fila en la cual se situarán las vistas concretas, las cuales harán las veces de columnas.

- **Eventos**: son los encargados de la **interactividad**, respuesta a la entrada del usuario, aunque no solo. Los eventos son todas las acciones que el usuario o un sistema inicia, dar clic sobre un botón, presionar las teclas del teclado, tick de reloj, pero la mayoría se basan en unas clases auxiliares llamadas ***escuchadores (listeners)*** que reciben eventos específicos y ejecutan el código correspondiente.

  Tipos de eventos más destacados:

  - **Eventos de Ventana**: Son los que se generan en respuesta a los cambios de una ventana, un frame o un dialogo.
    - WINDOW_DESTROY cerrar
    - WINDOW_EXPOSE: abrir ventana
    - WINDOW_ICONIFY minimizar
    - WINDOW_DEICONIFY restaurar la ventana
    - WINDOW_MOVED: mover la ventana

- **Eventos de Teclado:** Son generados en respuesta a cuando el usuario pulsa y suelta una tecla mientras un Componente tiene el foco de entrada, el nombre depende el lenguaje de programación utilizado.
    - KEY_PRESS: evento al pulsar
    - KEY_RELEASE: similar al último
    - KEY_ACTION evento al pulsar una tecla
    - KEY_ACTION_RELEASE evento al liberar una tecla
- **Eventos de Ratón**: Son los eventos generados por acciones sobre el ratón dentro de los límites de un Componente.
    - MOUSE_DOWN
    - MOUSE_UP
    - MOUSE_MOVE
    - MOUSE_ENTER
    - MOUSE_EXIT
    - MOUSE_DRAG: pulsar y no soltar
- **Eventos de Barras**: Son los eventos generados como respuesta a la manipulación de barras de desplazamiento (*scrollbars).*
    - SCROLL_LINE_UP
    - SCROLL_LINE_DOWN
    - SCROLL_PAGE_UP
    - SCROLL_PAGE_DOWN
    - SCROLL_ABSOLUTE
- **Eventos de Lista.** Son los eventos generados al seleccionar elementos de una lista.
    - LIST_SELECT

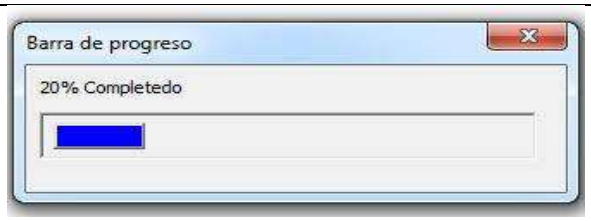
- LIST_DESELECT

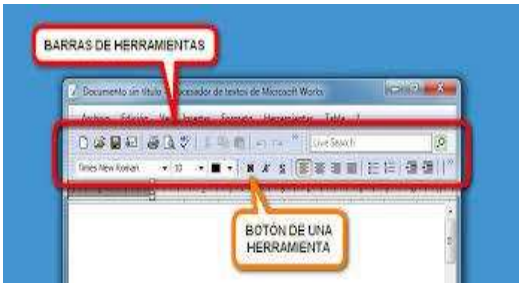
- **Eventos Varios.** Son los eventos generados en función de diversas acciones.
  - ACTION_EVENT
  - LOAD_FILE
  - SAVE_FILE
  - GOT_FOCUS
  - LOST_FOCUS

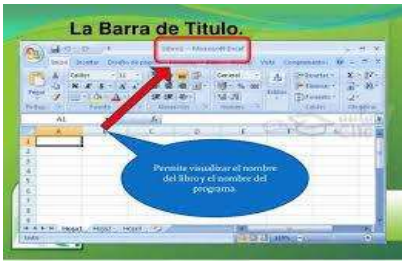
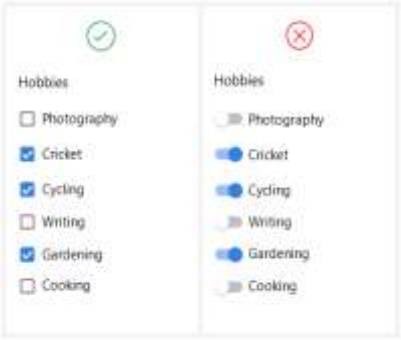
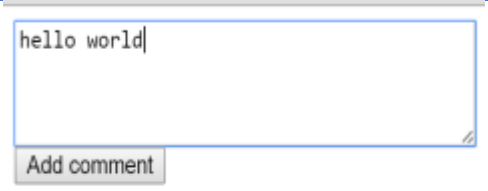
- **Errores:** Los errores en la programación suelen ser comunes. En algún momento y por alguna causa, podría generarse algún conflicto en el programa: no hay conexión a Internet, usuario o contraseña incorrecta, valores alfanuméricos en campos solo para números, o que el código del programa no está escrito correctamente etc**. Existen formas para detectar y corregir errores por medio de opciones llamadas excepciones.**

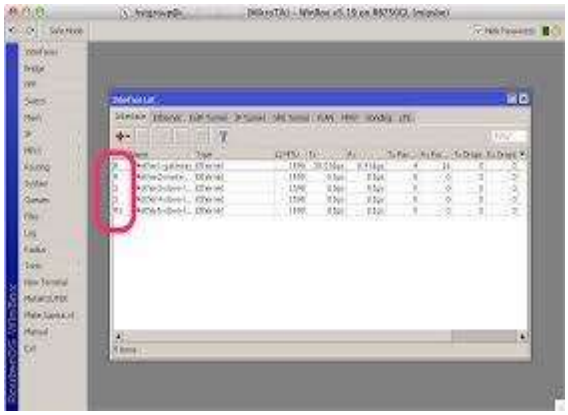
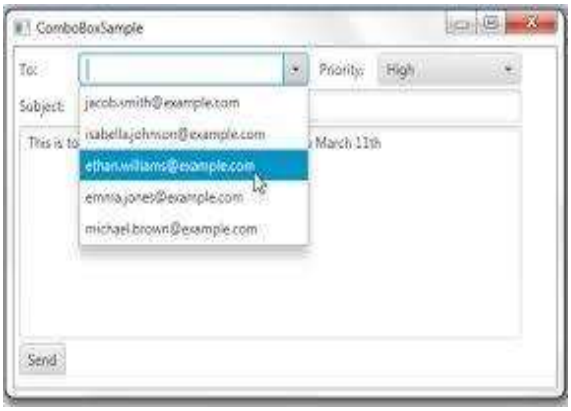
- **Excepciones** están destinadas en todos los lenguajes de programación que las soportan, para la detección y corrección de errores. Si hay un error, la aplicación no debería morirse y generar un ***core***. Se debería lanzar ***(throw)*** una excepción que se captura mediante un ***(catch)*** y resolve la situación de error.

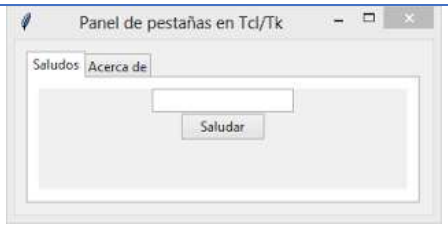
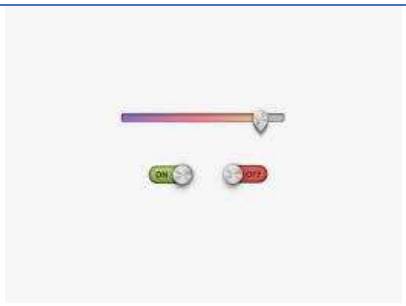
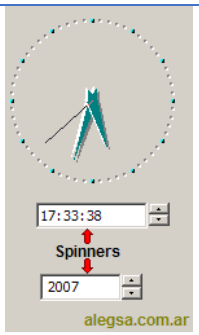
Los elementos típicos, que son más que los que se mencionan aquí, **de una interfaz gráfica son**:

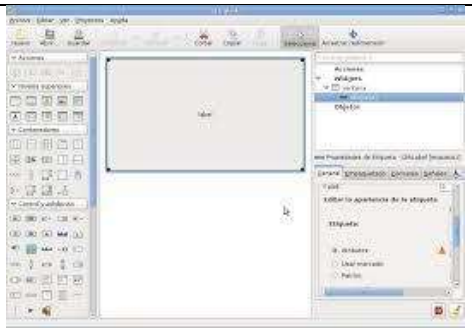
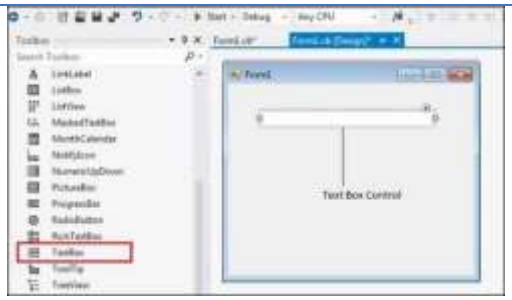
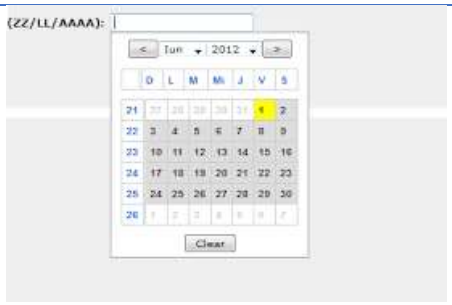
Las **barras de progreso** permiten mostrar de forma gráfica el estado de avance de una tarea o proceso.	
Los **botones de opción** o *radiobutton* permiten elegir **sólo una opción** de un conjunto predefinido de opciones.	

Las **barras de desplazamientos** constan de una barra horizontal o vertical con dos extremos con flechas en sentidos contrarios y ubicadas en los extremos de una ventana o recuadro. Permiten desplazar el contenido del cuadro hacia ambos lados. Suelen aparecer o activarse cuando el recuadro **no es lo suficientemente grande** como para ver todo su contenido.	
La **barra de estado** o *statusbar* permite mostrar información acerca del estado actual de la ventana. Generalmente las barras de estado se ubican en la parte inferior de las ventanas.	
La **barra de herramientas** o *toolbar* es una agrupación de íconos con los cuales es posible acceder a determinadas herramientas o funciones en una aplicación.	

La **barra de título** o *titlebar*se encuentra en la parte más superior de una ventana, donde aparece un título que se corresponde con el contenido de esta.	
Los **botones** son tipos de widget que permiten al usuario comenzar un evento, como buscar, aceptar una tarea, interactuar con un cuadro de diálogo, etc.	
Las **casillas de verificación** o *checkbox* permite al usuario marcar **múltiples selecciones** de un número de opciones.	
Un **cuadro de diálogo** o *dialogbox* es una ventana especial para mostrar información al usuario o para obtener de éste una respuesta.	
Un **cuadro de texto o** *textarea* es un elemento típico en las interfaces gráficas en donde es posible insertar texto.	

Los **íconos del sistema** son aquellos íconos que se utilizan para identificar archivos, accesos directos, programas y herramientas del sistema operativo.	
Una **lista** o *list* es una relacion de datos, ordenados o clasificados segun su tipo. Una evolución de ella que veremos a menudo en la parte práctica es el *treeview* como vemos en la imagen.	
Una **lista desplegable** o *combobox* al usuario escribir sobre un campo de texto mientras se muestra una lista de opciones.	
Un **menú** es una herramienta gráfica en la interfaz de páginas web y aplicaciones que consiste en una lista de opciones que puede desplegarse para mostrar más opciones o funciones.	

Un **menú contextual** es tipo de menú que se adaptará al contexto desde donde se accede. El menú contextual se accedo con el clic derecho sobre algún elemento en la pantalla.	
El **panel de pestañas** o *panel* es un elemento que se encuentra en las interfaces gráficas, que permite cambiar entre distintos documentos o secciones de forma rápida.	
Una **slider** es un elemento de las interfaces gráficas que permiten seleccionar un valor moviendo un indicador o, en algunos casos, el usuario puede hacer clic sobre algún punto de la slider para cambiar hacia ese valor.	
Un **spinner** es un elemento de las interfaces gráficas que permite al usuario ajustar un valor dentro de un cuadro de texto adjunto a dos flechas que apuntan en direcciones opuestas.	
Un **tooltip** es un elemento de las interfaces gráficas que se emplea junto con el cursor del ratón.	

La **ventana** es la parte delimitada de la pantalla en un sistema operativo gráfico que suele ser rectangular y que contiene elementos afines entre sí en ella.	
El **contenedor** es un elemento a modo rejilla o *grid* que facilita el movimiento de diferentes *widgets* de formar unitaria.	
La **etiqueta** o *label* permite incluir mensajes cortos y de carácter informativo cerca de otros widgets	
Las **cajas de texto** o *textbox* permiten la introducción de datos que luego se van a procesar	
El **calendario** o *calendar* como su nombre indica permite introducir o examinar fechas.	

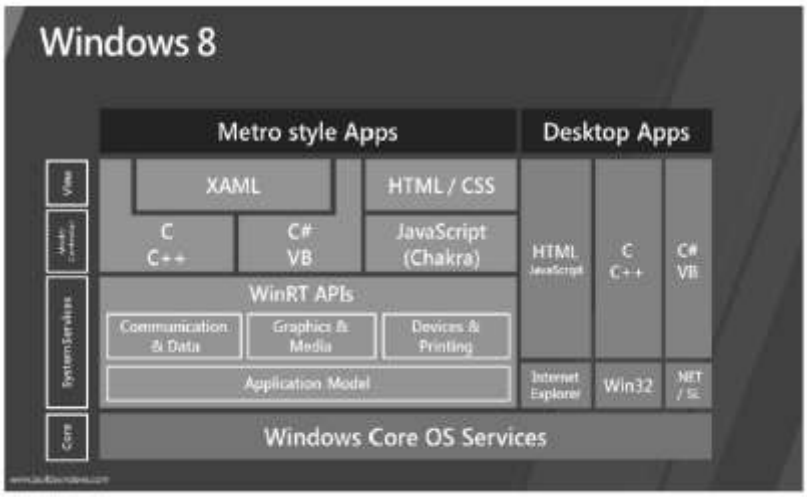
Existen más elementos en una interfaz gráfica como las imágenes, menús laterales, cuadros de diálogo con funciones definidas, pero estos son una muestra bastante representativa de los más usuales.

XML e Interfaces gráficas

Los elementos gráficos que están presentes en las interfaces, que además de los controles contienen aspectos como colores, degradados, fondos, figuras y otros, es posible representarlos con los mismos lenguajes de programación utilizados para definir los objetos entrada/salida, aunque usando librerías gráficas específicas. Citaremos solo dos, uno para formateado de páginas y otro para imágenes.

1. XAML

Es el **lenguaje declarativo de formato interpretado** para la interfaz de usuario para la **Base de Presentación de Windows** (WPF por sus siglas en inglés), Blend y Silverlight, el cual es uno de los "pilares" de la interfaz gráfica de **programación de aplicaciones .NET.**



XAML fue diseñado para soportar las clases y métodos de la **plataforma de desarrollo .NET** que tienen relación con la interacción con el usuario, en especial el despliegue en pantalla. El acrónimo XAML originalmente significaba *Extensible Avalon Markup Language*, Lenguaje Extensible de Formato de Avalon; habiendo sido Avalon el nombre clave original de la Base de Presentación de Windows, nombre que engloba a este grupo de clases de .NET.

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

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

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.

La unidad básica de trabajo con XAML son las **páginas.** En ellas se encuentran los **controles visuales** (botones, etiquetas…) que se refieren a ellos como **elementos.** Internamente los controles que tienen una **apariencia visual** heredan dos clases ***UIElement y Framework Element*** que aportan características de *layout* o disposición y sus **propiedades y atributos** (color, tamaño…)

Crear un control en XAML es tán facil como escribir su nombre rodeado de los signos Mayor y Menor. Un botón con algunas propiedades se vería de la siguiente forma:

```
<Button>
<Button.FontWeight>Bold</Button.FontWeight>
<Button.Content>
<WrapPanel>
<TextBlock Foreground="Blue">Multi</TextBlock>
<TextBlock Foreground="Red">Color</TextBlock>
<TextBlock>Button</TextBlock>
</WrapPanel>
</Button.Content>
</Button>
```

La mayoría de los **marcos (frameworks) de UI modernos al igual que WPF,** están impulsados por **eventos** que se enlazan a los **elementos** mediante los ***bindings.*** Todos los **controles, incluida la ventana (*Window)*** (que también hereda la clase control) exponen un rango de eventos a los que puede suscribirse.

```
<Window x:Class="WpfTutorialSamples.XAML.EventsSample"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="EventsSample" Height="300" Width="300">
<Grid Name="pnlMainGrid" MouseUp="pnlMainGrid_MouseUp" Background="LightBlue">
```

</Grid>

</Window>

La codificación del evento en cuestión sería:

private void pnlMainGrid_MouseUp(object sender, MouseButtonEventArgs e)

{

MessageBox.Show("You clicked me at " + e.GetPosition(this).ToString());

}

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window
    id="sample-window"
    title="Beispiel"
    xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
      <vbox>
          <checkbox label="CheckBox"/>
          <hbox>
              <spacer flex="1"/>
              <button label="OK"/>
              <button label="Abbrechen"/>
          </hbox>
      </vbox>
</window>
```

Darstellung im Mozilla Firefox

2. SVG

Es un formato de **gráficos vectoriales bidimensionales**, tanto estáticos como animados, en formato **XML,** cuya especificación es un estándar abierto desarrollado por el W3C desde el año 1999.

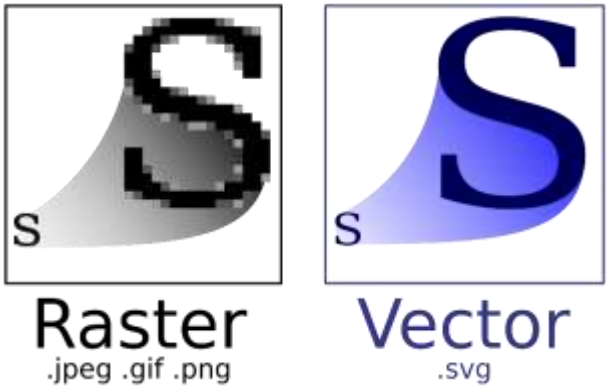
Una **imagen vectorial** es una imagen digital formada **por objetos geométricos dependientes** (segmentos, polígonos, arcos, muros, etc.), cada uno de ellos definido por **atributos matemáticos** de forma, de posición, etc. Por ejemplo, un círculo de color rojo quedaría definido por la posición de su centro, su radio, el grosor de línea y su color.

La especificación de SVG permite tres tipos de objetos gráficos:

- **Elementos geométricos vectoriales**, como los caminos o trayectorias consistentes en rectas y curvas, y áreas limitadas por ellos.
- **Imágenes de mapa de bits**/digitales.
- **Texto**.

El dibujado de los SVG puede ser **dinámico e interactivo.** El modelo **Document Object Model (DOM**) para SVG, que incluye el DOM XML completo, permite animaciones de gráficos vectoriales sencillas y eficientes mediante ECMAScript o SMIL. Un conjunto amplio de manejadores de eventos, como "*onMouseOver" y "onClick*", puede ser asignado a cualquier objeto SVG.

Si el espacio de almacenamiento es un problema, las imágenes SVG pueden ser guardadas como archivos comprimidos con ***gzip***, en cuyo caso pasan a ser imágenes SVGZ.



Ejemplo de código:

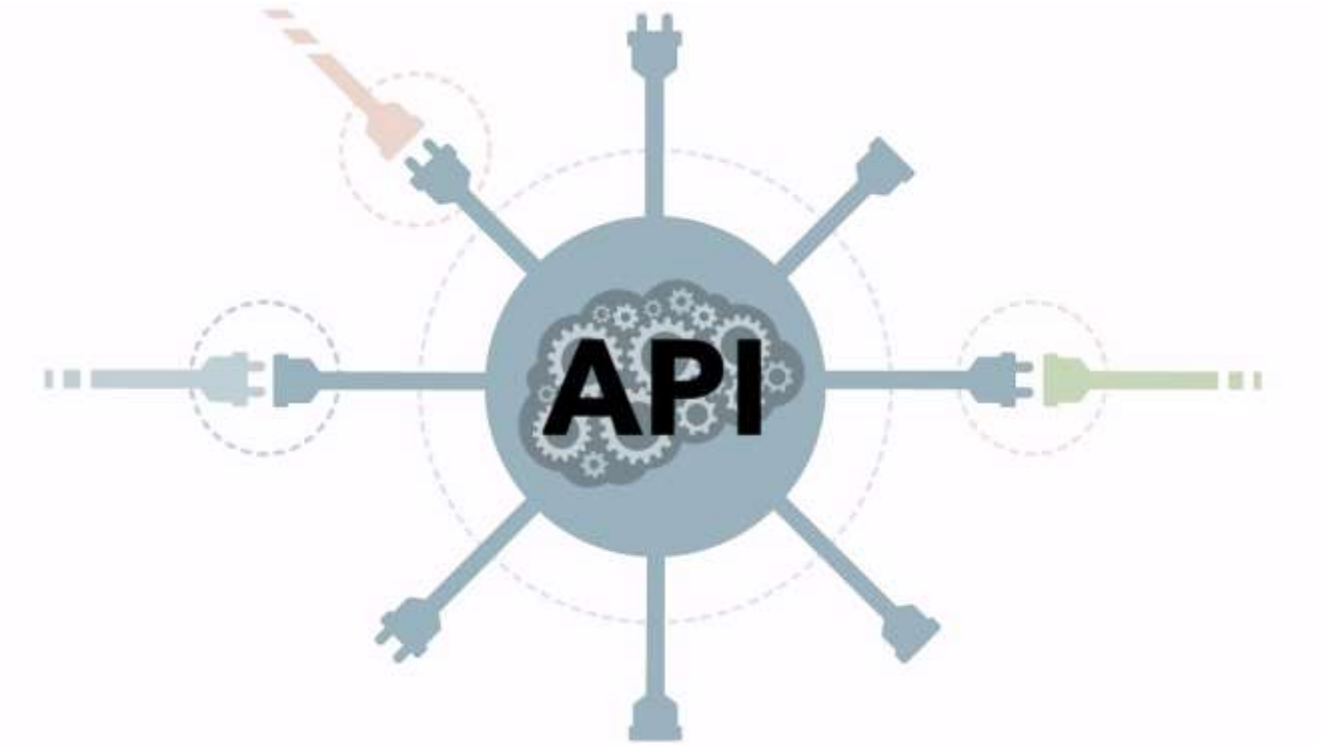
```
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
        <g transform="translate(50,150)">
        <g transform="scale(1,-1)">
                ...
        </g>
        </g>
</svg>
```

Actualmente los principales obstáculos al uso generalizado de SVG siguen siendo que las implementaciones en los navegadores todavía son incompletas y los potenciales riesgos de seguridad (debido a que las imágenes SVG pueden contener código Javascript, por ello se recomienda no incorporar imágenes SVG sin haber comprobado antes su contenido).

Acceso a datos

El **acceso a datos** es una de las principales tareas en el desarrollo de aplicaciones informáticas y que contempla la visualización, adición, modificación o eliminación de datos. Para llevar a cabo estas tareas desde una interfaz gráfica, además de los *widgets* correspondientes (botones, cajas de texto, treeview…) es necesario llevar a la conexión de la aplicación con las bases de datos para la ejecución de los eventos correspondientes.

Una **interfaz de programación de aplicaciones (API)** es un conjunto de herramientas, definiciones y protocolos que se usa para diseñar e integrar software de aplicaciones. Permite que su producto o servicio se comunique con otros productos y servicios, en nuestro caso con una base de datos, **sin la necesidad de saber cómo se implementan**. Las API simplifican el desarrollo de las aplicaciones, lo cual permite ahorrar tiempo a los desarrolladores y dinero a las empresas. Cuando diseña herramientas y productos nuevos (o maneja otros actuales), las API le **otorgan flexibilidad; simplifican el diseño, la administración y el uso**.



Algunas de las API para base de datos:

- **Java Database Connectivity** más conocida por sus siglas JDBC es una API que permite la ejecución de operaciones sobre bases de datos desde el **lenguaje de programación Java**, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

- **Open DataBase Connectivity (ODBC)** es un estándar de acceso a las bases de datos desarrollado por SQL Access Group (SAG) en 1992. El objetivo de ODBC es hacer posible el acceder a cualquier dato desde cualquier aplicación, sin importar qué sistema de gestión de **bases de datos (DBMS)** almacene los datos. ODBC logra esto al insertar una **capa intermedia denominada nivel de Interfaz de Cliente SQL (CLI), entre la aplicación y el DBMS**. El propósito de esta capa es traducir las consultas de datos de la aplicación en comandos que el DBMS entienda. JDBC derivó de esta.
- **ADO.NET** es un conjunto de componentes del software que pueden ser usados por los programadores para acceder a datos y a servicios de datos. Es parte de la biblioteca de clases base que están incluidas en el **Microsoft .NET Framework.** Usada por los programadores para acceder y para modificar los datos almacenados en un Sistema Gestor de Bases de Datos Relacionales DBMS, aunque también puede ser usado para acceder a datos en fuentes no relacionales.

Hay muchas más, sobre todo en el campo del mundo web com API REST, JSon… pero su conocimiento escapa a los objetivos del módulo.

Usabilidad

La **Usabilidad** es la medida de la **calidad de la experiencia** que tiene un **usuario** cuando interactúa con un **producto o sistema.** Esto se mide a través del estudio de la relación que se produce entre las herramientas (entendidas como los controles y funcionalidades de una interfaz gráfica) y quienes las utilizan, para determinar la eficiencia en el uso de los diferentes elementos ofrecidos en las pantallas y la efectividad en el cumplimiento de las tareas que se pueden llevar a cabo a través de ellas.

Jakob Nielsen, uno de los mayores expertos de la usabilidad de las interfaces gráficas, abstrajo **diez principios que nos permitirán crear productos.** Hay que decir que no haremos distinción entre las interfaces gráficas Web y las de las aplicaciones en entornos de escritorio o bien móviles, a fin de cuentas, todas persiguen lo mismo con las mismas herramientas y con pequeñas diferencias.

- **Visibilidad del estado del sistema**

El sistema **mantiene informado al usuario** en todo momento sobre el estado actual del sistema, apoyándose en indicadores que sirvan de *feedback* y que sean de fácil lectura.

1. **Correspondencia entre el sistema y el mundo real**

Debemos de investigar un poco al público al que irá dirigido nuestro sistema para **utilizar lenguaje que le resulte familiar**. Mostrar información en un **orden natural y lógico**, usar representaciones gráficas claras y seguir convenciones que faciliten esta homogeneidad. Los árabes, por ejemplo, leen de derecha a izquierda.

2. **Libertad y control del usuario**

Los usuarios se sentirán más cómodos en tu sistema si les otorgas **libertades y control en su uso**. Debemos estar conscientes que es muy probable que los usuarios elijan opciones que posteriormente quieran cambiar, y permitirles deshacer/rehacer acciones o cancelar ciertas elecciones es una manera de darles el control.

3. **Prevención de errores**

Todos podemos equivocarnos, por ello debemos procurar que el **impacto de estos errores sea mínimo** en el sistema. **Realizar pruebas sobre casos pocos probables y probar nuestros algoritmos** nos ayudarán a que el sistema pueda reaccionar.

4. **Coherencia y estándares**

Los usuarios deben experimentar un sistema homologado, en donde el **lenguaje, uso de colores y elementos gráficos sea consistente.**

5. **Reconocimiento en vez de recordar**

Minimiza la información que el usuario tiene que recordar mostrando objetos, acciones y opciones de una manera visible en tu interfaz. Permite que vea las instrucciones en todo momento y que, al elegir opciones, estás sean mostradas explícitamente. El texto de un botón que pone Salir… debe hacer eso salir.

6. **Flexibilidad y eficiencia de uso**

Permite que **los usuarios experimentados puedan tener comandos "aceleradores"** para que el uso de tu sistema sea cada vez más cómodo. Estás opciones en un principio no se mostrarán a los nuevos usuarios, pero con el uso, harán que su trabajo sea más productivo. Los atajos de teclado son un ejemplo.

7. **Diseño estético y minimalista**

Como regla de oro: **muestra sólo la información que se necesite en el momento.** Recuerda que es muy fácil perder la atención de los usuarios, por tal motivo debemos ser claros y concisos. En la actualidad existen diferentes paradigmas de diseño que tienen reglas claras para hacer más estético y funcional un sistema sin invertir tanto en diseño.

8. **Ayuda y documentación**

Aun cuando nuestro sistema sea lo suficientemente intuitivo para usarse, **siempre es conveniente otorgar documentación y proveer de una forma de otorgar ayuda** a los usuarios. Toda la información mostrada en FAQs y documentación, así como las respuestas a las dudas de usuarios, deberán ser claras y directas.

9. **Ayuda a los usuarios a reconocer, diagnosticar y recuperarse de errores**

Debemos **mostrar mensajes con la causa específica del error** y con indicaciones de como solventarlo, cuidar el lenguaje utilizado y ser precisos en las acciones siguientes. Mostrar **mensajes genéricos** confunden a los usuarios y hace que no sepan cómo reaccionar.

Con respecto a la usabilidad existen también d**efiniciones estandarizadas, las más importantes son:**

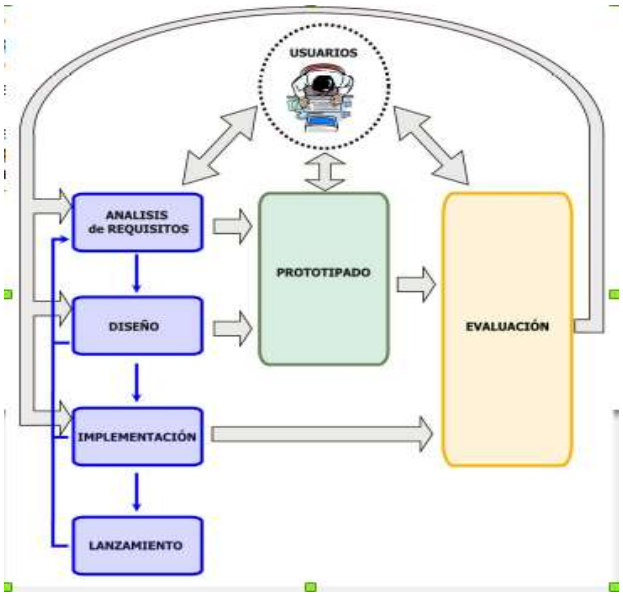
- **ISO/IEC 9126: se** refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso.
  Esta definición **hace énfasis en los atributos internos y externos del producto, los cuales contribuyen a su funcionalidad y eficiencia**. La usabilidad depende no sólo del producto si no también del usuario.
- **ISO/IEC 9241: em este caso se refiere a la eficacia, eficiencia y satisfacción con la que un producto** permite alcanzar objetivos específicos a usuarios específicos en un contexto de uso específico.

IPO

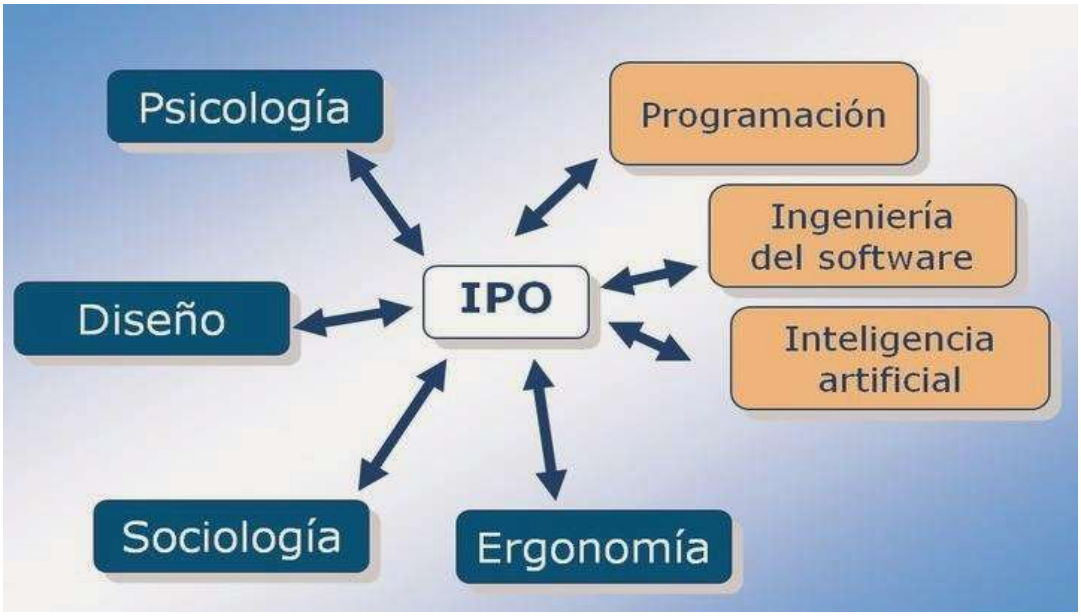
Se denomina **interacción persona-ordenador (IPO)** como la disciplina dedicada al **diseño, evaluación e implementación de sistemas informáticos interactivos para el uso humano**. Otras definiciones incluyen además la influencia de estos diseños sobre las personas u organizaciones.

En este punto introducimos el concepto de **experiencia de usuario (UX)** concepto que se importa del área del marketing y que describe la **relación entre las personas y la tecnología** desde una perspectiva más global y tiene que ser este el principal protagonista del diseño.

Para ello el **prototipado centrado en el usuario** es la técnica de desarrollo, en opinión de muchos, que mayores probabilidades de éxito tiene a la hora de lograr un producto de calidad y adaptado a las necesidades del usuario. Es decir, diseñar, testear, volver a diseñar y así sucesivamente.



Esta definición nos introduce en un contexto nuevo, y es que la interacción del ser humano con el ordenador no es un tema de diseño y desarrollo exclusivo de los profesionales "informáticos" sino que abarca a otras disciplinas y áreas de conocimiento, tal como se refleja en la imagen que se adjunta a continuación.



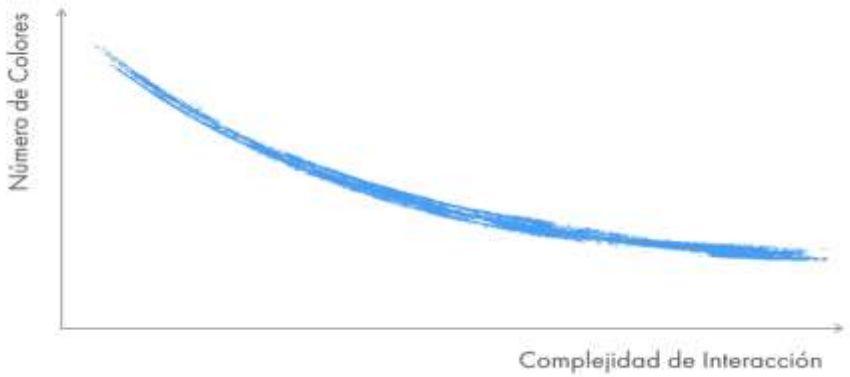
Además, hay gran variedad de dispositivos, desde tablets, pc, portátiles, móviles, interfaces industriales, etc… y que cada uno tiene unas características propias que a su vez dependen de su tamaño, uso, contexto, etc.

En este caso nos centraremos en los dispositivos más usados a nivel empresarial como son el pc y el portátil. El móvil se estudiará en otro módulo, pero muchas de las aseveraciones que aquí se hagan son válidas para el mismo.

Los sentidos **más importantes en IPO** son **vista, oído y tacto,** por este orden. Los conocimientos científicos sobre los mecanismos de percepción tienen aplicación directa al diseño de interfaces.

1. El **color.**

La complejidad de la interacción tiende a ser inversamente proporcional a la cantidad de los colores para reducir la sobrecarga, es decir, si la aplicación es compleja, **conviene reducir el número de colores.**



Lo ideal, según muchos autores, es el uso de **un color principal y dos variaciones** (por ejemplo, Facebook).

Ello permite establecer la **coherencia en la interfaz gráfica** con un mismo color con distintos contrastes.

En la cultura occidental, los colores tienen cierto significado que se puede resumir en la siguiente imagen y que de acuerdo con el fin de la web o de la aplicación conviene que destaquen unos sobre otros.

Evidentemente, esto **no incluye a las imágenes e iconos**. Si aplicamos el principio de la imagen izquierda podemos obtener variaciones equilibradas y que facilitan la interacción con el usuario.

Los grises son fundamentales, sobre todo porque **permiten mantener la atención al usuario**, sobre todo, porque los asocia al texto. A ellos le podemos aplicar la ecuación anterior obteniéndose también resultados interesantes.

Finalmente, hay una **recomendación tradicional nos dice que jamás olvidemos el blanco y evitemos el negro absoluto.** Simplemente porque genera un contraste muy fuerte en las pantallas que cansan la vista. La masiva adopción de “versiones nocturnas” en aplicaciones nos obliga a tener presente paletas invertidas dónde el *background* será negro absoluto, allí tenemos que “evitar” el blanco, lo que a veces da aspectos ilegibles.

Existe una interesante herramienta creada por Google para facilitar la selección de paletas para aplicaciones Android en el siguiente enlace.

Por otro lado, este **enlace** 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 *deutanopia, protanopia y tritanopia*).

2. Las **fuentes.**

Uno de los aspectos importantes es la **escalabilidad,** es decir, que la fuente funcione igual a ser escalada en diferentes tamaños, por ello es necesario realizar pruebas para que ello sea así.

Otra recomendación general es que **no necesitamos más de 2 fuentes** y en la mayoría de los casos con una es suficiente. Algunos diseñadores utilizan **una fuente para los títulos y otras para el cuerpo.**

La **legibilidad** del contenido es nuestra prioridad, en especial cuando nuestro objetivo es la lectura continua. Una regla general es: cuanto más le **exigimos al usuario que lea, más simple y práctica debe ser la fuente**. En cambio, cuanto menos texto ofrecemos, mayor expresividad podemos darle.

Una regla habitual **es la "regla del 8"**, es decir, si tu fuente básica tiene tamaño 10, lo recomendable, si necesitas otros tamaños es ir a 18, 26 y 44.

En la siguiente imagen nos da una idea de cómo debe nuestra carga de texto en función de la legibilidad, es decir, **más texto entonces fuente más sencilla.**

Serif: Estas fuentes tienen pequeños bordes al principio y final de las letras. En su tipografía estándar, estas letras son preferidas para los bloques de texto grandes, puesto que las fuentes de texto Serifs **se hacen más fácil de leer en líneas o párrafos largos.**

Sans-serif - versión imprimible (Pdf): Estas fuentes consisten solamente en linestrokes y por lo tanto son más simples en la forma (e.g., Helvetica, Arial, Futura). Habitual en textos cortos y títulos.

Otro factor importante al elegir una fuente es **la cantidad de *Pesos***: *Light, Regular, Bold, Black son* algunos de los pesos que encontramos. Una fuente con suficientes pesos será mucho más versátil y nos facilitará el trabajo a la hora de jerarquizar. *Fira* de Firefox y *Roboto* de Google son excelentes ejemplos de fuentes versátiles y optimizadas para pantallas.

En cuanto al **interlineado,** distancia entre líneas, es recomendable utilizar **alrededor de 1,5,** que representa más o menos, una vez y media el tamaño de la fuente, pero refiriéndonos al cuerpo.

El **ancho de línea** es el tamaño del contenedor del texto y se mide en palabras por línea o caracteres por línea. Anchos muy grandes nos hacen perder la línea siguiente y anchos muy pequeños nos interrumpen demasiado la lectura. **Entre 50 y 70 caracteres por línea es el ideal.**

3. Los **iconos**

Los **iconos** son un elemento muy utilizado en el diseño de los botones. Los botones no solo son un elemento común en cualquier diseño interactivo (como por ejemplo en una web, una app o en un panel informativo), son fundamentales ya que de ellos depende la interacción y experiencia de usuario.

Las indicaciones de un botón **deben ser claras.** Bien mediante **el uso de un icono o de un verbo en infinitivo** que represente sin confusión. Si debemos seguir unos estándares ya definidos a nivel de forma, también deberíamos hacerlo a nivel de **localización y orden**. Por otro lado, no es lo mismo un botón en un ordenador que en una *app,* y dentro de estas el tamaño del dispositivo. Por ello, es necesario crear los ***botones responsive*** o personalizados y ajustar su tamaño a unas medidas mínimas en función del dispositivo, así como su separación con respecto a otros.

Una estrategia es el ***Call to Action,*** es decir, llamar la atención. Por ejemplo, si quieres que destaque el botón *Confirmar,* suele ponerse de un color diferente al resto. Esto incluso se utiliza en otro tipo de *widgets* como las cajas de texto, aquellas que son obligatorias cubrir, **se pueden destacar con un asterisco o bien cambiarles el fondo**.

En el siguiente **enlace** te permiten diseñar el botón que luego puedes descargar en formato **png** o su código CSS.

4. La **disposición** o ***layout***

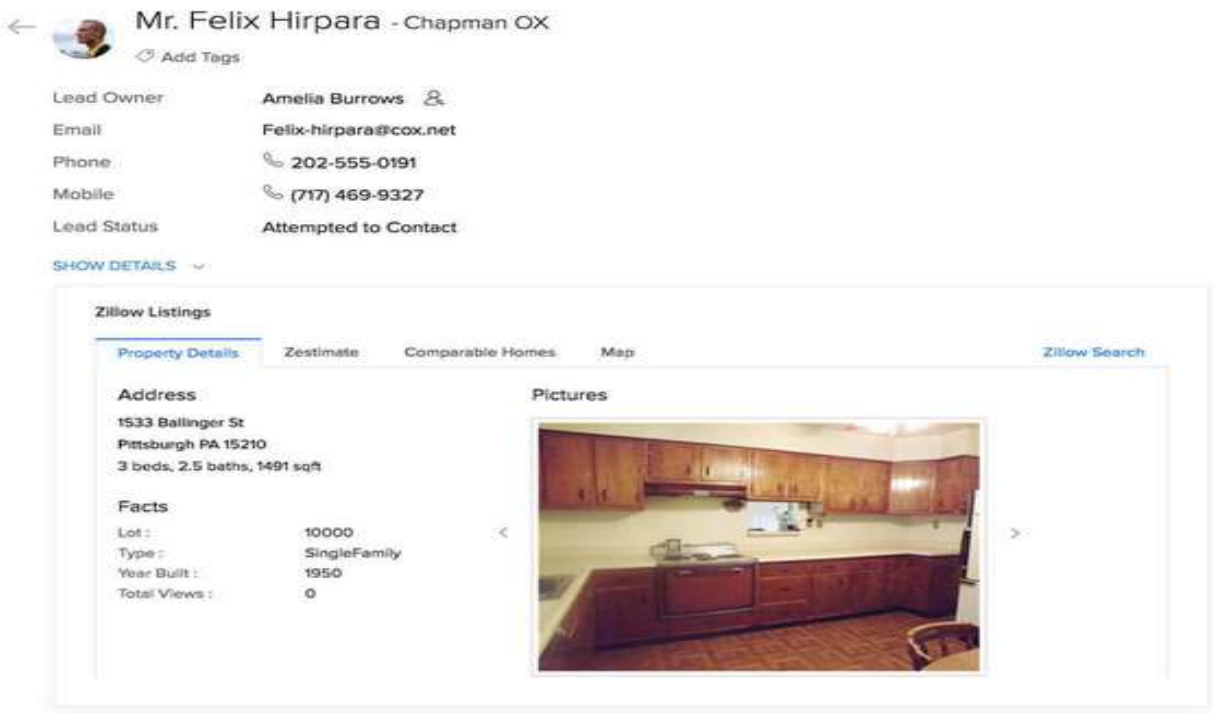
De todos **los factores que pueden influir en la calidad de las interfaces de usuarios, la disposición de los diferentes *widgets* es el de mayor influencia.**

La primera premisa que debemos seguir es **menos, es más.** Hay que evitar controles inútiles o bien, aquellos que no sean relevantes llevarlos a un segundo plano de actuación. Por ejemplo, el botón de imprimir si puede ser importante que esté a la vista en un documento en Word, pero no así si estamos utilizando una herramienta de desarrollo o en una aplicación de streaming.

La segunda premisa es que los ***widgets* deben ser familiares.** No inventemos botones que ya existen y que son más que intuitivos. Volviendo al ejemplo de imprimir, el que aparece en la mayoría de los SO que suele ser la imagen de una impresora antigua (matricial) es más que válido.

Los cuadros **de diálogo y modales:** los justos y necesarios. Los pop-ups, cajas modales y cuadros de diálogo son necesarios, pero **sin abusar de ellos ya que además de interrumpir el flujo de trabajo del usuario** llegando a ser molestos.

Debemos usar los **contenedores de wigdets.** Hay diferentes tipos, absolutos y relativos, pero su gran ventaja es que **permiten manejar varios widgets al mismo tiempo con lo de ahorro de tiempo y mejora de productividad** que supone. Se comportan como las tablas donde en cada celda vas colocando un widget permitiendo mostrar la interfaz de forma ordenada.



En la imagen anterior perteneciente al CRM Zoho, se muestra de forma ordenada una interfaz de gestión de empleados con **dos contenedores claramente diferenciados**. En la parte superior una serie de *labels* a la izquierda y los datos básicos o más importantes contenidos en unos *entry* o *textbox.* En la parte inferior existe un contenedor, más concretamente un ***panel con pestañas*** para entrar en datos o más información acerca, en este caso de las casas que vende este empleado agrupadas en diferentes tipos de características ç

Otro ejemplo de contenedor sería la botonera:

Medidas de la usabilidad

Hay diferentes **métricas para determinar el grado y calidad de una interfaz gráfica**. No se entrará en los modelos matemáticos para la medida de dicha calidad simplemente se definirán las escalas más importantes:

- Las **métricas de la Eficiencia** hacen referencia al **tiempo de media que es necesario invertir para completar cada tarea.** Esta es la métrica principal y se puede determinar con diferentes parámetros:
    - Tiempo invertido en el **primer intento**.
    - Tiempo requerido para **completar una tarea comparado con el que necesitaría un experto**.
    - Tiempo invertido en **subsanar errores** cometidos.

  Si **sobre 10 tareas, 7 superan el tiempo previsto** en una primera aproximación a la aplicación es indicativo de realizar las siguientes mejoras:

    - Observar si existe alguna incoherencia entre los enlaces o **texto de los botones y los nombres de los módulos a los que se dirigen**. Por ejemplo, si queremos imprimir y ponemos como nombre, por ejemplo, *sacar informe* es que el nombre resulta confuso, mejor *imprimir informe.*
    - **Diseñar el mapa de la aplicación de forma lógica**. No pongamos primero el módulo de elaborar facturas si antes no damos de alta al cliente.
- Las **métricas de la Efectividad** miden el "**porcentaje de éxito" (o porcentaje de conclusión) hace referencia a los participantes que alcanzan de forma correcta cada objetivo**. La prueba no debe contar con la ayuda de un moderador. Un porcentaje por **encima de 75% es aceptable**. Si hay problemas, podrían considerarse las siguientes medidas:
    - Asegurarse de que los **botones tienen aspecto de botones**.
    - **Simplifica los procesos o flujos de tareas**. Cuantos menos para realizar un proceso mejor. Ello reduce la curva de aprendizaje del uso incluyendo si es necesario instrucciones introductorias.
- La **métrica o encuestas de la Satisfacción** puede medirse y calcularse por medio de la **"Escala de Sistemas de Usabilidad".** La escala estándar debe contener al menos diez preguntas que **miden la opinión general del usuario sobre la usabilidad del software.** Por ejemplo, si realizas las pruebas con cuatro sujetos y, de media, se obtiene un resultado **inferior a un 75%,** probablemente sea una señal de que se necesita trabajar más en el diseño del proceso o de la interfaz. En el siguiente cuadro se puede observar las preguntas para determinar la satisfacción dentro de la escala de sistemas de usabilidad más utilizadas. (traducción de la web *usability.gov).*

1. *Creo que usaría este [sistema, objeto, dispositivo, aplicación] frecuentemente*
2. *Encuentro este [sistema, objeto, dispositivo, aplicación] innecesariamente complejo*
3. *Creo que el [sistema, objeto, dispositivo, aplicación] fue fácil de usar*
4. *Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar este [sistema, objeto, dispositivo, aplicación]*
5. *Las funciones de este [sistema, objeto, dispositivo, aplicación] están bien integradas*
6. *Creo que el [sistema, objeto, dispositivo, aplicación] es muy inconsistente*
7. *Imagino que la mayoría de la gente aprendería a usar este [sistema, objeto, dispositivo, aplicación] en forma muy rápida*
8. *Encuentro que el [sistema, objeto, dispositivo, aplicación] es muy difícil de usar*
9. *Me siento confiado al usar este [sistema, objeto, dispositivo, aplicación]*
10. *Necesité aprender muchas cosas antes de ser capaz de usar este [sistema, objeto, dispositivo, aplicación]*

En el siguiente recuadro presentamos las normas ***ISO*** o estándar existentes **a título informativo (no es necesario aprenderlo de memoria)** para la elaboración de una interfaz gráfica para usuario adecuada que pueden ser consultadas.

ISO 9241-10: Principios para diálogos, 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 y 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 (cursores 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, 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.

Hay multitud de **métodos para medir el grado de usabilidad,** pero nos centraremos en las líneas generales de dos de ellos que básicamente resumen al resto:

- **User testing**: en este método el usuario tiene una gran **influencia el usuario** en el desarrollo del prototipo de la aplicación. Por ello requiere una gran planificación y supone un gran coste.
- **Usability inspection**: 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.

Pautas de diseño.

Las **pautas de diseño** se basan en los siguientes parámetros:

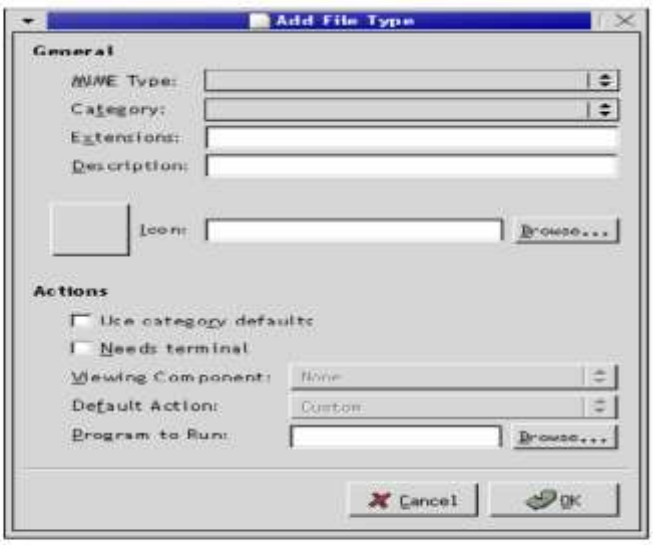
- **Estructura de la información y las tareas del usuario en la aplicación**: Se distingue aquí la **posición y jerarquía de los elementos visuales**, lo más importante es lo que tienen que prevalecer. Si estamos en gestión de clientes, botones como alta, baja y los campos de texto deben estar en la zona focal de la ventana.
- **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. 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 con 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 y otros. **No se debe utilizar en una misma interfaz distintos tipos de botones o etiquetas**, por ejemplo.
- **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 o los datos personales de alguien deben de estar en un recuadro delimitado y no desperdigado por la interfaz.
- **Legibilidad y Flujo entre los elementos:** Facilidad de lectura y comprensión de la comunicación de las ventanas, dando espaciado y alineación a 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.

Veamos algunas **directrices generales** las cuales que, por cierto, no son seguidas por muchos programadores pero que convendría considerar estas recomendaciones:

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 para tener en cuenta son:

- Cuando los cuadros de texto tengan **la misma longitud**, se recomienda **alineación izquierda.**
- Si la mayoría de las etiquetas o *labels* **difieren en longitud**, se recomienda **alineación derecha**.
- Mantener la 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.

Barra de menús: Proporciona acceso submenús desplegables (*drop-down* o *pull-down menú*). Solo se muestra el título del menú hasta que el usuario hace clic sobre el ítem. Esta barra esta siempre **visible y accesible desde el teclado y/o el ratón**.

- 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**.

Dentro del menús están los **submenús desplegables.** Cuando se selecciona un ítem de la barra de menú (clickeando con el ratón o con el foco pulsando Enter) aparecen los **submenús desplegables**.

Se deben organizar los ítems en **grupos relacionados** sobre la función que realizan. Por ejemplo, no poner el desplegable guardar fichero en el ítem principal formato.

- Hay discusión de cuantos submenús deben existir, pero nunca menos **mínimo de 7 y un máximo de 10**.
- Evitar la creación de nuevos ítems en tiempo de ejecución.
- Usad combinación de teclas en los más frecuentes.

Con respecto a los **menús en cascada.**

- Se debe aplicar su uso **solo en casos necesarios**, dificulta la navegación y la búsqueda de los ítems.
- No diseñar submenú con menos **de tres opciones,** a menos que sus ítems sean agregados dinámicamente (tipo Archivos Recientes Usados de algunas herramientas de productividad).
- **No puede haber más de dos niveles de jerarquía.**

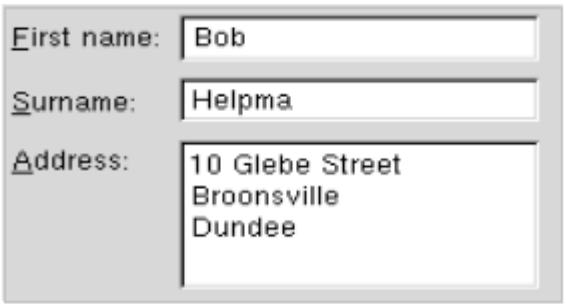
Finalmente tenemos los **menús contextuales.** Es un tipo de menú desplegable que se muestra bajo determinadas situaciones cuando están enfocado un objeto y presionarse el **botón derecho del ratón.** Se emplean como su nombre lo indica, para proporcionar la ejecución "Contextual" de una serie de comandos asociados **al objeto que tiene el foco al momento** de ser invocado (con botón secundario del ratón) el menú.

- Es usado principalmente por **usuarios intermedios y avanzados**.
- Se debe proporcionar **acceso alternativo para cada una de las funciones** o tareas que configure en un menú contextual con un **máximo de 10 ítems.**
- **Evitar el uso de menú de cascada** dentro de los menús contextuales.
- Usar **la línea como separador gráfico** para denotar agrupación de opciones relacionadas.

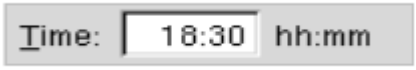
Cuadros de Texto (*TextBox o Entry*)

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

- Rotular los *TextBox* con etiquetas textuales colocadas del control *textbox*, de acuerdo con el uso de mayúsculas Oración.



- **Justificar a la derecha** los cuadros de texto **cuyo contenido sea exclusivamente numérico**.
- Ajustar **el tamaño del cuadro de texto de acuerdo con el probable tamaño de los datos de entrada**. Esto **da una información visual útil** acerca del tamaño de la entrada que se espera. **Evitar colocar del mismo ancho en 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:



- 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: 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

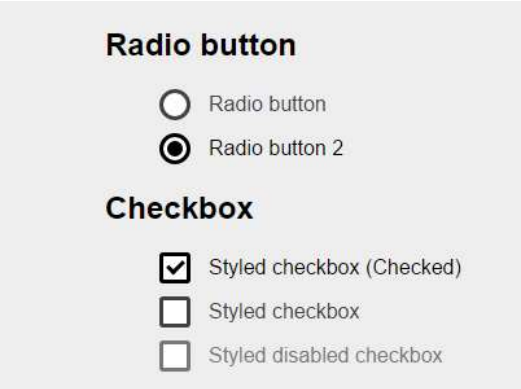
Un botón de comando o *Comand button,* inicia una acción determinada cuando el usuario hace clic sobre él.

- Rotular todos los botones con **verbos en infinitivo**, en combinación con un adjetivo, si se requiere, aplicando el uso de mayúsculas, por ejemplo: *Guardar, Ordenar, Actualizar...*
- 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 u operaciones 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**.
- **No asociar acciones a los eventos Doble-Clic ni Clic-Derecho** de un botón de comando.

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 (sí/no, casado/soltero/viudo, tramos de edad, etc…)**. Estos valores son cada uno **mutuamente excluyentes** y al menos debe ser dos opciones**.**

- 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. Por ejemplo, que al seleccionar tramo de edad te aparezca algún tipo de descuento en la compra o similar.
- Para rotular el grupo de botones de opción, use **combinación de mayúsculas de encabezado,** por ejemplo, **E**stado **C**ivil. Ubique esta etiqueta del grupo arriba de los botones o al lado izquierdo de los mismos.
- El número de elementos **para botones de opción no debe exceder de 8**, llama a confusión.
- 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 conjun**to, estos valores **no son mutuamente excluyentes**.

- 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 el grupo botones de chequeo**, ubicando la etiqueta arriba o a la izquierda.
- El número de elementos **para botones de chequeo no debe exceder a 8**, llama a confusión.
- Tratar de alinear **los botones de chequeo verticalmente** ya que facilita la visibilidad.

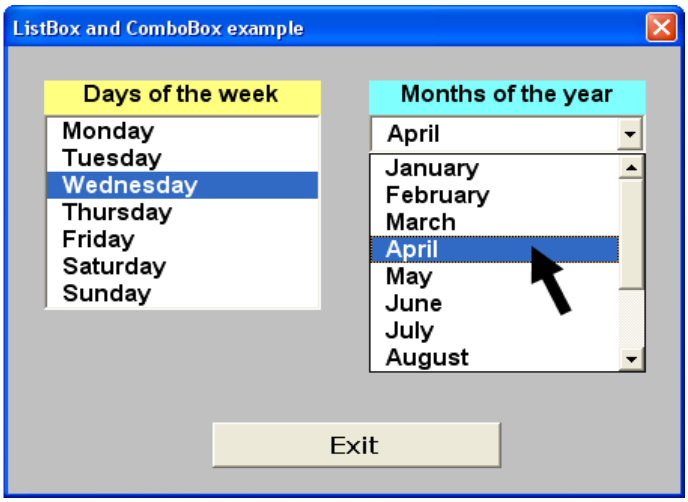
Objeto Cuadro Combinado *(Combo Box)*

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.

- Se recomienda su uso para gestionar la selección de un único valor entre un conjunto de valores de **más de 8 elementos**, por ejemplo, provincias.
- La selección de un ítem **no debería iniciar ninguna acción en la aplicación**, aunque hoy en día si suele iniciar otro evento. Por ejemplo, en provincias suele cargar los municipios al elegir la provincia.
- 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.

Objeto Lista *(List Box)*

Son usados para brindar al usuario la **capacidad de selección de uno o varios valores** dentro de un conjunto. Se debe rotular con una etiqueta colocada **arriba o a la izquierda del control**, usando combinación **de mayúscula de encabezado**.

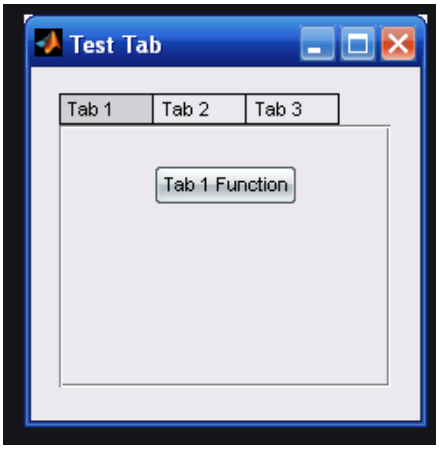


- Aplicar a los elementos de la lista, combinación de **mayúsculas de oración.**
- En el diseño de la lista para **mostrar al menos cuatro ítems** antes de realizar s*crolling*.
- **No diseñar listas con menos de 5** ítems ni más de **10**.
- Sólo usar encabezado de columnas cuando la lista tenga más de una columna
- Para listas de selección múltiple **se aconseja mostrar el número de ítems** actualmente seleccionados en un texto estático debajo de la lista. Por ejemplo: *"Número de Item Seleccionados: 5"*. 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.

Objeto Tabbed (Control Tabbed o Tabbed NoteBooks)

Es un objeto adecuado para **presentar información relacionada en la misma ventana**. Es unos de los elementos más utilizados hoy en día en la organización y presentación de la información en una aplicación ya que evita la apertura y cierre de ventanas.

- **No colocar demasiadas** páginas en el mismo cuaderno, como máximo 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.



Panel con Tabs

Radio botones y marcos

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. Antes de agregar marcos y separadores gráficos evaluar la posibilidad de diseñar con *indentación* y espaciado para reflejar la relación entre controles, ya 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.

Estas son algunas de las recomendaciones más básicas de los objetos más comunes utilizados en las interfaces gráficas. No todos los desarrolladores, incluidos los más expertos, las siguen, pero en líneas generales hay cierto consenso a la hora de aplicarlas.

Informes

Al planear la creación de una aplicación una de las consideraciones es la posibilidad de la inclusión de informes que permitan al usuario acceder a la información de una forma estructurada.

Un informe suele generarse a partir de un **diseño previo o plantilla** en el cual **se vuelcan la información** procedente, principalmente, de una base de datos. Luego se podrá imprimir o visualizar una vez generado, pero sin modificación alguna. Aquellas modificaciones que se deseen precisan de **una generación nueva del informe**.

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

- El **informe incrustado o integrados** el proyecto crea directamente dicho informe. Para ello debe crearse **una clase contenedora** del mismo que represente el informe en el proyecto interactuando con ella directamente el código de la aplicación. Al compilar el proyecto tanto el diseño del informe como la clase que lo contiene **se incrustan en el ensamblado como un módulo o paquete adicional**, es el caso de ACCESS de Office. Apenas se usan.
- Por otro lado, **un informe no-incrustado o no-integrado** como su nombre indica **es externo al proyecto** y se genera mediante **una herramienta independiente** y que no forma parte del proyecto compilado. A un informe no incrustado siempre se obtiene acceso externamente de distintas formas. El informe se puede descargar en la unidad local, como servicio Web en formato HTML o XML. También, el informe puede ser creado en formato **pdf** a través de herramientas externas como *Crystal Reports o ReportLabs*, entre otras. Son más sencillos y seguros, **pero requieren de mayor trabajo en su diseño.**

A continuación, daremos una lista de las aplicaciones para la elaboración de informes más utilizadas.

- **Crystal Reports**: es una aplicación utilizada para diseñar y generar informes desde una amplia gama de fuentes de datos. Es utilizada por muchos IDE y lenguajes de programación. El formato de salida puede ser *.pdf, .rpt, .xls, xml, .txt y .CVS* entre otros. El **formato CVS** 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.

  "Nombre", "Apellidos", "Email", "Edad"

  "Mario", "", "mario@misitioweb.com", "34"

- **JasperReports + iReport** 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 . JasperReports se usa comúnmente con iReport, un front-end gráfico de código abierto para la edición de informes.
- **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 IDE Visual Net de Microsoft.

Estructura general de un informe.

De forma general un informe puede tener hasta 5 **secciones** y aunque no todas se las incluyen**:**

Seccion 1: Report Header. Cabecera del informe, donde se imprime una sola vez al inicio del reporte. Puede contener desde el logo de la empresa, fechas, nombre del informe...

Seccion 2: Page Header. Cabecera de página, donde se imprime al inicio de cada página impresa. Contiene **anotaciones generales**, por ejemplo, si es una factura o el nombre y dirección del cliente, número de factura.

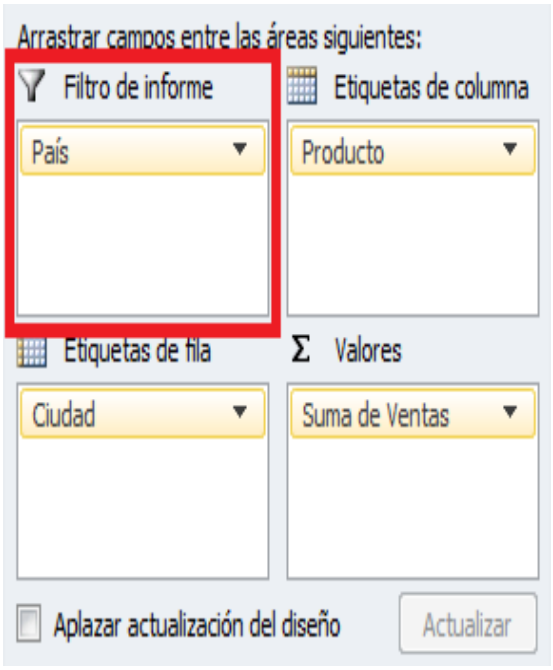
Seccion 3: Details. Detalle del informe, donde las filas o registros que forman el reporte. Es allí donde se alojan los campos del origen de datos y **es la parte más dinámica y la más sometida a cambios.**

Seccion 4: Report Footer. Pie del informe, donde se imprime una sola vez al finalizar el reporte. Se utiliza esta sección para imprimir **los totales generales, promedios**...

Seccion 5: Page Footer. Pie de página, donde se imprime al final de cada página. Se utiliza esta sección para imprimir la **paginación, los totales por página**.

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 interés del usuario. De esta forma en **los agrupamientos pueden ser interesantes los filtrados de datos.**

Por ejemplo, listado de empleados por departamentos que obtuvieron un mínimo de X ventas o que pertenezcan a tal o cual ciudad o el listado de clientes por vendedor.



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

- **modelo de datos:** sería la plantilla del informe
- **acceso a los datos:** captura de los datos desde la base de datos
- **presentación de los datos:** diseño final ante el usuario

En primer lugar, llevamos a cabo el **diseño del informe que no se más que una "plantilla"** que utilizará el motor de la generación del informe. Como es un documento XML este lleva un DTD asociado que se encuentra en la mayoría las librerías de las herramientas de generación de informes. Su estructura es similar a la de cualquier documento de texto, pudiendo tener más atributos que los enseñados en la imagen. Posteriormente se compila para llevar a cabo la presentación del informe a través del método correspondiente de la herramienta.

Posteriormente se lleva a cabo una llamada a la base de datos para cargar aquella información a petición del usuario en la plantilla y finalmente se realiza la presentación, bien en formato *.pdf, .html,* etc.

Documentación de programas

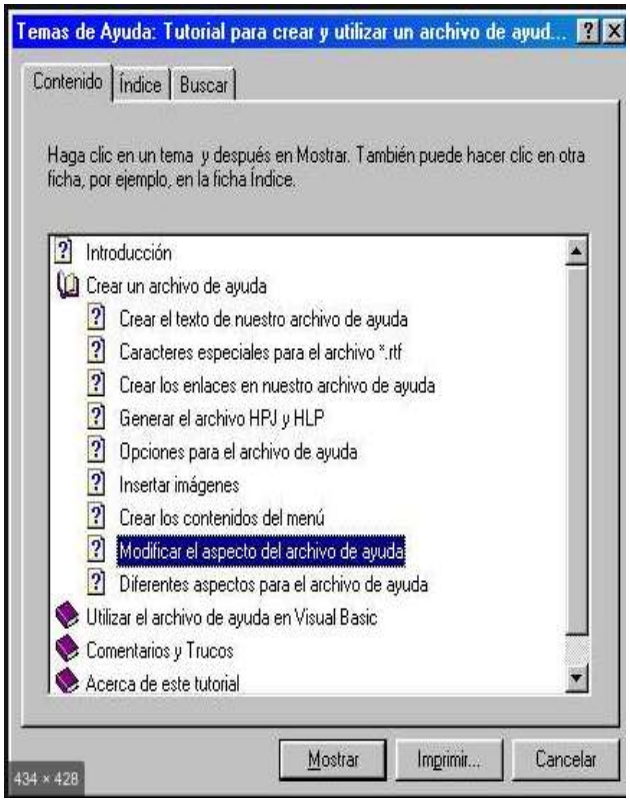
Un **fichero de ayuda** es un documento sobre papel o digital cuyo fin es servir de **guía de referencia**, **manual o como su nombre indica ayuda** a los diferentes usuarios de una aplicación y/o sus desarrolladores, en este caso, a aquellos que puedan en un futuro modificar dicha aplicación.

Los formatos de ayuda a usuarios, ciñéndonos a soporte digital, pueden ser en **pdf o cualquier formato ofimático** que cada vez son menos habituales.

Luego están los que se encuentran **incrustados en la propia aplicación**.

Winhelp (.hlp) son archivos con extensión ***hlp*** de ayuda de Windows, donde el nombre de la extensión proviene de Help. Al hacer doble click sobre un archivo con extensión hlp éste se abrirá con la **aplicación Ayuda de Windows** gracias al programa ***winhlp32.exe***. Fueron los primeros.

El formato de archivo ***.hlp*** 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 más recientes como **.chm.**



Un archivo de WinHelp 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.

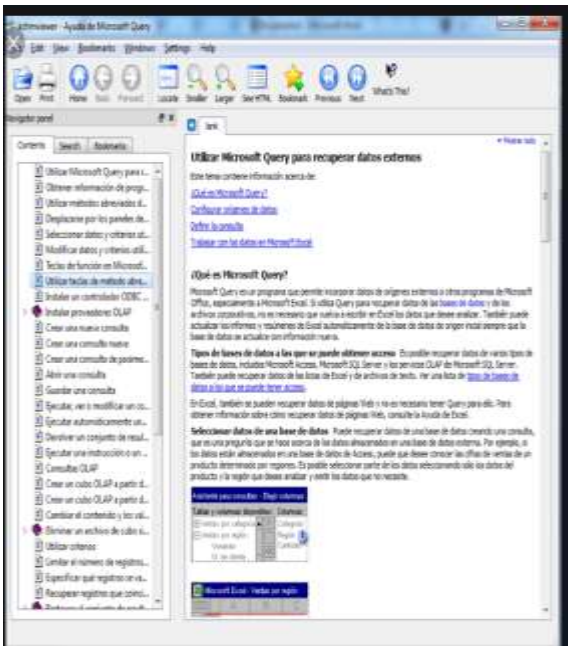
Los archivos *.hlp* pueden convertirse en otros formatos como *.html o .pdf* existiendo para ello diferentes tipos de herramientas.

.chm es el archivo de Ayuda de HTML Compilado (Microsoft Compiled HTML Help en inglés) es un formato privativo de ayuda en línea desarrollado por Microsoft. Se popularizó con Windows 98, pero sobre todo se usó considerablemente en 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. Fue sustituido por **el MAML o Microsoft Assistence Markup Language.**

La sustitución tuvo lugar debido a que 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***.



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.

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

- Preguntas frecuentes conceptuales
- Glosario
- Procedimiento de referencia

- Contenido reutilizable
- Tarea y solución de problemas
- Tutorial.

La presentación permite que el contenido creado en MAML utilizar muchos formatos diferentes, incluyendo DHTML, XAML, RTF y material impreso. En la representación se aplican hojas de estilo y visualiza el contenido final a los usuarios.

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 más información como *changelogs, readmes especifícos 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.

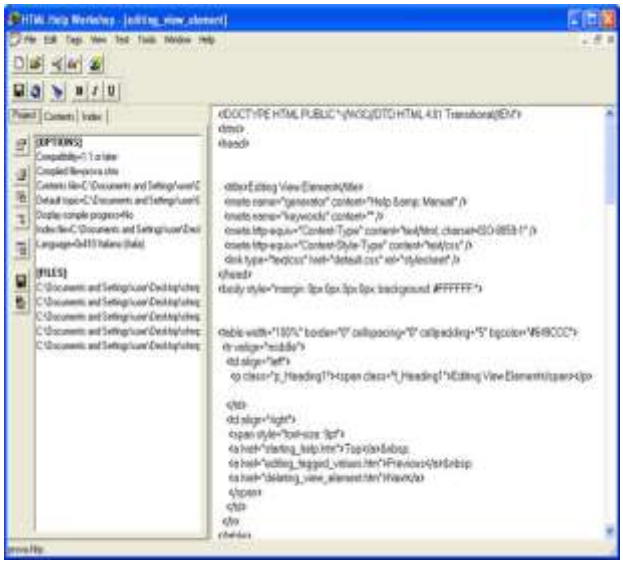
Herramientas de generación de ayudas. Ayudas genéricas y sensibles al contexto.

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..
- **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.



- **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: Java, 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).
- **Javadoc:** permite la búsqueda de código comentado para generar las ayudas o documentación correspondiente de aplicaciones en Java.

Tipos de manuales: manual de usuario, guía de referencia, guías rápidas, e manuales de instalación, configuración y 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 esta. 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 de este.

La documentación de un programa **empieza a la vez que la codificación de este** y finaliza justo antes de la entrega del programa o aplicación al cliente.

Una vez concluido el programa, los documentos que se generan son **la guía técnica, la guía de uso y la instalación si fuese necesario.**

Tipos de documentación

La documentación 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 sí. Dentro de esta categoría también se encuentra la **ayuda electrónica.**

  - **La guía técnica**

En la guía o manuales técnicos se reflejan el diseño del proyecto, la codificación de la aplicación y las pruebas realizadas para su correcto funcionamiento. Generalmente este documento está diseñado para 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 está 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 suprimen los tecnicismos** y se presenta de forma que sea comprensible para el usuario que no sea experto en informática.

En el caso de que se haga uso de algún tecnicismo debe ir acompañado de un glosario al final de esta para su fácil comprensión. Las secciones de un manual de usuario a menudo incluyen:

- Una página de portada, una página de título y 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 las principales funciones del sistema.
- 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 utilización de este. Debido al tipo de aplicaciones actuales y a la automatización de las instalaciones apenas se genera esta ayuda.

Distribución de aplicaciones

El **empaquetado de aplicaciones de escritorio** consiste en proporcionar a los futuros usuarios las aplicaciones en forma de paquetes, termino en inglés conocido como ***software bundle o application bundle*** para su instalación y posterior uso. Estos paquetes están formados por:

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

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,** sea cual fuere en un paquete único.

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 los ficheros necesarios, 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 dis**co, 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.

Estos 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 Structure 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 bajo servidor** Windows.

Estos paquetes **se pueden incluir software de terceros** para luego instalarlos en equipos en red pertenecientes a un dominio, de forma que cuando un usuario de un dominio inicio su sesión desde un equipo cliente desde los servidores se envía el paquete para que se instala antes de ponerse a usar el equipo, es decir, **al iniciarse**. 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. W**indows Installer** identifica a los **paquetes por un GUID**.

• **Java:** el principal es **JAR** que 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.

En cuanto a **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 descargado 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 paqute

  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 compilar 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 Mangaer) son los de la distribución Red Hat y derivados (OpenSuse, Mandriva, Centos...).

Un paquete .rpm no lo podremos instalar en la distro Debian o derivados, y un .deb tampoco en RedHat y derivados. No obstante, existen programas, por ejemplo, uno llamado 'Alien' que permite convertir un paquete .rpm a .deb y viceversa, aunque a veces con resultados algo dudosos.

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.

Dentro de un paquete deb podemos configurar los siguientes datos:

- ***Maintainer:*** *el desarrollador principal del paquete.*
- ***Summary****: una descripción del paquete.*
- ***Name:*** *nombre que quieres 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.*

Una vez hayamos modificado lo que queremos, compilaremos el código para obtener el paquete. Cuando haya terminado, en el directorio donde compilamos habrá aparecido un paquete ***.deb*** de la aplicación, listo para instalar.

De todas formas, aunque resulte una mecánica sencilla, la **aparición de errores suele ser habitual.** Linux adolece de empaquetadores de software estables y con la eficiencia de Windows.

En **RedHat** para 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

Esto nos creará un entorno de trabajo para crear los RPM's o "Árbol del Proyecto".

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

- **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 y 64 bits superiores) Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7 y sucesores.

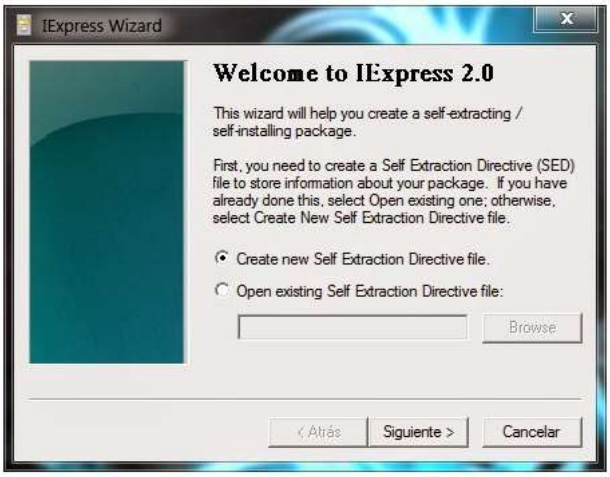
Iexpress.exe se utiliza para crear auto-paquetes de un conjunto de archivos. Estos paquetes pueden utilizarse para instalar aplicaciones, el ejecutable, los paths, otros componentes del sistema, o *bootstrappers* de configuración.

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) utilizando la interfaz proporcionada que facilita la automatización.

Todos los archivos autoextraíbles creados por el uso de ***Iexpress*** se comprimen con el MakeCab (MAKECAB.EXE) herramienta, 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.

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 de este. Por ello no solo deben ser borrados los archivos, sino también cambios en otros 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 contra productiva sino también poner en peligro la estabilidad del sistema. El desarrollador del software debe ofrecer una función para de instalar su software sin dañar o desestabilizar el sistema.

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 **eliminen 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 (derivados Debian) los comandos para desinstalar aplicaciones son los siguientes:

 # apt remove "nombre-del-paquete" //desinstalamos el paquete

 # apt purge "nombre-del-paquete" //borra los archivos de configuración

 # apt 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.**

En estos sistemas, el software se distribuye en forma de paquetes, frecuentemente encapsulado en un solo fichero. Estos paquetes incluyen otra **metainformación 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.

Entre las funciones principales de los gestores está:

- Comprobación de las 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.

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.

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. 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 *reconfigurar* el software durante la instalación.

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.

Existen también **los CVS, o sistema de control de versiones**, como ***GitHub*** que permiten la distribución de software a terceros. Los veremos en las prácticas.

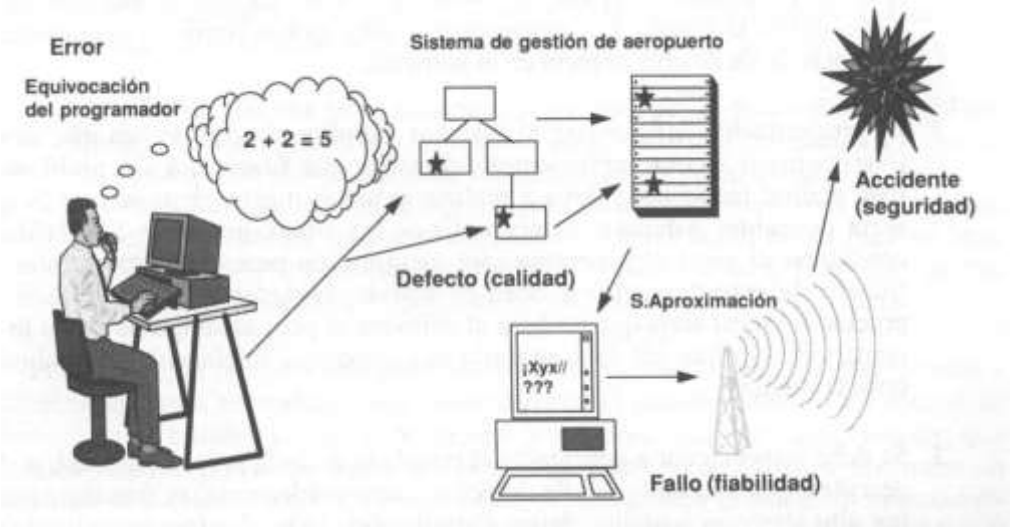
Finalmente, algunos lenguajes de programación interpretados tienen su propio sistema de gestión de paquetes para manejar módulos del lenguaje, como pasa con los lenguajes de programación **Perl (CPAN), Python (pip), PHP (PEAR) o Ruby (RubyGems).** Otros lenguajes pueden venir con su propio sistema para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

Snap es un nuevo concepto introducido con Ubuntu 16.04. Se trata de una nueva forma de instalar aplicaciones en Ubuntu, que resuelve muchos problemas y simplifica, aún más, la instalación de estas para los usuarios menos avanzados.

pip es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python. Muchos paquetes pueden ser encontrados en el Python Package Index (PyPI). Python 2.7.9 y posteriores (en la serie Python2), Python 3.4 y posteriores incluyen pip (pip3 para Python3) por defecto. **pip es un acrónimo recursivo que se puede interpretar como Pip Instalador de Paquetes o Pip Instalador de Python.**

Pruebas de software

Las **pruebas de software** son investigaciones empíricas 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 control de calidad siendo una etapa más dentro del desarrollo de software.



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 **sino la detección de defectos en el software (descubrir un error es el éxito de una prueba)**. 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 satisfacen las condiciones impuestas al comienzo de dicha fase, es decir, responde a la pregunta: **"¿estamos construyendo el producto correctamente?".**

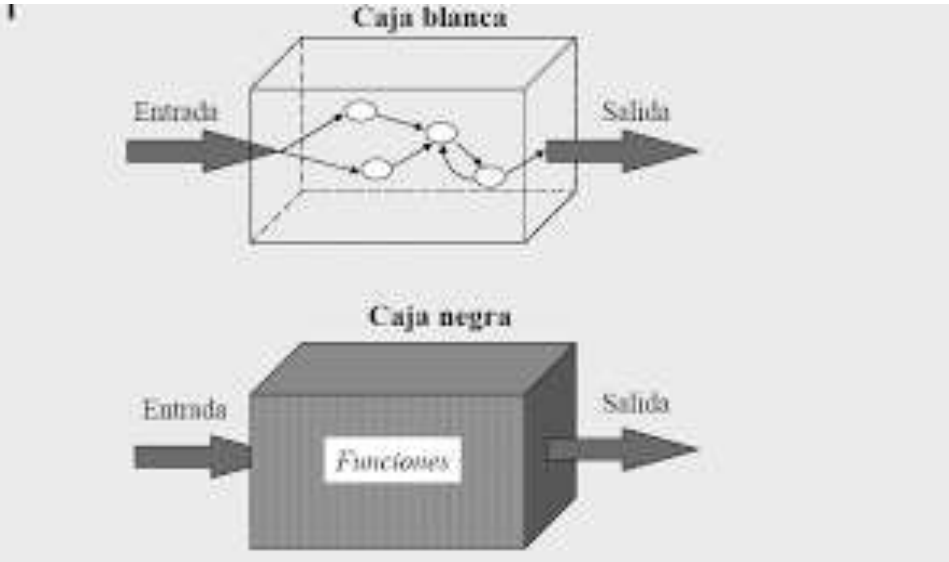
Finalmente está el **proceso de validación** o proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos marcados por el usuario, es decir, responde a la pregunta: **"¿estamos construyendo el producto correcto?"**

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 incorrecto en un programa**. Por ejemplo: "el software es incapaz de cargar los sumandos o no se lanza".
- **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".

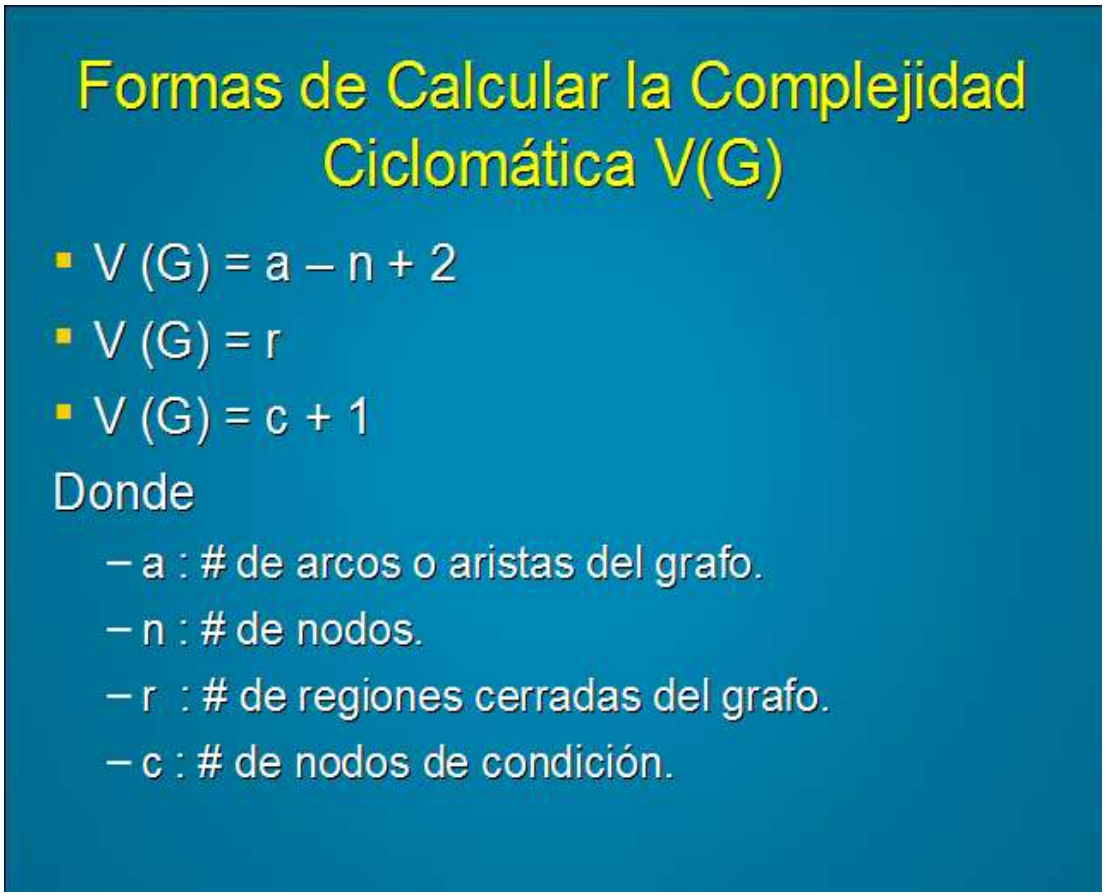
Las **pruebas de Caja Negra** o integración también conocidas como Pruebas de Comportamiento, estas pruebas se basan en la especificación del programa o componente a ser probado para elaborar los casos de prueba independientemente de su diseño interno o *"de cómo lo haga".* El componente se ve como una "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.



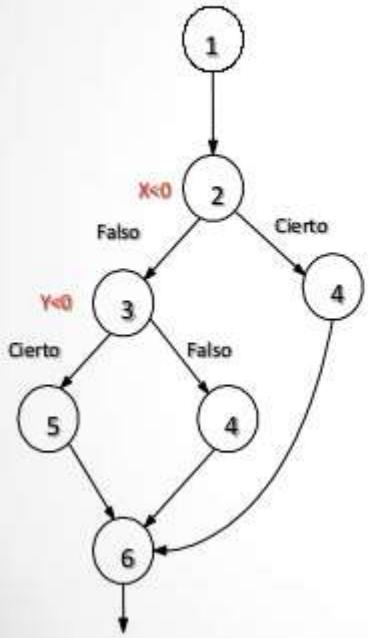
El objetivo de la técnica es diseñar casos de prueba para que se ejecuten, al menos una vez, todas las sentencias del programa, y todas las condiciones tanto en su vertiente verdadera como falsa.

De estas pruebas se obtiene la **complejidad ciclómatica** de los algoritmos.



Veremos unos ejemplos sencillos:

Prueba del camino básico: Ejemplo



V(G) = 3 regiones. Por lo tanto, hay que determinar tres caminos independientes.

- Camino 1: 1-2-3-5-6
- Camino 2: 1-2-4-6
- Camino 3: 1-2-3-4-6

Casos de prueba para cada camino:

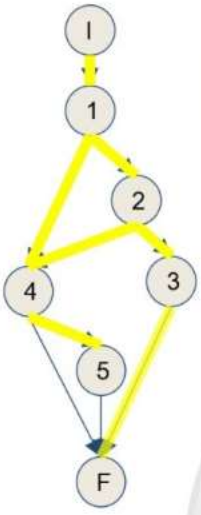
Camino 1: x=3, y= 5, rdo=4

Camino 2: x=-1, y=3, rdo=0, error

Camino 3: x=4, y=-3, rdo=0, error

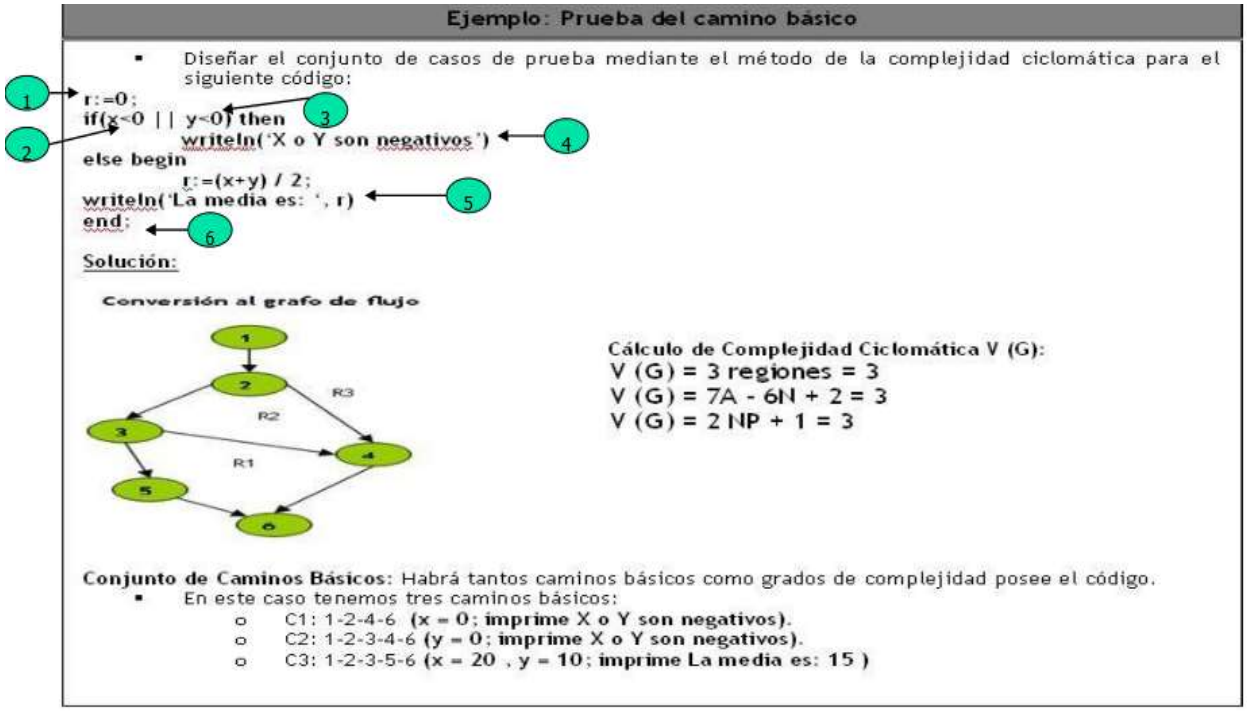
```
public void aMethod() {

     if (a && b) {
          doSomething();
     } else if (c) {
          doSthElse();
     }
}
```



V(G) = Aristas-Nodos+2

V(G) = 9 - 7 + 2 = 4



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.
- Al generar casos de prueba, se deben incluir tanto **datos de entrada válidos y esperados como no válidos e inesperados.**
- Probar si el software **no hace lo que debe hacer**.
- Probar si el **software hace lo que no 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.

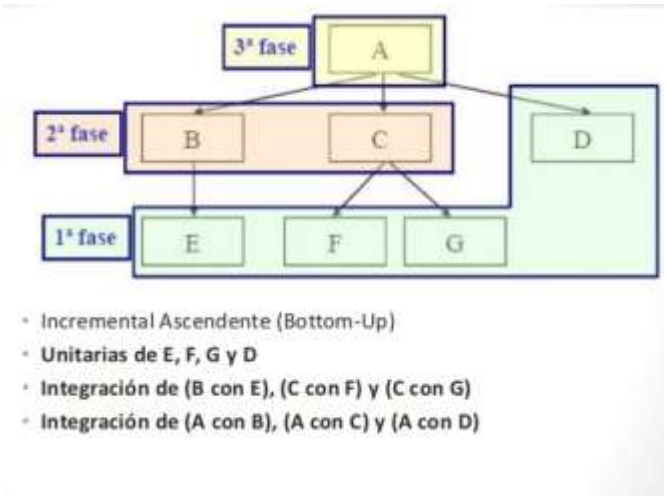
- Las pruebas son una tarea tanto o más creativa que el desarrollo de software. Siempre se han considerado las pruebas como una tarea destructiva y rutinaria.
- Es interesante planificar y diseñar las pruebas para poder detectar el máximo número y variedad de defectos con el mínimo consumo de tiempo y esfuerzo.
- Las tareas para realizar y probar un software según el orden establecido son:
    - Diseño de las pruebas.
    - Generación de los casos de pruebas.
    - Definición de los procedimientos de las pruebas.
    - Ejecución de las pruebas.
    - Análisis e informes de los resultados.

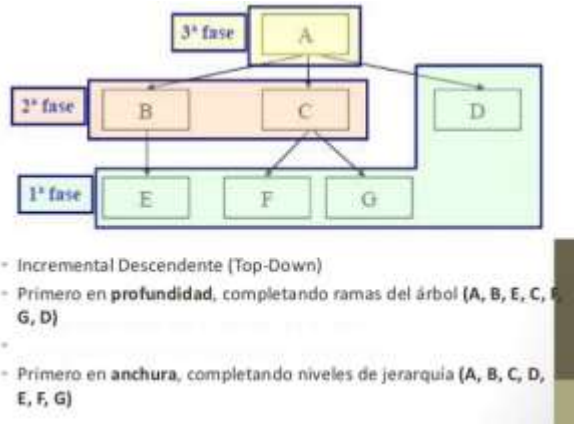
Pruebas de integración

Aun 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 individuamente puede crecer hasta niveles inaceptables al combinar los módulos. Y **este es el objetivo de las 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**.





Pruebas de sistema

Este tipo de pruebas tiene como propósito probar 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 mencionarán porque su nombre es *autoexplicativo):*

- **Pruebas de comunicaciones**. Comprueba que las interfaces tanto locales como remotas funcionan adecuadamente.
- **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 progrma 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 los **nuevos errores o bugs, carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software.**

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.

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.

Pruebas manuales y automatizadas

La prueba manual se realiza ejecutando el software, **introduciendo datos de entrada e inspeccionado los de salida**. El proceso es sencillo y no hace falta aprendizaje previo, además ejecutamos las pruebas exactamente como el usuario final, pero tiene algunos inconvenientes: es **repetitiva, susceptible de error, solo pruebas lo que ves y es realizada por el desarrollador.**

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,** para determinar que produce **los mismos resultados siempre.**

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.

Extraído: https://es.wikipedia.org/wiki/Prueba_unitaria

Algunos ejemplos de 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* (especifíca para aplicaciones web creada por Apache Foundation)
- Para la gestión y manejo de pruebas: *rth, QaTraq*. Ambas almacenan los resultados de los diferentes tipos de pruebas.
- Para **pruebas unitarias**: JUnit en Java, muy enfocadas a las pruebas de regresión) y *Testunit* para Python, *NUnit* para NET y *PHPUnit* para PHP