

DISEÑO DE INTERFACES

TEORIA

INFORMES

INTRODUCCIÓN

- Al planear la creación de una aplicación una de las consideraciones más importantes es la posibilidad de la inclusión de informes que permitan al usuario acceder a la información de una forma estructurada, sea en formato papel o visualización en pantalla.
- Un informe suele generarse a partir de un **diseño previo** o **plantilla** en el cual se vuelca 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 precisarán 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**.

DEFINICIÓN

- El **informe incrustado** se importa del proyecto o directamente se ha creado en él. 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.
- Por otro lado, **un informe no-incrustado** 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 y se puede tener acceso a él de diversas formas.
- El informe puede estar en un directorio de archivos en la unidad local. También puede estar expuesto como servicio Web de informes y obtenerse en formato HTML o XML. También puede formar parte de un grupo de informes creados a través de herramientas externas como *Crystal Reports* o *ReportLabs*, entre otras.
- La elección entre uno y otro depende de la aplicación, pero los informes **no-incrustados** elaborados por herramientas independientes están prevaleciendo. Son más sencillos y seguros, **aunque requieren de mayor trabajo en su diseño**.

Aplicaciones para la elaboración de informes (I)

- **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 como VisualNet, Eclipse o NetBeans. El formato de salida puede ser *pdf*, *.rpt*, *.xls*, *xml*, *.txt* y *.CSV* entre otros.
 - El **formato CSV** 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.
 - CrystalReport puede considerarse como una herramienta potente en la elaboración de informes. Pero su complejidad por sus enormes posibilidades hace que muchos programadores sean reacios a utilizarla.

Aplicaciones para la elaboración de informes (II)

- **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, incluyendo J2EE o aplicaciones web, para generar contenido dinámico. 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 entorno de desarrollo Visual Net de Microsoft.





ESTRUCTURA GENERAL DE UN INFORME

- De forma general un informe y aunque no todas se las incluyen, contiene las siguientes **secciones**:
- **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 conforman el reporte. Es allí donde se alojan los campos del origen de datos y **es la parte más dinámica**.
- **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**.

FILTROS

- **Los filtros limitan los datos que se muestran** al usuario tras la recuperación de todos los datos.
- Dado que se recupera el conjunto completo de datos y luego se filtra cuando se procesa el informe, es posible que el informe no sea tan bueno como cuando el informe obtiene datos filtrados de otra manera (específicamente, si escribe un código que filtra datos antes de que pasen al informe). De esta forma en **los agrupamientos pueden ser interesantes los filtrados de datos**.

Arrastrar campos entre las áreas siguientes:

 Filtro de informe	 Etiquetas de columna
País ▼	Producto ▼
<hr/>	
 Etiquetas de fila	 Valores
Ciudad ▼	Suma de Ventas ▼
<hr/>	
<input type="checkbox"/> Aplazar actualización del diseño	<input type="button" value="Actualizar"/>

Ejemplo

- **Por ejemplo, listado de empleados por departamentos** que obtuvieron un mínimo de X ventas o que pertenezcan a la ciudad Y.

The image shows a screenshot of a report design tool interface. The report is titled "Libreta de teléfonos del cliente". The layout is divided into several sections:

- Encabezado del informe**: Contains the report title "Libreta de teléfonos del cliente".
- Encabezado de página**: Contains the text "File As: Contact Name" and "Teléfono".
- Archivo como encabezado**: Contains the text "=UC".
- Detalle**: Contains the text "ID", "Nombre del contacto", and "Teléfono".
- Pie de página**: Contains the text "Page".
- Pie del informe**: Contains the text "Page".

Encabezado del informe		
Libreta de teléfonos del cliente		
File As: Contact Name		
Teléfono		
=UC		
ID	Nombre del contacto	Teléfono
Page		
Page		

ELABORACIÓN

- 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 es 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 de 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, y 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

INTRODUCCIÓN

- Un **fichero de ayuda** es un documento sobre papel o en formato digital cuya finalidad es servir **de guía de referencia, manual o** como su nombre indica **ayuda** a los diferentes usuarios de una aplicación incluidos los desarrolladores, en especial, a aquellos posibles desarrolladores que puedan en un futuro modificar dicha aplicación.
- Los formatos de ayuda, ciñéndonos a soporte digital, pueden ser en **pdf o cualquier formato ofimático** que cada vez son menos habituales. Pueden ser abiertos por cualquier visor pdf en el primer caso o bien herramientas ofimáticas de carácter general en el segundo caso.
- Luego están los que se encuentran incrustados en la propia aplicación.

Winhelp

- **Winhelp (.hlp)** son archivos con extensión **hlp** de ayuda de Windows (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 de ayuda 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

- **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 publicó por primera vez en 1997 como sucesor del sistema de ayuda Winhelp. 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 HTML hiperenlizadas 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 Assistance 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**.

Ejemplo



- Los archivos .chm a veces se utilizan como e-books.

MAML

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

En Linux

- **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 específicos de distribución*, *archivos de ejemplo*, *etc*, *de la aplicación y/o utilería que estemos estudiando*. Es en esta sección donde la mayoría de los programas utilizados en Linux guardan sus documentos de ayuda.

Herramientas de generación de ayudas (I)

- Si las ayudas están generadas en formatos .pdf u otro formato ofimático se utilizarían los editores de texto o documentos habituales.
- En el caso de que los formatos sean otros entonces tenemos:
- **HelpMaker**: es una aplicación para crear archivos de ayuda para programas. Permite la creación de archivos de ayuda enteros; en diferentes formatos, tales como: *WinHelp*, *RTF (texto enriquecido)* y *HTML-Help*. El programa permite crear la estructura de la ayuda, pudiendo cambiarla, ampliarla, editarla, añadir vínculos (incluso de una zona a otra de la misma ayuda), etc. También cuenta con un corrector ortográfico, que vendrá muy bien si haces las ayudas también en inglés.
- **DocBuilder** es una aplicación capaz de generar archivos de documentación y ayuda en diversos formatos para el software desarrollado por ti mismo. El programa reconoce varios lenguajes de programación: C/C++, Pascal, Delphi. Y puede generar documentación en RTF, HTML y archivos Windows de ayuda. Esta herramienta examina la estructura del código fuente del programa, distinguiendo los comandos de los comentarios sobre el software, analizando estos últimos y distribuyéndolos de acuerdo con el formato elegido. Es de las más antiguas.

Herramientas de generación de ayudas (II)

- **Microsoft HTML Help Workshop:** permite crear ficheros de ayuda de windows (HLP) y páginas web que utilicen controles de navegación. Crea estos ficheros y los distribuye con las aplicaciones. Incluye un administrador de proyectos, un compilador de ayuda y un editor de imágenes. HTML Help Workshop ofrece algunas ventajas sobre el estándar HTML, incluyendo la habilidad de implementar una tabla de elementos combinada y un índice, así como el uso de palabras clave para **capacidades avanzadas de hiperenlazado**. El compilador permite comprimir HTML, gráficos y otros ficheros en un fichero compilado chm relativamente pequeño, que puede ser distribuido junto a la aplicación o bien descargado desde Internet. También se incluyen un control ActiveX y un applet de Java. El primero puede ser usado en cualquier navegador con soporte ActiveX, o en HTML Help Workshop, y el applet de Java puede ser utilizado en cualquier navegador que soporte Java. Con ambos controles puedes crear páginas web con soporte de ayuda HTML, incluyendo tablas de contenido, índices, y temas relacionados. Asimismo, crea ficheros compilados de ayuda para ser ejecutados directamente desde el web.
- **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). Para poder visualizar un fichero chm en Linux existen diferentes herramientas. En Gnome por ejemplo, está xchm.

TIPOS DE MANUALES

- 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 construcción de este** y finaliza justo antes de la entrega del programa o aplicación al cliente. Así mismo, la documentación que se entrega al cliente tendrá que coincidir con la versión final de los programas que componen la aplicación.
- Una vez concluido el programa, los documentos que se deben entregar son una guía técnica, una guía de uso y de instalación.

DOCUMENTACIÓN

- La documentación que se entrega al cliente se divide claramente en dos categorías, **interna y externa**:
 - **Interna**: Es aquella que **se crea en el mismo código**, ya puede ser en forma de comentarios o de archivos de información dentro de la aplicación.
 - **Externa**: Es aquella que se escribe en cuadernos o libros, totalmente ajena a la aplicación en sí. Dentro de esta categoría también se encuentra la **ayuda electrónica**.

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 personas con conocimientos de informática, generalmente programadores. El principal objetivo es el de facilitar el desarrollo, corrección y futuro mantenimiento de la aplicación de una forma rápida y fácil.
- Esta guía 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.

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.
- Un punto para tener en cuenta en su creación es que no debe hacer referencia a ningún apartado de la guía técnica y en el caso de que se haga uso de algún tecnicismo debe ir acompañado de un glosario al final de 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, es decir, sus funciones básicas.
 - Una sección de solución de problemas que detalla los posibles errores o problemas que pueden surgir, junto con la forma de solucionarlos.
 - Una sección de preguntas frecuentes. (FAQ)
 - Dónde encontrar más ayuda, y datos de contacto.
 - Un Glosario y, para documentos más grandes, un índice.

GUÍA DE INSTALACIÓN

- Es la guía que contiene la información necesaria para **implementar dicha aplicación**.
- Dentro de este documento se encuentran las instrucciones para la puesta en marcha del sistema y las normas de utilización de este.
- Dentro de las normas de utilización se incluyen también las normas de seguridad, tanto las físicas como las referentes al acceso a la información, así como, comportamiento de la función, los valores que devuelve, las excepciones que arroja y cualquier otro detalle que consideremos relevante.

DISTRIBUCIÓN DE APLICACIONES

- El **empaquetado de aplicaciones** consiste en proporcionar a los futuros usuarios las aplicaciones en forma de paquetes, 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**, tanto de forma **dinámica** como **estática**.
- El usuario percibe el paquete como un conjunto que representa al programa en sí, cuando en realidad incluye varios ficheros.

- Una de las mayores ventajas de un correcto empaquetado de aplicaciones es que permite **evitar** los **problemas de las dependencias** tanto a la hora de **instalar** la aplicación como a la hora de **usarla**, ya que cada paquete lleva los ficheros necesarios, y la instalación o desinstalación de otro software no va a afectar a las dependencias de dicho paquete.
- Sin embargo, como desventaja estos paquetes ocupan mucho más espacio en el disco, especialmente si el paquete incluye bibliotecas.

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

- **Linux:** tenemos dos tipos de paquetes que sobresalen del resto:
 - **rpm:** (Redhat Package Manager) de la familia de RedHat (RHEL, Fedora, CentOS), Mandriva, Suse
 - **deb:** de la familia de Debian (Debian, Ubuntu, y derivados)
- **Windows:** tomando como base su IDE Visual Studio el formato de empaquetado es:
 - **msi:** se definen como instaladores de Microsoft.

- Estos paquetes de software contienen la información necesaria para automatizar su instalación, minimizando la intervención manual del usuario, ya que toda la información irá 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 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 instale antes de ponerse a usar el equipo. El Active Directory configurado con la herramienta Samba4 de Linux es capaz de gestionar este tipo de archivos.
- 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. Windows Installer identifica a los paquetes por un GUID. El paquete está compuesto de componentes agrupados dentro de sus características.

LINUX (I)

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 -xzf archivo.tar.gz //para descomprimir el paquete
```

Si está en .tar.bz2 escribimos:

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

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

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

Lo que estamos haciendo es 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. Situación que se da habitualmente con lo que se aconseja los métodos siguientes.

LINUX (II)

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

Los paquetes *.deb* son paquetes que se pueden instalar en la distribución Debian y derivados (Ubuntu, Kubuntu...). Los *.rpm* (RedHat Package Manager) son los de la distribución Red Hat y derivados (OpenSuse, Mandriva, 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 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 que facilitan bastante su creación. Uno de ellos es **checkinstall**. 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.

- *Cada una tiene un número a su izquierda, así que para editarla solo escribimos su número. No es necesario modificar todas, pero si las más importantes como **Maintainer, Summary, Name y Version**.*
- *Una vez hayamos modificado lo que queremos, presionamos [Enter] (sin ningún número previo) y comenzará a compilar e instalar el paquete. Cuando haya terminado, en el directorio donde compilamos habrá aparecido un paquete **.deb** de la aplicación, listo para instalar. Asegurarse en “Version”, que no haya letras. Eso suele evitar que se cree el paquete. Es posible que “Requires” dé un fallo, con dejar el espacio en blanco suele arreglarse.*
- Aunque resulte una mecánica sencilla, la **aparición de errores suele ser habitual** ya que 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 creará un entorno de trabajo para crear los RPM's o "Árbol del Proyecto".

Como se puede observar el proceso es mecánico, pero como en el caso anterior con alta probabilidad de error. 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 (I)

- Una de las herramientas “ocultas” de Windows es ***lexpress***. 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 Windows 8.
- *lexpress.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.
- Los archivos pueden ser modificarse con cualquier texto plano / editor ASCII, como el Bloc de notas. Todos los archivos autoextraíbles creados por el uso de ***lexpress*** se comprimen con el MakeCab (MAKECAB.EXE) herramienta, y se extraen mediante el Wextract (wextract.exe) de Microsoft.

WINDOWS (II)

- `lexpress.exe` se encuentra en la carpeta `System32` de Windows. La interfaz frontal (IExpress Wizard) se puede iniciar manualmente escribiendo `IExpress` en la ventana Ejecutar del menú Inicio. También se puede utilizar desde la línea de comandos (o archivo por lotes en la consola) para crear paquetes de instalación personalizada

```
C:> IEXPRESS / N letraDeUnidad \ directory_name \ file_name.SED
```

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

DESINSTALACIÓN

- La **desinstalación de software** es el proceso de revertir los cambios producidos en un sistema por la instalación de este.
- 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.
- Cada vez es más difícil la desinstalación, dado que muchas bibliotecas se comparten entre aplicaciones de diferentes productores de software que utilizan enlaces duros o simbólicos a través del directorio.
- En sistemas de alta complejidad, el esfuerzo para desinstalar un programa puede ser mayor que el de la instalación.

- En Windows la desinstalación mejora si utilizamos software que además de eliminar los archivos del software **elimine las entradas en los registros**. Esto es debido a que el desinstalador que lleva Windows (*Agregar y Quitar Programas*) no es muy fiable en este 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 ATOMATIZACIÓN DESCARGA

- 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 información importante, además del software mismo, como pueden ser el nombre completo, una descripción de su funcionalidad, el número de versión, el distribuidor del software, la suma de verificación y una lista de otros paquetes requeridos para el correcto funcionamiento del software. Esta metainformación se introduce normalmente en una base de datos de paquetes local.

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

- Comprobación de la suma de verificación para evitar que haya diferencias entre la versión local de un paquete y la versión oficial.
- Comprobación de la firma digital.
- Instalación, actualización y eliminación simple de paquetes.
- Resolución de dependencias para garantizar que el software funcione correctamente.
- Búsqueda de actualizaciones para proveer la última versión de un paquete, ya que normalmente solucionan bugs y proporcionan actualizaciones de seguridad.
- Agrupamiento de paquetes según su función para evitar la confusión al instalarlos o mantenerlos.
- Muchos de los sistemas de gestión de paquetes ampliamente utilizados utilizan backends simples para instalar los paquetes. Por ejemplo, YUM utiliza RPM como backend y APT utiliza dpkg.

- Algunos de los sistemas de gestión de paquetes más avanzados tienen la capacidad de desinstalar los paquetes recursivamente o en cascada, de forma que se eliminan todos los paquetes que dependen del paquete a desinstalar y todos los paquetes de los que el paquete a desinstalar depende, respectivamente.
- 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.
- Estos problemas, como fallos al reiniciar el sistema o incluso hacer que el sistema no arranque, pueden ocasionarse cuando el formato de los ficheros de configuración cambia. Por ejemplo, cuando el antiguo fichero de configuración no deshabilita nuevas opciones que deberían ser deshabilitadas. Algunos sistemas de gestión de paquetes, como el dpkg de Debian, permiten *reconfigurar* el software durante la instalación. En cualquier otra situación es preferible instalar los paquetes con la configuración por defecto y sobrescribirla posteriormente.
- El software normalmente se pone a disposición de los usuarios en los repositorios, con el fin de proporcionar a los usuarios de un sencillo control sobre los diferentes tipos de software que van a instalar en su sistema y, en ocasiones, debido a razones legales o conveniencias por parte de los distribuidores.

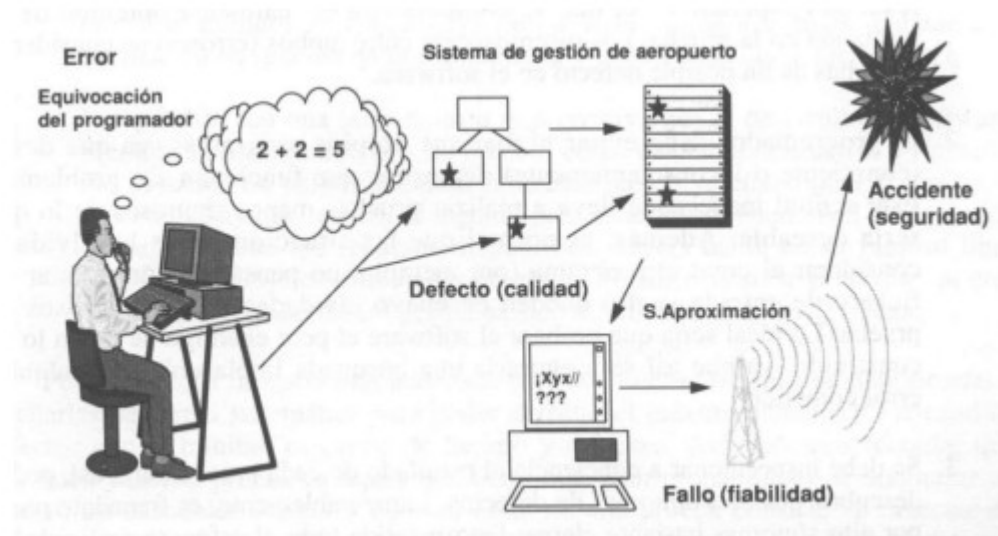
- Existen también los **CVS, o sistema de control de versiones**, como **GitHub** que permiten la distribución de software a terceros.
- Finalmente, algunos lenguajes de programación interpretados tienen su propio sistema de gestión de paquetes para manejar módulos del lenguaje, como pasa con los lenguajes de programación **Perl (CPAN), Python (pip), PHP (PEAR) o Ruby (RubyGems)**. Otros programas pueden venir con su propio sistema para gestionar módulos.
- **Snap** es un nuevo concepto que ha llegado junto a (o que ha sido introducido en) 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. Sus sistemas de detección de dependencias son más avanzados que el de apt.
- **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

INTRODUCCIÓN (I)

- Las **pruebas de software** son investigaciones empíricas y técnicas cuyo objetivo es **proporcionar** al futuro usuario y/o comprador **información objetiva e independiente sobre la calidad del producto desarrollado**. Son una actividad más en el proceso de control de calidad, siendo una etapa más dentro del desarrollo de software.



INTRODUCCIÓN (II)

- 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 de suma de dos números es imposible probar 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**, es decir, **descubrir un error es el éxito de una prueba**. El hallazgo de un error o defecto no implica que seamos malos profesionales ya que todo el mundo comete errores. Es más, el **descubrimiento de un defecto significa un éxito para la mejora de la calidad del producto**.
- Por otro lado, el **proceso de verificación** es el proceso de evaluación de un sistema o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase, es decir, responde a la pregunta: “¿estamos construyendo el producto correctamente?”.
- Finalmente 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?”

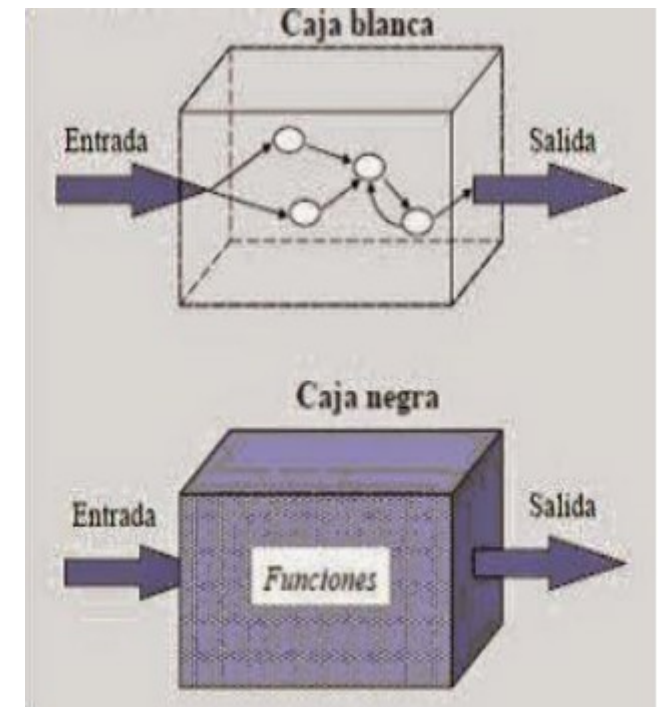
DEFINICIONES (I)

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”.
- **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 medido y el valor verdadero**, especificado o teóricamente correcto. Por ejemplo: “la suma es incorrecta”.

DEFINICIONES (II)

- 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 las 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. Se examina así **la lógica interna del programa sin considerar los aspectos de rendimiento**.



OBJETIVO

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

Formas de Calcular la Complejidad Ciclomática $V(G)$

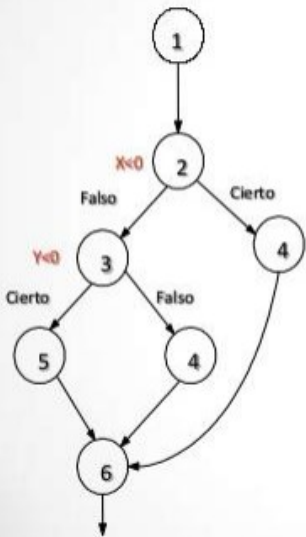
- $V(G) = a - n + 2$
- $V(G) = r$
- $V(G) = c + 1$

Donde

- a : # de arcos o aristas del grafo.
- n : # de nodos.
- r : # de regiones cerradas del grafo.
- c : # de nodos de condición.

Ejemplos

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

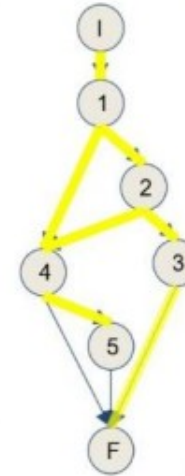
Casos de prueba para cada camino:

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

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

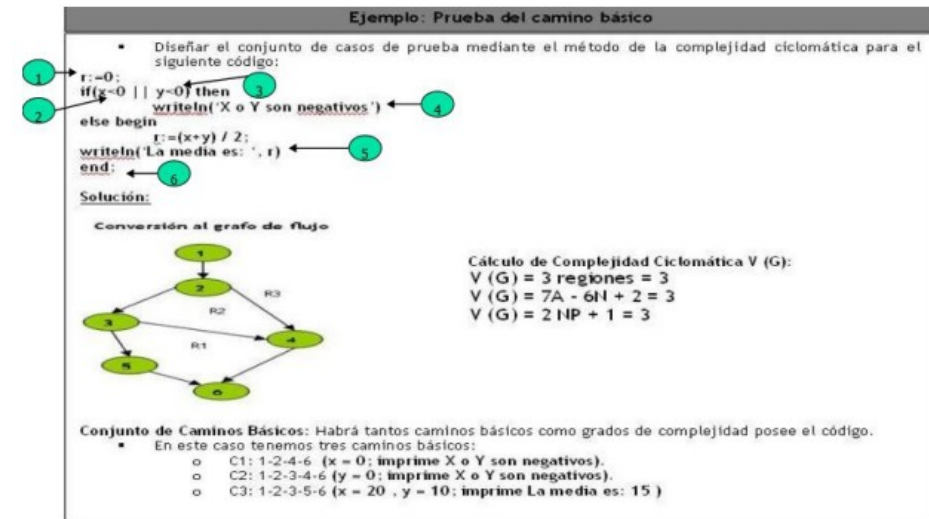
Camino 3: $x=4, y=-3, rdo=0$, error

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



$$V(G) = \text{Aristas} - \text{Nodos} + 2$$

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



RECOMENDACIONES/CONSEJOS (I)

- 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.
- No deben hacerse planes de prueba suponiendo que, prácticamente, no hay defectos en los programas y, por lo tanto, dedicando pocos recursos a las pruebas. Siempre hay defectos.
- Probar si el software no hace lo que debe hacer.
- Probar si el software hace lo que debe hacer, es decir, si provoca efectos secundarios adversos.

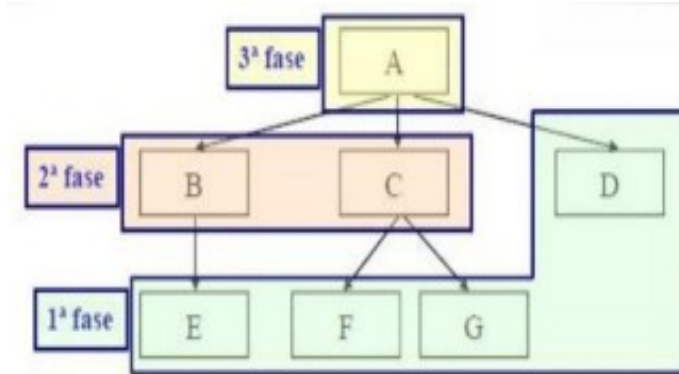
RECOMENDACIONES/CONSEJOS (II)

- 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.

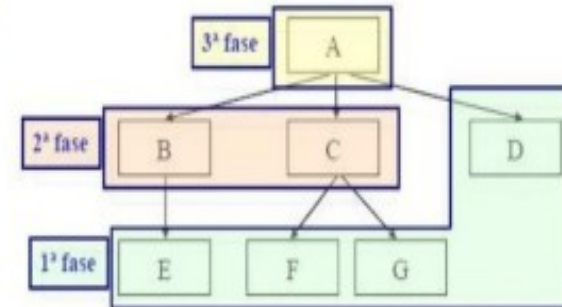
PRUEBAS DE INTEGRACIÓN

- 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 individualmente 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 provocaron.
- 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 la **descendente**.

Ejemplo – Integración Incremental Creciente vs Decreciente



- Incremental Ascendente (Bottom-Up)
- Unitarias de E, F, G y D
- Integración de (B con E), (C con F) y (C con G)
- Integración de (A con B), (A con C) y (A con D)



- Incremental Descendente (Top-Down)
- Primero en **profundidad**, completando ramas del árbol (A, B, E, C, F, G, D)
- Primero en **anchura**, completando niveles de jerarquía (A, B, C, D, E, F, G)

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**.

TIPOS de PRUEBAS de SISTEMA (I)

Se distinguen los siguientes tipos de pruebas:

- **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.

TIPOS de PRUEBAS de SISTEMA (II)

- **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.** Visto en una unidad anterior
- **Pruebas de almacenamiento.** Comprobación de la cantidad de memoria principal y secundaria que el programa usa y el tamaño de los archivos temporales.
- **Pruebas de configuración.** Aunque a veces resulta imposible se debe comprobar el programa con cada tipo de hardware, sistema operativo, antivirus....
- **Pruebas de instalación.** En especial la automatización para facilidad del usuario final.
- **Pruebas de la documentación.** Hace referencia tanto a la documentación técnica para desarrolladores futuros como a la documentación para el usuario.

PRUEBAS DE REGRESIÓN (I)

- 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, inducidos por cambios recientemente realizados en partes de la aplicación que con anterioridad al citado cambio no eran propensas a este tipo de error**. Esto implica que el error tratado se reproduce como consecuencia inesperada del citado cambio o modificación realizada en el programa para su mejor o adaptación a nuevas necesidades del cliente.
- **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.
- Existen **herramientas de software** que permiten detectar este tipo de errores de manera parcial o totalmente automatizada. La práctica habitual en programación es que este tipo de pruebas se ejecuten en cada uno de los pasos del ciclo de vida del desarrollo del software.

PRUEBAS DE REGRESIÓN (II)

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 inspeccionando 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**. **Repetible significa que la prueba se puede repetir indefinidamente produciendo los mismos resultados siempre**.
- Realizando pruebas automáticas, podemos tener dos tipos: **prueba funcional y prueba unitaria**.

PRUEBAS AUTOMÁTICAS

- La **prueba funcional es de tipo “caja negra”** realizada sin conocimiento interno de la aplicación, a alto nivel, simulando la actuación del usuario.
- La **prueba unitaria**, por el contrario, **implica un conocimiento del software a bajo nivel**, no solo las entradas y las salidas. Se deben probar todos los métodos y clases importantes, e implica simular las entradas de información. Proporciona también soluciones a los problemas antes planteados:
 - **Elimina el componente repetitivo.** Se transmite el trabajo repetitivo al ordenador que seguirá exactamente la secuencia de pruebas cada vez que lo solicitemos.
 - **Reduce los errores.** Cada repetición será exactamente igual cada vez que se ejecute, de forma que se puede probar fácilmente todas las funcionalidades ante cualquier cambio sin temor a olvidar alguna parte.
 - **Permite probar las partes no visibles de código**, el corazón de la aplicación realmente. El nivel de detalle depende de la programación de la prueba.
 - **Permite a los usuarios finales** verificar que el software funciona como debe.

CARACTERÍSTICAS

- Repetibles



No debe alterar el estado del sistema.
Sin importar las veces que se ejecute.

- Independientes



La ejecución de un test no debe
afectar la de otro. Por eso no importa
el orden en que se ejecuten.

- Rápido



Al ser pequeñas unidades de código
que se prueban, estas se deben
ejecutar rápido

EJEMPLOS

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* (específica 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