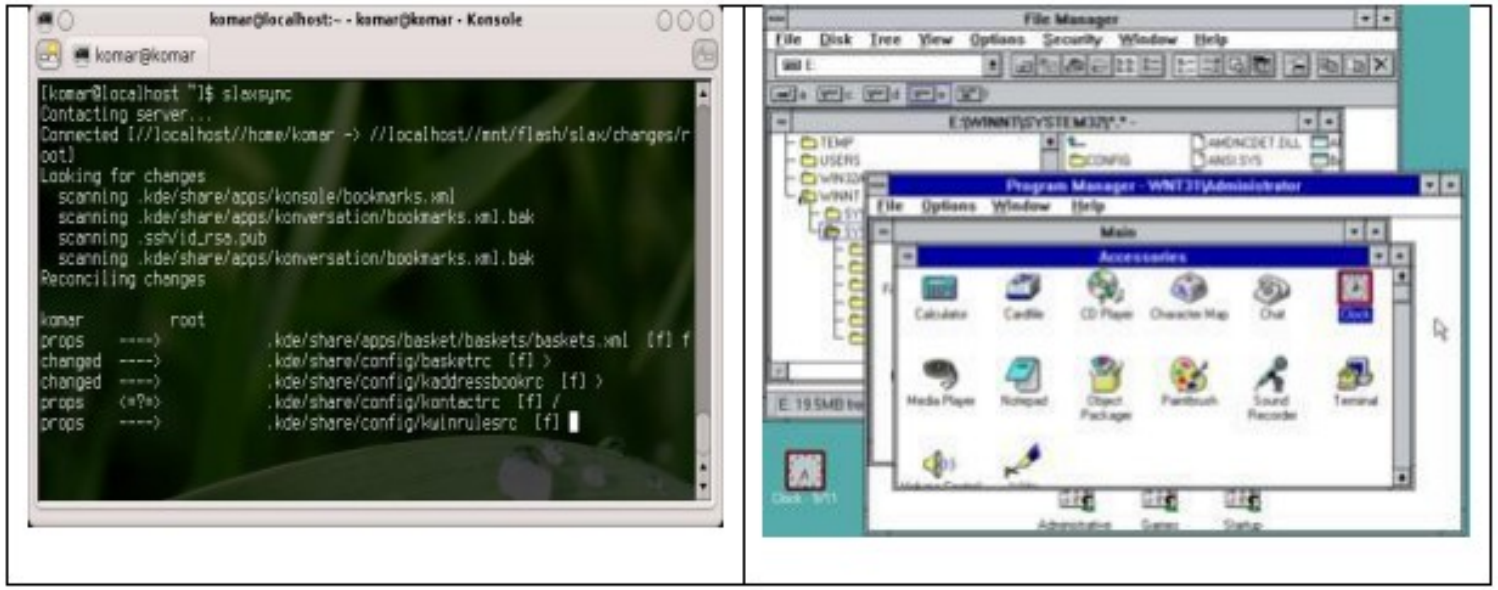
DISEÑO DE INTERFACES

TEORIA

INTRODUCCION (I)

- No fue hasta finales de la década de los 60 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**.



INTRODUCCION (II)

- Posteriormente aparecieron los primeros IU o Interfaces de usuario 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 versión de la programación orientada a objetos en el manejo de las interfaces gráficas. En el siguiente enlace se puede observar la evolución de las interfaces en lo que hace referencia a los sistemas operativos.
- En el futuro, presente ya, se va hacia la generación eventos mediante voz o vista e incluso el ordenador tomará sus decisiones y acciones en función de la observación del usuario.

LA METÁFORA DEL ESCRITORIO- DEFINICIÓN

- La **metáfora de escritorio** 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 (como en las interfaces gráficas de usuarios, sería un 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.

- También existe el control de sistemas operativos por medio de la voz humana, denominado control por reconocimiento del habla o reconocimiento de voz, como por ejemplo Siri, Google Now u OK Google.

CONFECCIÓN DE INTERFACES DE USUARIO

BIBLIOTECA (I)

- 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**.

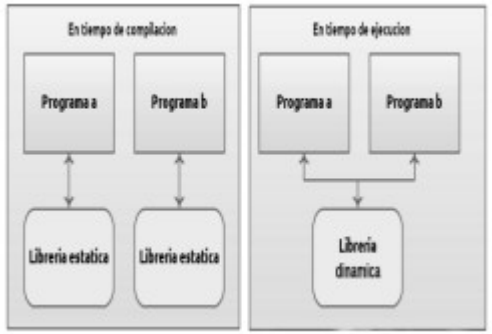
- A diferencia de un **programa ejecutable**, el comportamiento que implementa una biblioteca no es ser utilizada de forma autónoma, sino que **su fin es ser utilizada por otros programas**, independientes y de forma simultánea, eso sí, sin dejar de ser también un programa.
- Incluso unas bibliotecas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

BIBLIOTECA (II)

- El término que utilizamos es el de ***librería***, que es una 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.



TIPOS – LIBRERÍAS ESTÁTICAS

- 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. El enlace estático da como resultado, un archivo ejecutable con todos los símbolos y módulos respectivos incluidos en dicho archivo

TIPOS – LIBRERÍAS DINÁMICAS

- 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. 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 cargar.
- Las librerías que mayor interés para este módulo profesional son las **librerías gráficas**.

LIBRERÍAS GRÁFICAS

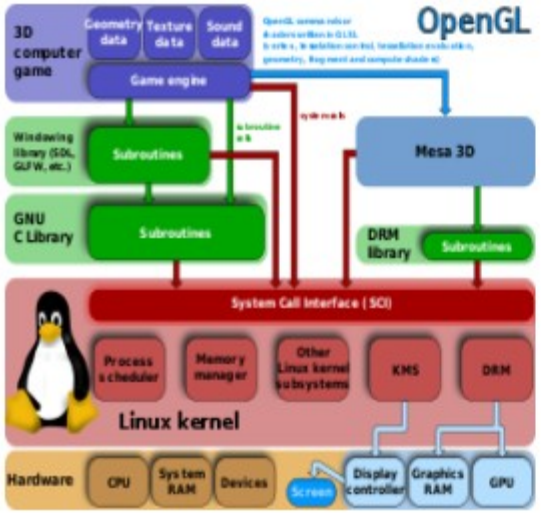
OpenGL (Open Graphics Library)

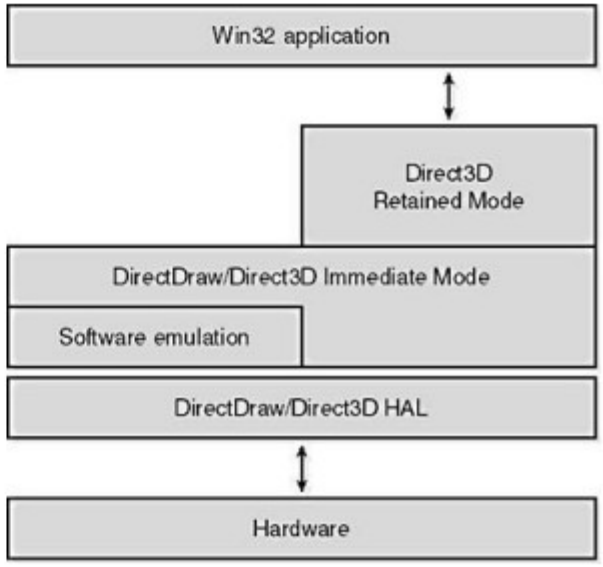
- Es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos.

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

- El funcionamiento básico de OpenGL consiste en aceptar primitivas tales como puntos, líneas y polígonos, y convertirlas en píxeles. Este proceso es realizado por un pipeline gráfico conocida como Máquina de estados de OpenGL. El sucesor de OpenGL se llama ***Vulkan***.

OpenGL


3D computer game
Geometry data
Texture data
Sound data
Game engine
OpenGL
Windowing library (SDL, GLFW, etc.)
Subroutines
Mesa 3D
GNU C Library
Subroutines
DRM library
Subroutines
System Call Interface (SCI)
Process s cheduler
Memory manager
Other Linux kernel subsystems
KMS
DRM
Linux kernel
Hardware
CPU
System RAM
Devices
Screen
Display controller
Graphics RAM
GPU


Win32 application
Direct3D
Retained Mode
DirectDraw/Direct3D Immediate Mode
Software emulation
DirectDraw/Direct3D HAL
Hardware

DirectX-Direct3D

Es parte de DirectX (conjunto de bibliotecas para multimedia), propiedad de Microsoft. Consiste en una API (Application Programming Interfaces o conjunto de rutinas, funciones y procedimientos que ofrecen la biblioteca para la programación de gráficos 3D disponible tanto en sistemas Windows de 32 y 64 bits, y sus consolas Xbox. Direct3D está compuesto por dos grandes APIs: el **modo retenido y el modo inmediato**.

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

GTK “GIMP Tool Kit”

Es una biblioteca que contiene los objetos y funciones para crear la interfaz gráfica de usuario. Maneja *widgets(**wind**ow-gad**get**)* como ventanas, botones, menús, etiquetas, deslizadores, pestañas... y es usado por **Gnome**.

GTK se ha diseñado para permitir programar con lenguajes como C, C++, C#, Java, Ruby, PHP, Perl, o Python.

Está formada por:

- **GLib**. Biblioteca **de bajo nivel estructura básica** de GTK+ y GNOME. Proporciona manejo de estructura de datos para C, portabilidad, interfaces para funcionalidades de tiempo de ejecución como ciclos, hilos, carga dinámica o un tema de objetos.
- **GTK**. Biblioteca la cual realmente contiene los objetos y funciones para crear la interfaz de usuario. Maneja los widgets como ventanas, botones, menús, etiquetas, deslizadores, pestañas, etc.
- **GDK**. Biblioteca que actúa como intermediario entre gráficos de bajo nivel y gráficos de alto nivel.
- **ATK**. Biblioteca para crear interfaces con características de una gran accesibilidad muy importante para personas discapacitadas o minusválidos. Pueden usarse utilerías como lupas de aumento, lectores de pantalla, o entradas de datos alternativas al clásico teclado o ratón.
- **Pango**. Biblioteca para el diseño y renderizado de texto, hace hincapié especialmente en la internacionalización. Es el núcleo para manejar las fuentes y el texto de GTK+2.
- **Cairo**: Permite dibujar las animaciones y sombras de gráficos.
- **GdkPixbuf**: Librería encargada de la carga y manipulación de imágenes.
- **GIO**: permite la manipulación de archivos: apertura, cierre, lectura y escritura.
- **GObject**: es la piedra angular de GNOME. Básicamente permite la abstracción a objetos de **Glib**, librería escrita en C, lenguaje que no soporta la programación orientada a objetos.

QT

- Es una biblioteca multiplataforma usada para desarrollar aplicaciones con interfaz gráfica de usuario, así como también para el desarrollo de programas sin interfaz gráfica, como herramientas para la línea de comandos y consolas para servidores.
- Usada preferentemente **por el sistema de escritorio KDE.**
- **Qt utiliza el lenguaje de programación C++** de forma nativa, adicionalmente puede ser utilizado en varios otros lenguajes de programación a través de bindings.

wxWINDOW

- Es una biblioteca de clases para C++ y Python, que permite el desarrollo de aplicaciones con interfaces gráficas de usuario de una manera rápida y sencilla.
- Su principal característica es que es multiplataforma.
- Se distribuye bajo licencia wXWindows Library License, que es similar a la GNU Library General Public License pero que además permite usar la biblioteca para desarrollos comerciales (ya sean aplicaciones o modificaciones de la propia biblioteca), siempre y cuando estos desarrollos comerciales no usen ningún código distribuido bajo alguna licencia GNU.
- Su mayor ventaja con respecto a las anteriores es su bajo requerimiento de hardware.

DEFINICIÓN- widget

- En informática, un **widget o artilugio** es una pequeña aplicación o programa, usualmente en archivos o ficheros pequeños que son ejecutados por un motor de widgets o Widget Engine.
- Entre sus objetivos están **dar fácil acceso a funciones frecuentemente usadas y proveer de información visual**.
- Dado que son pequeñas aplicaciones, los widgets pueden hacer todo lo que la imaginación desee e interactuar con servicios e información distribuida en Internet; pueden ser vistosos relojes en pantalla, notas, calculadoras, calendarios, agendas, juegos, ventanas con información del tiempo en su ciudad, incluso sistemas de tiendas de comercio, etcétera.

ENTORNOS DE PROGRAMACIÓN
IDE

DEFINICIÓN

- 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** y 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. Además, esto puede ayudar a aprender un nuevo lenguaje de programación de una manera más rápida, así como sus librerías asociadas.

Más

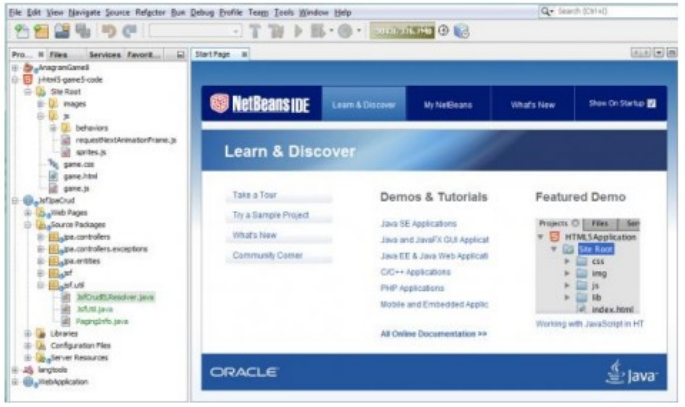
- Algunos IDE están dedicados específicamente a un lenguaje de programación, permitiendo que las características sean lo más cercanas al paradigma de programación de dicho lenguaje.
- Por otro lado, existen muchos IDE de múltiples lenguajes tales como Eclipse, ActiveState Komodo, IntelliJ IDEA, MyEclipse, 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.
- Veremos algunos de los IDEs de mayor implantación.

ECLIPSE

- **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.

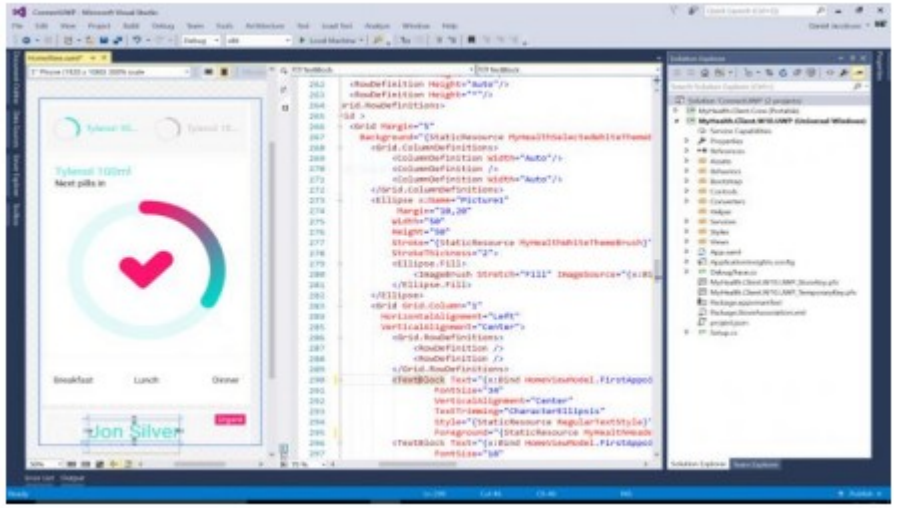
NETBEANS

- 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.
- Es muy utilizado por los programadores.
- El lenguaje que mejor soporta es Java, ya que fue creado por Oracle y su creación fue para ser el IDE de Java. Aunque es multilenguaje debido a que soporta JavaScript, HTML5, PHP, C/C++ y otros.



VISUAL STUDIO

- **Visual Studio** fue diseñado por Microsoft y es uno de los mejores entornos de programación actualmente soporta todo tipo de lenguajes.
- Antiguamente tenían una versión de pago que incluía todos los lenguajes, y **versiones express** que eran gratuitas para un lenguaje en concreto.
- Ahora como Microsoft lo presenta como software libre, 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.

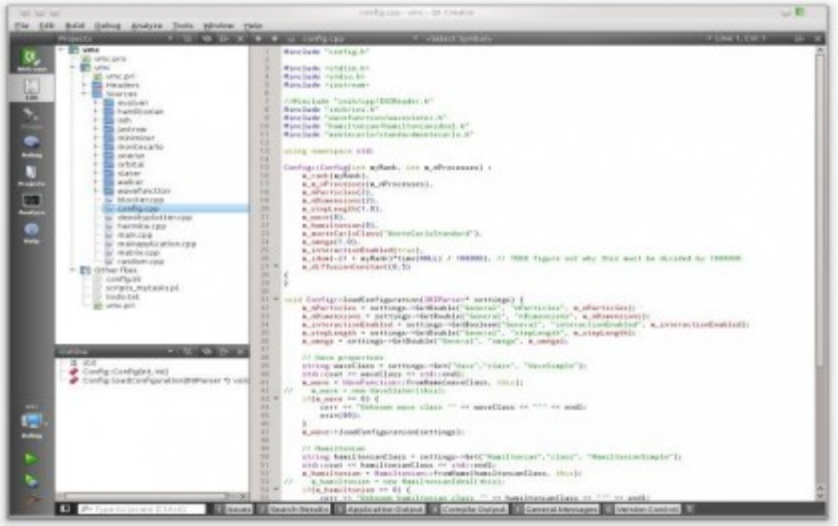


JETBRAIN

- **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.

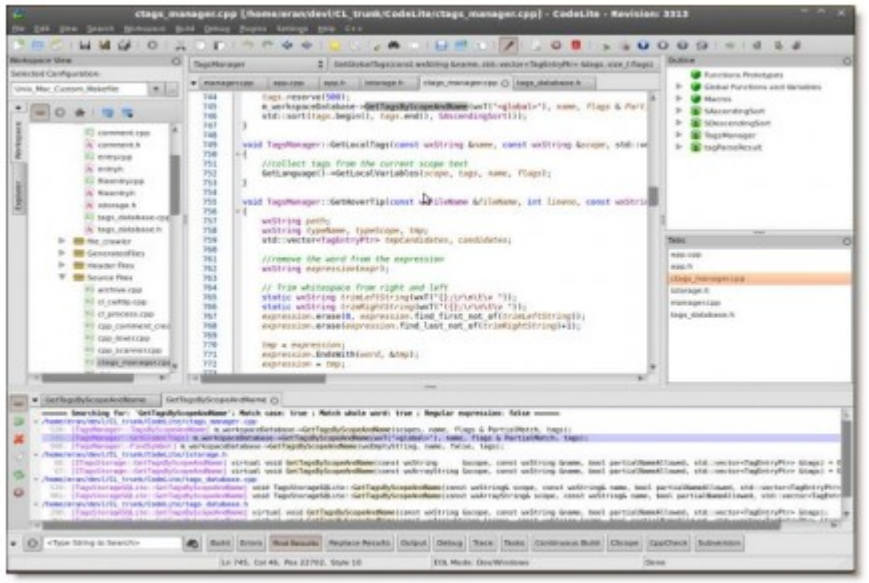
QTCREATOR

- **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**.



CODELITE

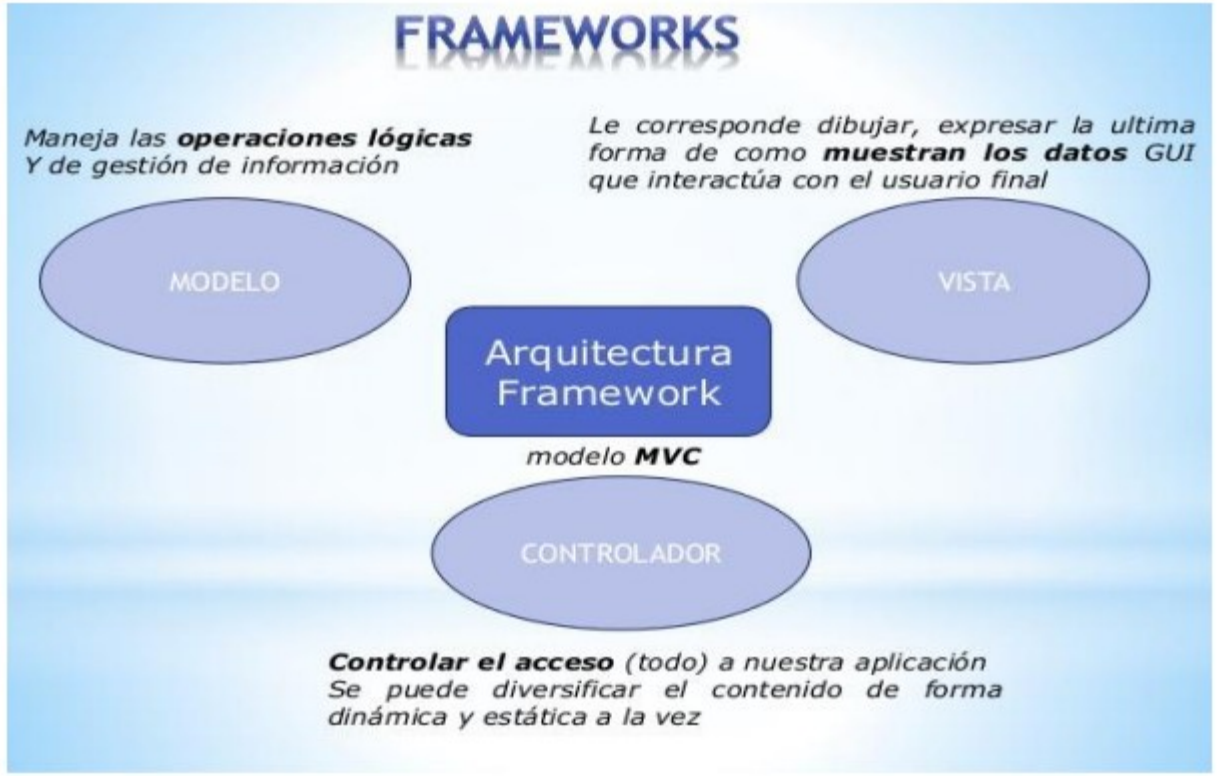
- **CodeLite** es un IDE de código abierto y libre bajo la licencia GNU (General Public License) y para diversos sistemas operativos.
- El entorno de desarrollo integrado usa *wxWidgets* para su interfaz gráfica, ya que al cumplir con la filosofía de código abierto usa herramientas completamente libres.
- Hoy en día soporta los lenguajes *C/C++, PHP y Node.js*.



FRAMEWORK- DEFINICIÓN

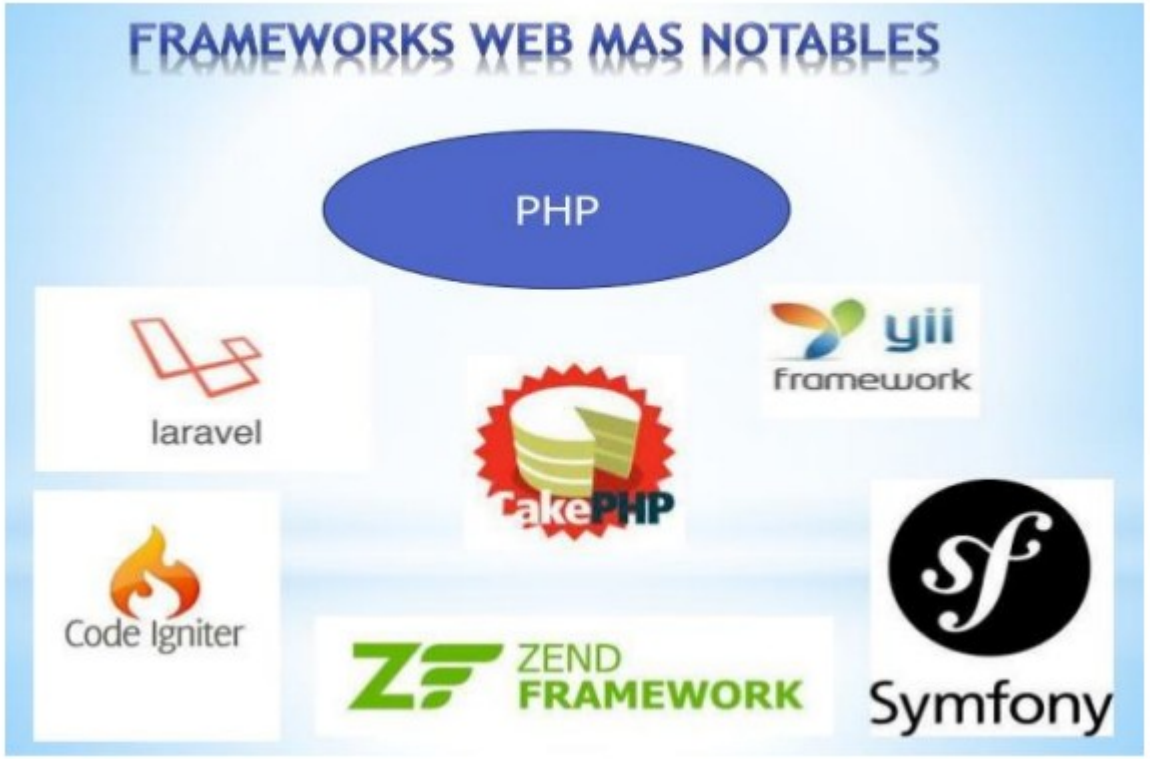
- Herramienta que trabaja con la arquitectura basada en la metodología **modelo, vista y controlador.**
- Un **framework** es una estructura conceptual y tecnológica de soporte definido, normalmente con módulos de software concretos, que puede servir de base para la organización y desarrollo de software.
- Básicamente, es un marco 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 (I)



FRAMEWORKS (II)

- Cada uno de ellos con sus ventajas e inconvenientes



INTERFAZ GRÁFICA

CARACTERÍSTICAS Y COMPONENTES

DEFINICIÓN- CARACTERÍSTICAS

- 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 de **principios generales** 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.

ELEMENTOS BÁSICOS DE UNA INTERFAZ GRÁFICA

- Componentes GUI (widgets)
- Disposición (layout)
- Eventos
- Errores
- Excepciones

ELEMENTOS BÁSICOS (I)

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.

ELEMENTOS BÁSICOS (II)

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***: se trata del gestor del espacio por excelencia. 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 solo una en específico.
- ***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.

ELEMENTOS BÁSICOS (III)

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.

Cada lenguaje de programación tiene su propio modelo de eventos, en Java, por ejemplo, se definen clases auxiliares llamadas escuchadores (listeners) que reciben eventos específicos.

De forma general, y dependiendo del lenguaje de programación existen distintos tipos de eventos, los cuales podríamos agruparlos en:

- Eventos de ventana
- Eventos de teclado
- Eventos de ratón
- Eventos de barras
- Eventos de lista
- Eventos varios

Eventos (I)

- **Eventos de Ventana**: Son los que se generan en respuesta a los cambios de una ventana, un frame o un dialogo.
  - WINDOW_DESTROY
  - WINDOW_EXPOSE
  - WINDOW_ICONIFY
  - WINDOW_DEICONIFY
  - WINDOW_MOVED
- **Eventos de Teclado**: Son generados en respuesta a cuando el usuario pulsa y suelta una tecla mientras un Componente tiene el foco de entrada.
  - KEY_PRESS
  - KEY_RELEASE
  - KEY_ACTION
  - KEY_ACTION_RELEASE

Eventos (II)

- **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
- **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 (III)

- **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

ELEMENTOS BÁSICOS (IV)

- **Errores**: Los errores en la programación suelen ser comunes, ya que es casi imposible programar sin que exista un solo error, inclusive en desarrollos finales existen errores de software, que casi siempre tratan de corregir con parches de seguridad y actualizaciones.
- 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 y que se comprueban mediante las llamadas pruebas unitarias.
- **Existen formas para detectar y corregir errores en un programa por medio de opciones llamadas excepciones.**

ELEMENTOS BÁSICOS (V)

- **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 así resolver la situación de error.

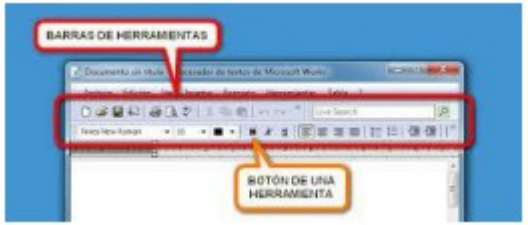
ELEMENTOS TÍPICOS

- 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.

ELEMENTOS TÍPICOS

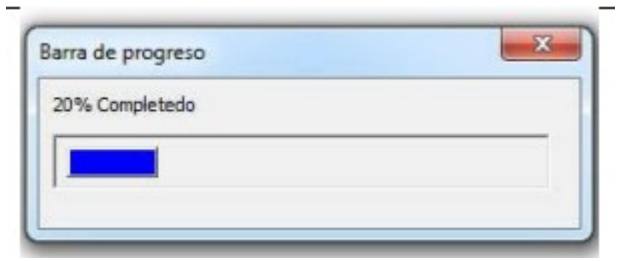
- La **barra de estado** o *statusbar* permite mostrar información acerca del estado actual de la ventana. Generalmente las barras de estado se ubican en la parte inferior de las ventanas.

- 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.

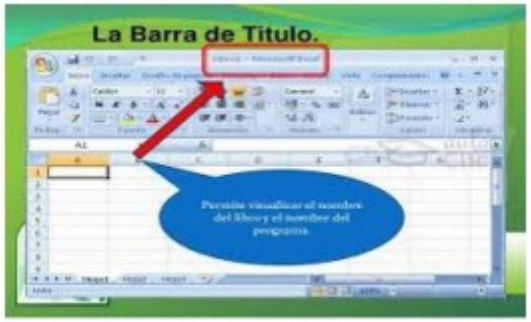


ELEMENTOS TÍPICOS

- Las **barras de progreso** permiten mostrar de forma gráfica el estado de avance de una tarea o proceso.



- 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.



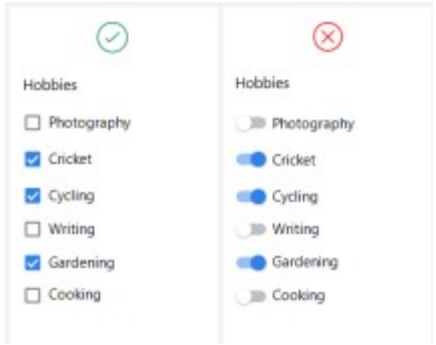
ELEMENTOS TÍPICOS

- 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.

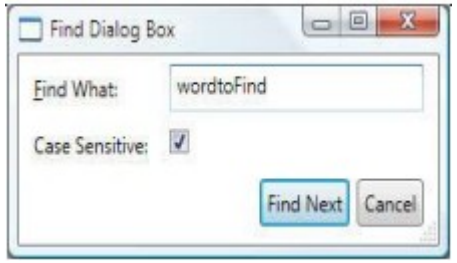
- Los **botones de opción** o *radiobutton* permiten elegir **sólo una opción** de un conjunto predefinido de opciones.

ELEMENTOS TÍPICOS

- Las **casillas de verificación** o *checkbox* permite al usuario marcar **múltiples selecciones** de un número de opciones.

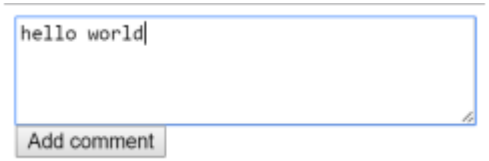


- Un **cuadro de diálogo** o *dialogbox* es una ventana especial para mostrar información al usuario o para obtener de éste una respuesta.



ELEMENTOS TÍPICOS

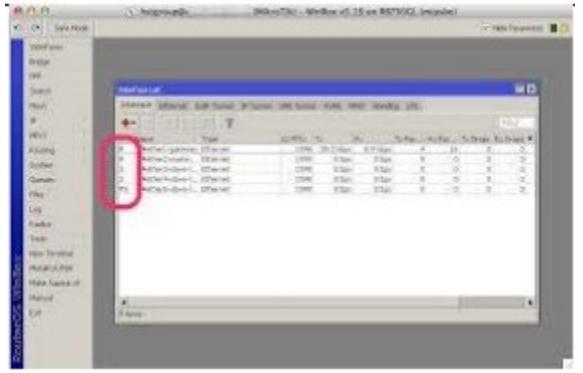
- Un **cuadro de texto** o *textarea* es un elemento típico en las interfaces gráficas en donde es posible insertar texto.



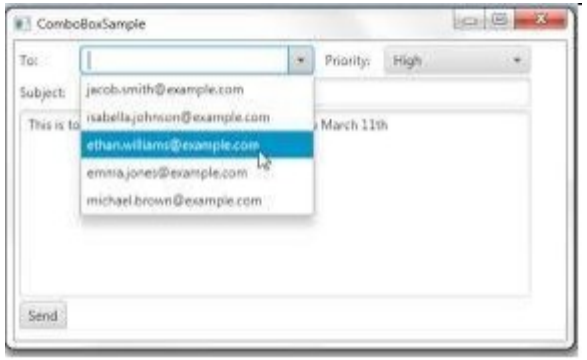
- Los **íconos del sistema** son aquellos íconos que se utilizan para identificar archivos, accesos directos, programas y herramientas del sistema operativo.

ELEMENTOS TÍPICOS

- Una **lista** o *list* es una relacion de datos, ordenados o clasificados segun su tipo. Una evolución de ella que veremos a menudo es el *treeview* como se muestra en la imagen.



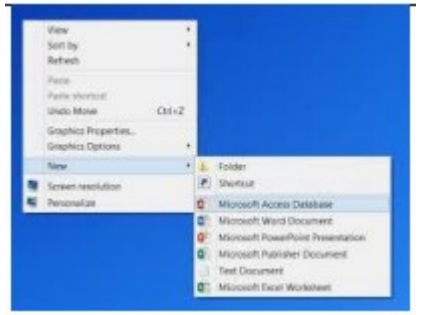
- Una **lista desplegable** o *combobox* al usuario escribir sobre un campo de texto mientras se muestra una lista de opciones.



ELEMENTOS TÍPICOS

- 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. Al menú contextual se accede con el clic derecho sobre algún elemento en la pantalla.



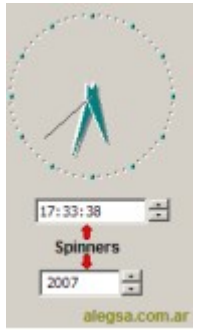
ELEMENTOS TÍPICOS

- 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.

ELEMENTOS TÍPICOS

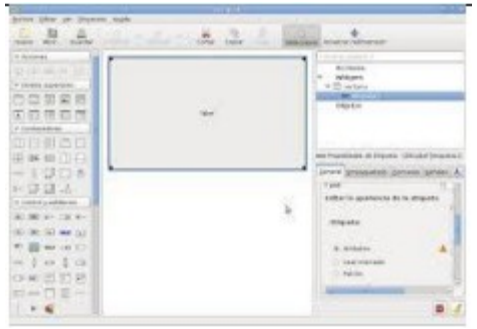
- Un **spinner** es un elemento de las interfaces gráficas que permite al usuario ajustar un valor dentro de un cuadro de texto adjunto a dos flechas que apuntan en direcciones opuestas.



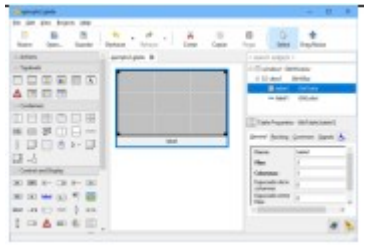
- Un **tooltip** es un elemento de las interfaces gráficas que se emplea junto con el cursor del ratón.

ELEMENTOS TÍPICOS

- 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.



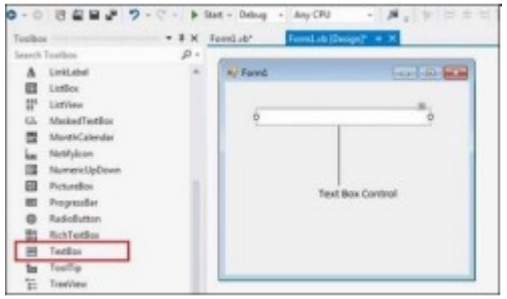
- El **contenedor** es un elemento a modo rejilla o *grid* que facilita el movimiento de diferentes *widgets* de formar unitaria.



ELEMENTOS TÍPICOS

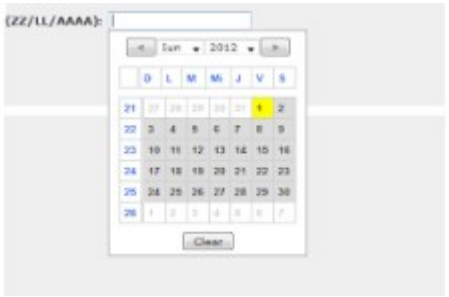
- La **etiqueta** o *label* permite incluir mensajes cortos y de carácter informativo cerca de otros widgets.

- Las **cajas de texto** o *textbox* permiten la introducción de datos que luego se van a procesar.



ELEMENTOS TÍPICOS

- 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

INTRODUCCIÓN

- 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 específicas.
- Veremos:
  - XAML
  - XUL
  - SVG
  - UIML
  - MXML

XAML (I)

- 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>
```

XAML (II)

- Un archivo XAML puede ser compilado para obtener un **archivo binario XAML .baml**, el cual puede ser insertado como un recurso en un ensamblado de Framework .NET. En el momento de ejecución, el motor del Framework extrae el archivo *.baml* de los recursos del ensamblado, se analiza sintácticamente, y crea el correspondiente árbol visual WPF o Workflow.

- Cuando se use en Windows Presentation Foundation, XAML es usado para describir interfaces visuales para usuarios. WPF permite la definición de objetos en 2D y 3D, rotaciones, animaciones y otra variedad de características y efectos.

- 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…)

XAML (III)

- 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>
```

XAML (IV)

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. Puede suscribirse a estos eventos, lo que significa que la aplicación será notificada cuando ocurran y podrá entonces reaccionar a ellos.

```
<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());
}
```

XUL (I)

- **XUL** (acrónimo de ***XML User Interface Language***), lenguaje basado en XML para la interfaz de usuario, desarrollado por **Mozilla**. XUL se implementa como un dialecto XML, lo que permite que las interfaces gráficas de usuario se escriban de manera similar a las páginas web.

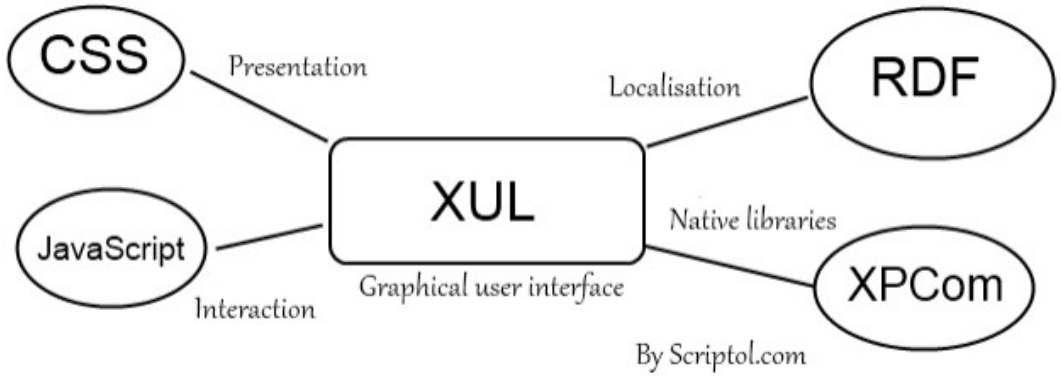
- XUL no es un estándar de facto sino una herramienta desarrollada por Mozilla para sus interfaces. De hecho, Mozilla anunció en su momento que abandonará su desarrollo.

- La principal ventaja de XUL es que aporta una definición de **interfaces GUI** simple y portable. Esto reduce el esfuerzo empleado en el desarrollo de software.

- Otras aplicaciones aparte de Mozilla usan este lenguaje para sus interfaces de usuario. Algunas de ellas usan JavaScript para su lógica. Las aplicaciones XUL tienen la ventaja de poder correr en distintos sistemas operativos.

XUL (II)

- Una interfaz XUL se define mediante la especificación de tres grupos de componentes distintos:
  - **Content**: Aquí se encuentran los documentos XUL, que definen el diseño de la interfaz.
  - **Skin**: Contiene las hojas de estilos (CSS) y las imágenes, las cuales definen la apariencia de la interfaz.
  - **Locale**: Los documentos DTD se encuentran aquí, estos documentos facilitan la localización de páginas XUL.
- XUL provee una gran cantidad de elementos o componentes de interfaz gráfica como botones, menús, etc… al igual que otros lenguajes XML.



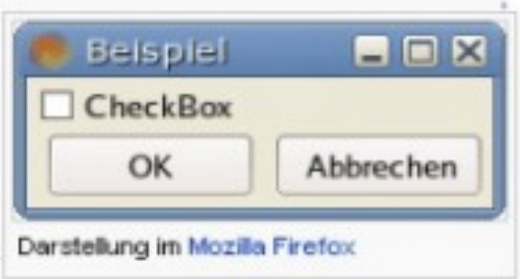
XUL (III)

- Hay cuatro tipos distintos de aplicaciones realizadas con XUL, según como las mismas son creadas:
  - **Extensiones de Firefox**: Son barras de herramientas, menús u otros documentos XUL que agregan funcionalidades a Firefox. Para hacer esto, se usa un elemento definido por XUL llamado “Overlay”, este elemento permite incorporar documentos al navegador mismo.
  - **Aplicaciones independientes**: Estas aplicaciones son creadas mediante **XULRunner**, que es una versión de Mozilla que permite crear aplicaciones XUL independientes. No es necesario el navegador para ejecutar estas aplicaciones, tienen su propio ejecutable.
  - **Paquete XUL**: es un intermedio entre los dos tipos anteriores, es instalado como una extensión, pero actúa como una aplicación separada del navegador. Este tipo de aplicaciones son creadas cuando no queremos utilizar una aplicación XULRunner completa.
  - **Aplicaciones XUL Remotas**: son aplicaciones XUL que se encuentran en un Servidor Web y son ejecutadas remotamente como cualquier otra página web.

XUL (IV)

- Ejemplo

```
<?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

SVG (I)

- 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.

SVG (II)

- 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.

- Debido a su compatibilidad y relación con otros estándares de Web, características como el scripting pueden ser aplicadas a elementos SVG y a otros elementos XML desde distintos espacios de nombre XML simultáneamente y dentro de la misma página web. Un ejemplo extremo de esto es un juego completo de *Tetris* realizado como un objeto SVG.3

- 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.

SVG (III)

Ejemplo

```
<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).

UIML

- Es un lenguaje basado en XML que trata de reducir el trabajo necesario para su desarrollo y para ello se permite describir la interfaz de usuario en **términos declarativos** (es decir, como texto) y **abstracto**. Básicamente que no se especifica exactamente la forma en la interfaz de usuario que se va a ver, sino más bien lo que son los elementos que se muestran, y cómo deben comportarse

```
<part class="DialogMessage" name="HelloWorld"/>
```

- Se puede usar esa descripción para generar interfaces de usuario para diferentes plataformas y dispositivos.

- En la práctica, no tiene un uso muy extendido.

MXML

- MXML es un lenguaje descriptivo desarrollado inicialmente por **Macromedia hasta el 2005** para la plataforma FLEX de Adobe. MXML se basa en XML y su acrónimo *"Macromedia eXtensible Markup Language"*.
- Permite describir interfaces de usuario, crear modelos de datos y tener acceso a los recursos del servidor, del tipo ***RIA (Rich Internet Application).***
- MXML tiene una mayor estructura sobre la base de etiquetas, similar a HTML, pero con una sintaxis menos ambigua, proporciona una gran variedad e inclusive permite extender etiquetas y crear sus propios componentes. Una vez compilado **genera ficheros .swf**.

ACCESO A DATOS

DEFINICIÓN

- 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**.

EJEMPLOS

- 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.

USABILIDAD

DEFINICIÓN

- 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** con un mayor grado de aceptación en los consumidores, ya que se basan en sus necesidades y en la forma en la que los utilizan en la vida diaria. 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.

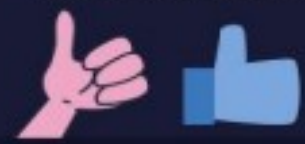
DIEZ PRINCIPIOS DE USABILIDAD HEURÍSTICA

PARA EL DISEÑO DE INTERFAZ DE USUARIO

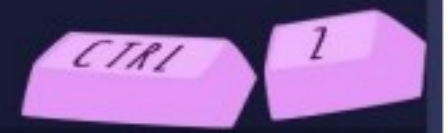
DE JAKOB NIELSEN

Visibilidad del estado del sistema: Da a los usuarios una retroalimentación adecuada.

Correspondencia entre el sistema y el mundo real: Que la información aparezca en un orden natural y lógico.



Control de usuario y libertad: Soporta las acciones deshacer, rehacer y salidas de emergencia.



Prevención de errores: Elimina condiciones propensas a errores y presenta opción de confirmación antes de llevar a cabo una acción.

Coherencia y estándares: Sigue las convenciones de la plataforma. Palabras, situaciones o acciones deben ser consistentes.

Reconocimiento en vez de recordar: Minimice la carga de memoria del usuario haciendo visibles objetos, acciones y opciones.

Flexibilidad y eficiencia de uso: Crea un sistema para usuarios con diferentes niveles de experiencia. Permite a adaptar acciones frecuentes.

Diseño estético y minimalista: No muestres información que sea irrelevante o raramente necesaria.

Ayuda y documentación: Crea una documentación y ayuda guía fácil de utilizar y enriquecer.

Ayudar a los usuarios a reconocer, diagnosticar y recuperar errores: Los mensajes de error deben expresarse en lenguaje sencillo (sin códigos), indicar con precisión el problema y sugerir constructivamente una solución.

PRINCIPIOS (I)

1. **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.

2. **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.

3. **Libertad y control del usuario**

Los usuarios se sentirán más cómodos en tu sistema si les otorgas **libertades y control en su uso**. Debemos ser 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.

PRINCIPIOS (II)

4. **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.

5. **Coherencia y estándares**

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

6. **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.

PRINCIPIOS (III)

7. 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.

8. 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.

9. 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.

10. 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.

ESTANDARES

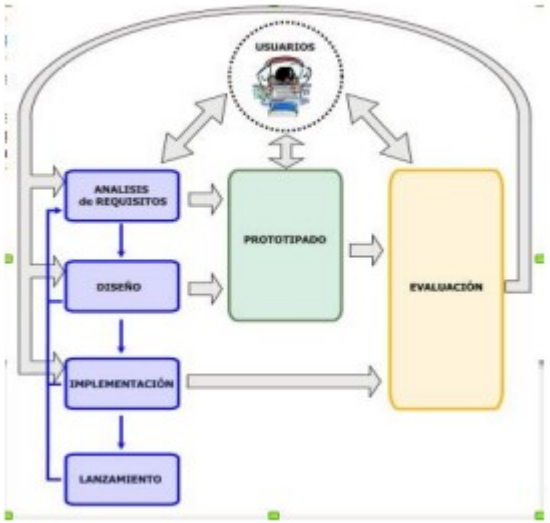
- **ISO/IEC 9126**: La **usabilidad** se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso.
  - Esta definición **hace énfasis en los atributos internos y externos del producto, los cuales contribuyen a su funcionalidad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

INTERACCIÓN PERSONA-ORDENADOR (I)

- Por otro lado, 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.

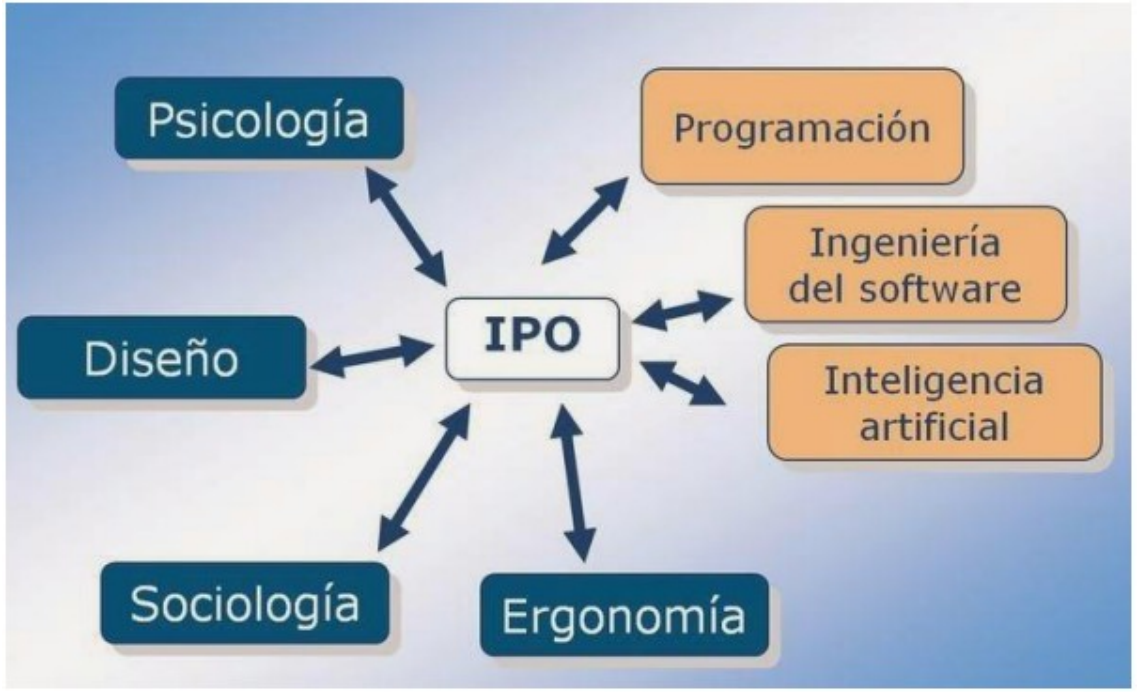
- En este punto introducimos el concepto de **experiencia de usuario (UX)** concepto que se importa del área del marketing y que describe la **relación entre las personas y la tecnología** desde una perspectiva más global y tiene que ser este el principal protagonista del diseño.

- 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.



INTERACCIÓN PERSONA-ORDENADOR (II)

- 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.

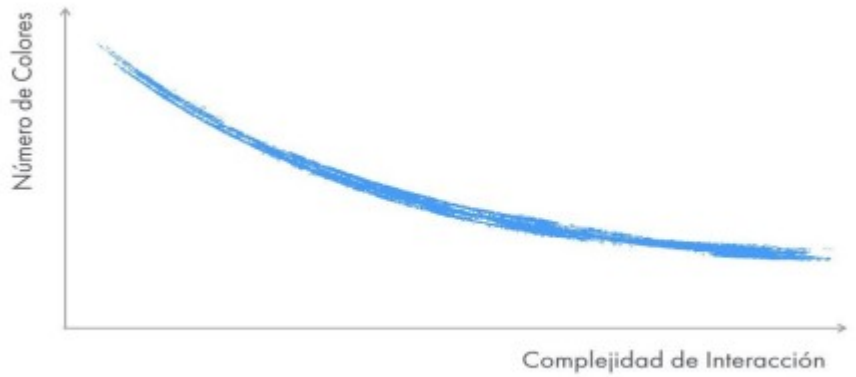


INTERACCIÓN PERSONA-ORDENADOR (III)

- 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.

COLOR (I)

- **La complejidad de la interacción tiende a ser inversamente proporcional a la cantidad de los colores.**



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

COLOR (II)

- 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.

COLOR (III)

- 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.

- Y otra herramienta muy útil en la web que permite evaluar la percepción de las imágenes que se usen en la aplicación, presentando como *feedback*, la imagen **gráfica tal y como será percibida por los usuarios con problemas en la visión** (en particular los afectados de deutanopia, protanopia y tritanopia).

FUENTES (I)

- Según el experto **Oliver Reichenstein** en *Web Design is 95% Typography* la optimización de la tipografía debe perseguir la maximización de la legibilidad, la accesibilidad, el equilibrio gráfico y facilidad de uso.

- 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 34.

FUENTES (II)

- 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 vs. Sans**: tradicionalmente las fuentes con ***serifas*** son usadas para lectura pesada porque esas pequeñas terminaciones **ayudan al ojo a conectar una letra con la siguiente y funciona mejor en impresos**. Por eso la encontramos comúnmente en libros y periódicos. Por otro lado, las fuentes ***sin serifas*** permiten un **reconocimiento más rápido del texto y en términos generales funciona mejor en pantallas**. Por eso reina en el mundo digital. Un gran ejemplo de Serif y Sans es *Medium*. En sus títulos, extractos, cards, menús, etc. Usan Sans pero en el cuerpo de los posts usa Serif.

FUENTES (III)

- 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.

- Estas últimas características son más importantes en las interfaces gráficas web que en las interfaces gráficas de escritorio, aunque si son útiles en los apartados de *Ayuda* que es donde la cantidad de texto es mayor.

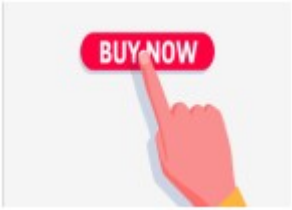
ICONOS (I)

- 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** 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*. Porque no es lo mismo un botón en una app en un smartphone de 4 pulgadas que en una tableta de 10,1 pulgadas. 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.

ICONOS (II)

- 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**.



- Hoy en día existen varias webs donde puedes diseñar botones en formato de imagen que luego puedes añadir a tu interfaz gráfica. En *buttonoptimizer* te permiten diseñar el botón que luego puedes descargar en formato png.

LAYOUT (I)

- 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, por ejemplo, *NetBeans* donde el botón de imprimir no es una de las operaciones principales.

- 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.

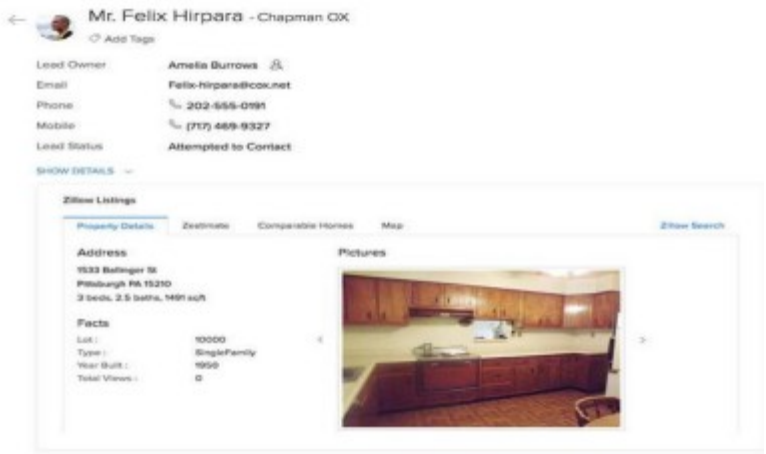
LAYOUT (II)

- 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.

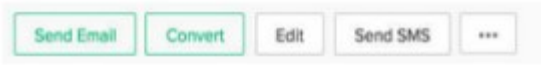
Ejemplo – LAYOUT

- En la imagen, 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. Si se quisiese alinear al centro los datos del empleado habría que ir *widget a widget*, pero si todos estuviesen contenidos en un *container* se podría hacer en un solo paso.



LAYOUT (III)

- Un tipo de contenedor especial es la ***botonera*** que al igual que los contenedores permite agrupar varios botones y disponerlos en un interfaz ya sea en un lateral, parte inferior o parte superior en un solo paso. En este punto hay que señalar que tradicionalmente siempre se colocaron en la parte inferior, pero con el tiempo se han situado también en la parte superior e incluso lateral, especialmente en la parte derecha en las aplicaciones de escritorio y en la izquierda en las webs.



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:

  - Métricas de la Eficiencia
  - Métricas de la Efectividad
  - Métricas de la Satisfacción

Métricas de la Eficiencia

Las **métricas de la Eficiencia** hacen referencia al **tiempo de media que es necesario invertir para completar cada tarea**. Esta es la métrica principal que normalmente se analiza, pero existen otras muchas métricas relacionadas que se pueden medir:

- 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.

Todo lo que **supere 7 errores sobre 10 tareas** 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.

Métricas de la Efectividad

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 78% es aceptable. Con respecto a los problemas de Efectividad, si los usuarios tienen dificultades para completar la tarea, porque se sienten frustrados o incapaces de identificar el siguiente paso, podrían considerarse las siguientes medidas:

- Asegurarse de que **los botones tienen aspecto de botones**. Haz que los clics para seleccionar opciones sean obvios.
- **Simplifica los procesos o flujos de tareas**. Cuantos menos para realizar un proceso mejor. Ello reduce la curva de aprendizaje del uso del sistema incluyendo si es necesario instrucciones introductorias.

Métrica de la Satisfacción

La **métrica 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**. Puedes incluso añadir más preguntas al cuestionario, no obstante, es mejor usar esta escala porque dispone de muchos indicadores con los que realizar comparativas. 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 para mejorar el nivel de satisfacción de los usuarios.

TEST Satisfacción Usabilidad

A continuación 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]*

NORMAS ISO

Presentamos las normas ***ISO*** o estándar existentes **a título informativo** 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** en el desarrollo del prototipo de la aplicación. Por ello requiere una gran planificación y supone un gran coste.
    - **Usability inspection**: en este caso la participación del usuario es testimonial y se echa mano de **expertos en desarrollo y usabilidad** además de consulta de guías de estilo y heurísticas ya establecidas.

PAUTAS DE DISEÑO (I)

Las pautas de diseño se basan en los siguientes parámetros:

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

PAUTAS DE DISEÑO (II)

- **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, dada en función de espaciado y alineación de los elementos de la interfaz.
- **Integración**: Se mide la **relación entre el diseño visual de la aplicación y las aplicaciones del sistema u otras aplicaciones del entorno** gráfico con las que se utiliza.

DIRECTRICES GENERALES

Veamos algunas **directrices generales:**

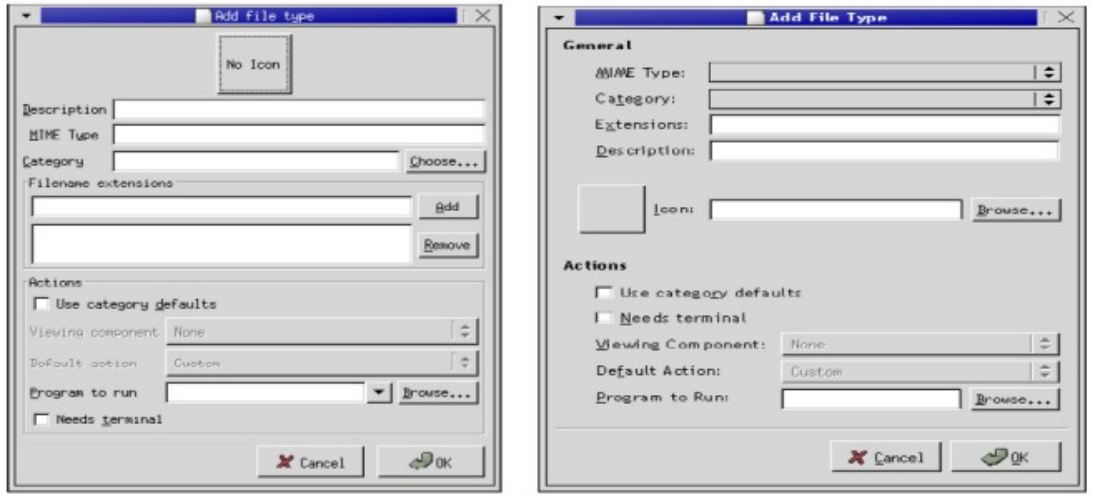
- Cuadros de diálogo
- Barra de menús
- Cuadros de texto
- Objeto Botones de comando
- Objeto Botones de opción
- Objeto Botones de chequeo
- Objeto Cuadro combinado
- Objeto Lista
- Objeto Tabbed (pestañas)
- Objeto Marcos y Separadores

Estas directrices aunque no suelen ser seguidas por muchos programadores, convendría considerarlas como recomendaciones.

CUADROS DE DIÁLOGO

Cuando un usuario revisa un cuadro de dialogo complejo (que contiene muchas etiquetas, cuadro de texto, botones varios, etc.), es fácil ver, como esta interfaz puede convertirse en un estorbo cuando el *layout* es deficiente.

En la siguiente imagen, 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.



Pésima distribución **Óptima distribución**

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**.

- No abusar de los bordes.
- 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 (I)

Proporciona acceso a un subconjunto de menús 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**.
- Finalmente evitar menús que se oculten.

BARRA DE MENÚS (II)

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**. Son llamados en algunas referencias bibliográficas también como *Drop-Down o Pull-Down* menú.

- 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.
- No superar más allá de **15 sub-items** en un mismo ítem. En caso necesario reorganizar los ítems de grado superior.
- Evitar la creación de nuevos ítems en tiempo de ejecución.
- Usad combinación de teclas en los más frecuentes.

BARRA DE MENÚS (III)

BARRA DE MENÚS (IV)

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

- Se debe aplicar su uso **solo en casos necesarios**.
- Su utilización hace difícil la navegación entorpece la búsqueda de los ítems contenidos en él.
- 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).
- **Evitar más de dos niveles de jerarquía** ya que son difíciles de memorizar y navegar.

BARRA DE MENÚS (V)

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.

- Se debe ser cuidadoso con su aplicación, ya que este tipo de 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.
- Deben ser lo más simples posibles para maximizar su eficiencia. Colocar un **máximo de 10 ítems**.
- **Evitar el uso de menú de cascada** dentro de los menús contextuales.
- Ordenar los ítems de acuerdo con el criterio lógico, operativo o funcional según convenga.
- Usar **la línea como separador gráfico** para denotar agrupación de opciones relacionadas.

CUADROS DE TEXTO (TextBox o Entry) (I)

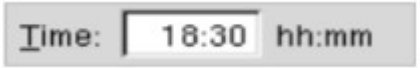
Son usados para ingresar una o más líneas de texto plano.

- 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**. Esto es especialmente útil cuando en una ventana el usuario quiera comparar dos valores numéricos en la misma columna de controles.
- 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. **No colocar del mismo ancho en todos** los campos de entrada.

CUADROS DE TEXTO (TextBox o Entry) (II)

- 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 adicionales** antes de ejecutarse la acción.
- **No aplicar más de una o dos anchuras diferentes para botones** en una misma ventana y todos los botones **deben tener la misma altura**. Esto dará una apariencia visual uniforme a la ventana que la hará más fácil de usar.
- **No asociar acciones a los eventos Doble-Clic ni Clic-Derecho** de un botón de comando.
- En las ventanas de diálogo, **trate de no asignar botones por defecto** para las respuestas.

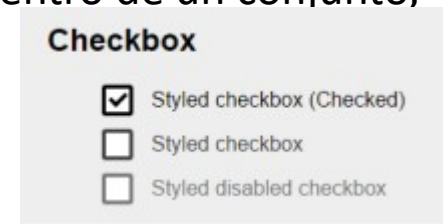
OBJETO BOTONES DE OPCIÓN

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.
- Rotular cada botón en particular en combinación de **mayúsculas de oración**.
- El número de elementos **para botones de opción no debe exceder de 8**, si se sobrepasa este valor evaluar la posibilidad de aplicar listas desplegables simples.
- Trate de alinear los botones de opción verticalmente, esto **contribuye a hacer más fácil** la revisión visual de la ventana.

OBJETO BOTONES DE CHEQUEO (CheckBox)

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

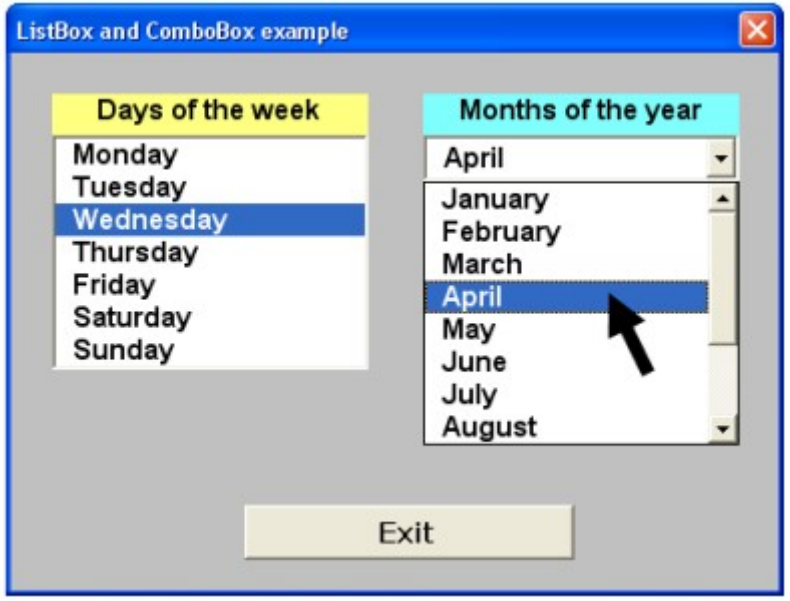


- Si la selección de un botón de chequeo afecta a otro control, **ubicarlo inmediatamente encima o al lado izquierdo del control que es afectado**.
- Usar combinación de **mayúsculas de encabezado para rotular el grupo botones de chequeo**, ubicando la etiqueta arriba o a la izquierda.
- Aplicar a los botones combinación **de mayúsculas de oración**.
- El número de elementos **para botones de chequeo no debe exceder a 8**, si se sobrepasa evaluar la posibilidad de aplicar listas desplegables con opción de marcas de chequeo.
- Tratar de alinear **los botones de chequeo verticalmente** ya que facilita la visibilidad.

OBJETO CUADRO COMBINADO (Combo Box)

Son listas desplegables usadas para brindar al usuario la capacidad de selección dentro de un conjunto de valores dados en la interfaz a través de una lista.

- 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 de valores dados en la interfaz a través de una lista.

- Se debe rotular con una etiqueta colocada **arriba o a la izquierda del control**, usando combinación **de mayúscula de encabezado**.
- Aplicar a los elementos de la lista, combinación de **mayúsculas de oración**.
- En la interfaz, diseñar la lista para que **muestre al menos cuatro ítems** a la vez sin realizar *scrolling*. Para listas de 10 o más ítems se debe adecuar al tamaño de manera apropiada.
- **No diseñar listas con menos de 5** ítems.
- Sólo usar encabezado de columnas cuando la lista tenga más de una columna y/o la lista tenga solamente una columna, pero exista la posibilidad de reordenamiento en ejecución.
- Para listas de selección múltiple **se debe mostrar el número de ítems** actualmente seleccionados en un texto estático debajo de la lista. Por ejemplo: "*Número de Item Seleccionados: 5*". Esto hace más evidente que la selección múltiple es posible.
- Considerar la **posibilidad de proporcionar botones** "*Seleccionar Todo*" "*Deseleccionar Todo*" al lado de la lista de selección múltiple si es apropiado.
- La selección de un ítem de la lista **no debería iniciar** ninguna acción en la aplicación.
- Alinear los **botones de opción verticalmente**, esto contribuye a hacer más fácil la revisión visual de la ventana.

OBJETO TABBED

Es un objeto adecuado para presentar información relacionada en la misma ventana, sin tener que desplegar otras al mismo tiempo. Es unos de los elementos más utilizados hoy en día en la organización y presentación de la información en una aplicación.

- **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

Un marco es un cuadro con título que puede dibujarse **alrededor de un grupo de objetos o controles** de la interfaz **para organizarlos como grupos funcionales**. Un separador **es una línea simple**, horizontal o vertical que pueden usarse para dividir la ventana en grupos funcionales. Antes de agregar marcos y separadores gráficos evaluar la posibilidad de diseñar con *identación* y espaciado para reflejar la relación entre controles, ya que suele ser 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.