TRABAJO DEL CURSO

Control de calidad y pruebas

- Cualquier usuario o desarrollador sabe, por experiencia propia, que los programas no son perfectos que los programas tienen errores.

- Cuanto mejor sea el proceso de ingeniería del software y el proceso de control de calidad y pruebas que se utilice, menos errores tendrá
- La gestión está muy relacionada con los requisitos

Términos comunes

- **ERROR(bug):** Fallo en la codificación

- **Alcance de código:** Parte de un programa que no va a ser utilizada
- **Prueba de estrés:** Proceso para medir el rendimiento de un sistema bajo cargas elevadas

Términos comunes

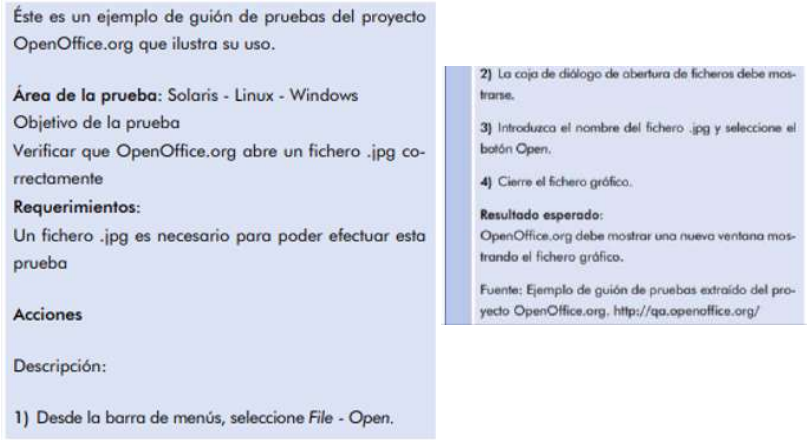
- **Prueba de regresión:** Comprobar que la funcionalidad del programa no haya sido dañada por modificaciones
- **Prueba de usabilidad:** pruebas para medir la usabilidad del sistema por parte de los usuarios
- **Testeador:** persona que se encarga de revisar si existen fallos

Principios de la Programación del SW

- Requisitos detallados del sistema: Utilizan los requisitos del sistema para verificar que cumplen con los mismo, por esa razón deben ser expresados de forma clara y detallados.
- Los procesos de calidad deben ser integrados en las fases iniciales: Los procesos de calidad deben ser parte del sistema desde el inicio.
- Los desarrolladores no deben ser quienes prueben el sistema: Debido a que una persona que no haya estado involucrada puede evaluar más fácilmente las fallas en un sistema.

Técnicas manuales de comprobación de software

- Conjunto de métodos ampliamente usados en los procesos de control de pruebas
- Dos versiones: informal (errores durante el uso) y formal (guiones de pruebas)
- El Objetivo de los guiones de pruebas es asegurar que la funcionalidad del programa no se ha visto afectada por las mejoras introducidas y que un usuario medio no encontrará ningún error grave
- Confiar el control de calidad únicamente a un proceso de pruebas manuales es bastante arriesgado y no ofrece garantías sólidas de la calidad del producto que hemos producido.



Éste es un ejemplo de guión de pruebas del proyecto OpenOffice.org que ilustra su uso.

Área de la prueba: Solaris - Linux - Windows
Objetivo de la prueba
Verificar que OpenOffice.org abre un fichero .jpg correctamente
Requerimientos:
Un fichero .jpg es necesario para poder efectuar esta prueba

Acciones

Descripción:

1) Desde la barra de menús, seleccione *File - Open*.

2) La caja de diálogo de abertura de ficheros debe mostrarse.

3) Introduzca el nombre del fichero .jpg y seleccione el botón Open.

4) Cierre el fichero gráfico.

Resultado esperado:
OpenOffice.org debe mostrar una nueva ventana mostrando el fichero gráfico.

Fuente: Ejemplo de guión de pruebas extraído del proyecto OpenOffice.org. http://qa.openoffice.org/

3.5

Técnicas automáticas de comprobación

Let’s start with the second set of slides

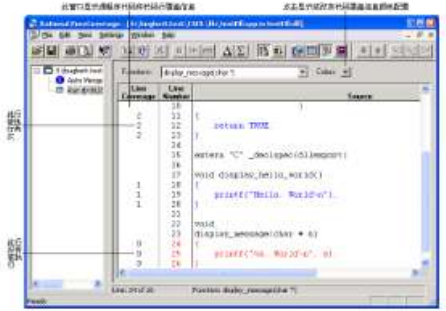
White- Box testing (conjunto de técnicas)

- Tiene como objetivo validar la lógica de la aplicación.
- Se centran en verificar la estructura interna del sistema, sin tener en cuenta los requisitos.

Inspección del código

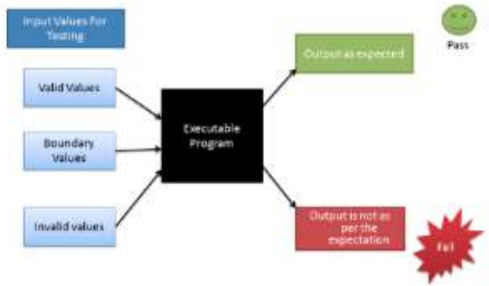
Con el objetivo de encontrar errores en la lógica del programa y la estructura interna de los datos y detectar el código que no se utiliza.

Existen Aplicaciones Propietarias



Black- Box testing

- Comprobar la funcionalidad de los componentes de una aplicación.
- Se llama black box (caja negra) porque el proceso de pruebas asume que se desconoce la estructura interna del sistema, sólo se comprueba que los datos de salida producidos p[illegible]eterminada son correctos.



Ranorex Studio

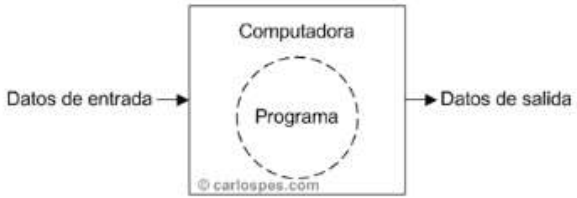
[illegible] siguientes errores:

- Funciones incorrectas o que faltan.
- Errores de interfaz.
- Errores en las estructuras de datos o en el acceso a la base de datos externa.
- Errores de comportamiento o de rendimiento.
- Errores de inicialización y terminación.

Unidades de comprobación

- Nace por la necesidad de procedimientos que aseguren la correcta funcionalidad de los diferentes componentes del software, y muy especialmente, de los que forman la base de nuestro sistema.

- El unit testing, se basa en la comprobación sistemática de clases o rutinas de un programa utilizando unos datos de entrada y comprobando que los resultados generados son los esperados.



Unidades de comprobación

Una unidad de prueba bien diseñada debe cumplir los siguientes requisitos:

- Debe ejecutarse sin atención del usuario (desatendida): Ejecutada sin ninguna intervención del usuario
- Debe ser universal: Posible ejecutar la prueba en cualquier sistema que tenga el software
- Debe ser atómica: Comprobar la funcionalidad concreta de un componente

3.6. Sistemas de control de errores

- Los sistemas de control de errores son herramientas colaborativas muy dinámicas proveen el soporte necesario para consolidar una pieza clave en la gestión del control de calidad:
  - Almacenamiento
  - Seguimiento
  - Clasificación de los errores de una aplicación
- El uso de estos sistemas en la mayoría de proyectos se extiende también a la gestión de mejoras solicitadas por el usuario e ideas para nuevas versiones.
- Usuarios los utilizan para tener un control de las llamadas de un soporte técnico, pedidos, o los utilizan como gestores de tareas que hay que realizar.

Reportado de errores

Cuando un usuario encuentra un error, una buena forma de garantizar que será corregido es escribir un informe de error lo más preciso y detallado posible.

Independientemente de cuál sea el sistema de control de errores que utilicemos, los siguientes consejos reflejan las buenas prácticas en el reporte de errores:

- Comprobar que el error no ha sido reportado anteriormente.
- Dar un buen título al informe de error.
- Dar una descripción detallada y paso a paso de cómo reproducir el error.
- Dar la máxima información sobre nuestra configuración y sus particularidades.
- Reportar un único error por informe.

Anatomia de un informe

Todos los sistemas detallan el error con una serie de campos que permiten definirlo y clasificarlo de forma precisa.

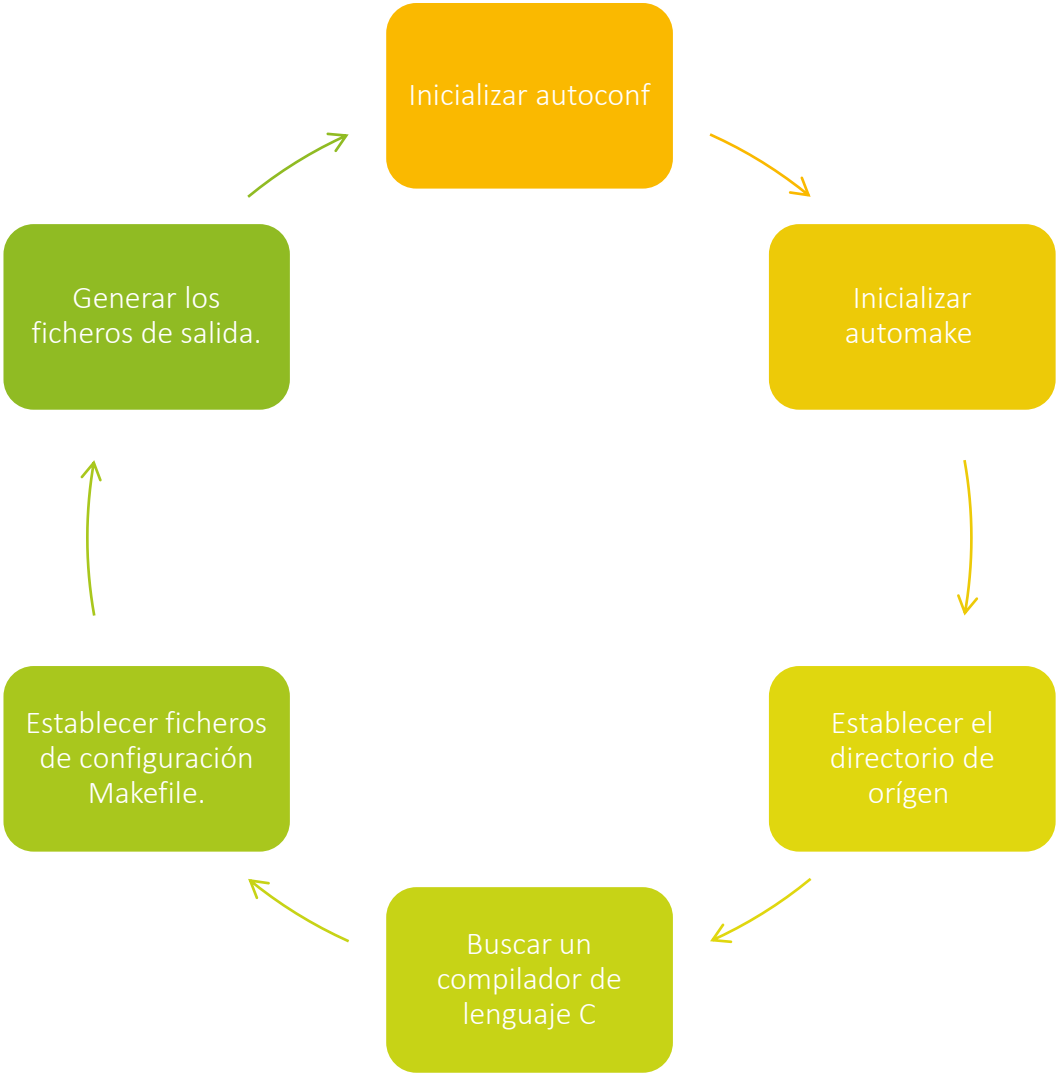
- Identificador.
- Título.
- Informador.
- Fecha de entrada.
- Estado.
- Gravedad.
- Palabras clave.
- Entorno.
- Descripción.

Éstos son los campos que podemos considerar los mínimos comunes a todos los sistemas de control de errores.

Instalación de los autotools

Install	•Describe el proceso de instalación
Copying	•Especifica la licencia bajo la que se permite o restringe la realización de las copias
News	•Especifica una lista de modificaciones
Readme	•Este fichero es históricamente el primero en ser revisado por el usuario.
Authors	•Es un fichero con el nombre de todas las personas que participaron en el desarrollo.
Changelog	•Permite conocer los cambios, actualizaciones y estado del software.

Fichero autoconf

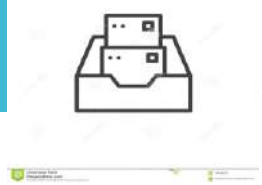


Mantenimiento de ficheros de entrada

Si se edita alguno de los ficheros de entrada de los autotools se deberá actualizar:

- autoconf
-automake

Una forma es crearse un script que repita los pasos anteriores para crear los ficheros.
Los autotools tienen un programa llamado autoreconf: ejecuta secuencia estándar de l le autotools.



Empaquetar ficheros output

- Práctica común es la inclusión de todos los ficheros generados que sea suficiente para ejecutar *./configure* y *make all* o *make install* para que el programa sea funcional.

Implicaciones positivas

- Facilidad de pasar parches y poder disfrutar del programa corregido en el menor tiempo posible.

Implicaciones negativas

- Si estos ficheros fuente se intentasen compilar en otro sistema sería posible que no fuese compatible, por lo que tendría que re-generarse.

Se debe incluir los ficheros originales configure. Ac y Makefile.am así como sus derivados especificando cual es la plataforma.

Construcción de software en entorno GNU

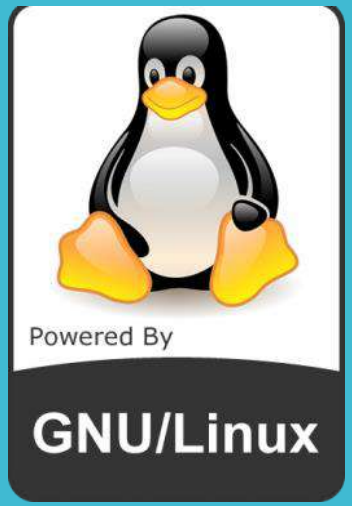
Changelogs y documentación

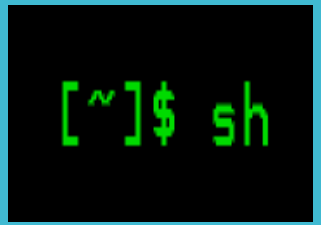
Desarrollar y mantener el código es crucial pues este será analizado y mejorado. La relevancia de la documentación para usuarios noveles, avanzados como para administradores es indispensable, y este es el ChangeLog.
Al realizar este fichero debemos definir que poner dentro aquí ingresamos los chequeos que debería tener el programa para funcionar efectivamente o su orden.

Creación de configure.in

Qué sigifica portabilidad

Es aquél que un usuario pueda hacer funcionar sin mayor esfuerzo en cualquier plataforma y sistema





SH PORTABLE Y CHEQUEOS NECESARIOS

- La herramienta más importante para la instalación de un programa es el programa de línea de comando en sí.
- Las autotools intentan generar un fichero ./configure con el conjunto de funcionalidades de sh más compatible posible.
- Los programas GNU suelen incluir una sección en su documentación (man) detallando las opciones que son compatibles con POSIX.

Secuencia principal

Inicialización-Configuración-Programas-Librerías-Encabezados- Estructuras-Funciones-Salida

4.4. Introducción a GNU Automake

- Automake es la utilidad encargada de crear los ficheros Makefile.in.
- Los ficheros Makefile, necesarios para la compilación de prácticamente todos los programas, una gran cantidad de usuarios es capaz de aportar información sobre cualquier tipo de incompatibilidad, error, u optimización que se deba corregir o incorporar a automake.

4.4.1. Principios generales de automake

- Automake se encarga de convertir Makefile.am en Makefile.in que posteriormente será usado por ./ configure como una plantilla para los ficheros Makefile finales.
- Los comentarios internos de automake, que serán descartados al crear el Makefile.in, y que empezarán con '##' en vez de '#'
- Todas las instrucciones y posibles macros no interpretadas directamente por automake son pasadas tal cual a Makefile.in.

4.4.2 Introducción a las primarias

- Las variables “primarias” se utilizan indirectamente para derivar de ellas nombres de objetos a partir de ellas
- En este caso se usa el prefijo especial bin que indicaría el destino del programa en “bindir”. Algunos de los destinos posibles son los siguientes:

- bindir: donde deben ir los archivos binarios
- sbindir: para los archivos binarios de sistemas
- libexecdir: programas y demonios ejecutables por otros programas
- pkglibdir: librerías no compartidas en el sistema
- noinst: archivos que no se instalarán
- check: sólo serán compilados con “make check” para comprobaciones
- man: páginas de manual

4.4.3 Programas y librerias

Como ya se ha mencionado, los programas se indican bajo la primaria PROGRAMS. En el caso de las librerías, se aplica la primaria LIBRARIES:

```
lib_LIBRARIES = libtest.a

libtest_a_SOURCES = ltest.c ltest.h
```

4.4.4 Múltiples directorios

- Los proyectos grandes, suelen estar divididos en diferentes subdirectorios. Para esto automake acepta la opción, SUBDIRS:
  SUBDIRS = src libsrc man
- La única restricción es que los directorios deben ser descendientes del actual.

:

4.4.5 Cómo probarlo

Una de las características de automake y de los Makefile generados, es la capacidad de realizar tests de funcionamiento del código compilado.

Para especificar un destino común a todos los programas de test, se

puede usar el destino check con la primaria PROGRAMS:

check_PROGRAMS = test1 test2

test1_SOURCES = test1.c

test2_SOURCES = test2.c

Así mismo, se pueden ejecutar los tests de DejaGnu con:

AUTOMAKE_OPTIONS = dejagnu

TEMA: 4.5 LA LIBRERÍA GNU LIBTOOL

GRUPO 2

Código independiente y librerías compartidas.

Las librerías internas no requieren de un código independiente de posición (Position Independent Code = PIC). Mientras que las librerías compartidas entre distintos programas necesitan que la dirección en la que se carga la librería sea alterable.

Para la creación de código libre se debe pasar por una serie de opciones al compilar. Libtool permite ocultar estas tareas encargadas del proceso de compilado.

Libtool siempre crea librerías compartidas por defecto siempre que sea posible. Únicamente en plataformas donde no sea posible creará librerías estáticas.

suma.c

```
int suma (int a, int b) {
 return a+b;
}
```

```
$ gcc -c suma.c
$ ls
suma.c  suma.o
$ ar cru libsuma.a suma.o
$ ranlib libsuma.a
$ ls
libsuma.a  suma.c  suma.o
```

```
$ libtool --mode=compile gcc -c suma.c
mkdir .libs
  gcc -c suma.c  -fPIC -DPIC -o .libs/suma.o
  gcc -c suma.c -o suma.o >/dev/null 2>&1
$ libtool --mode=link gcc -o libsuma.la suma.lo -rpath /usr/local/lib/
rm -fr  .libs/libsuma.a .libs/libsuma.la
gcc -shared  .libs/suma.o   -Wl,-soname -Wl,libsuma.so.0 -o .libs/libsuma.so.0.0.0
(cd .libs && rm -f libsuma.so.0 && ln -s libsuma.so.0.0.0 libsuma.so.0)
(cd .libs && rm -f libsuma.so && ln -s libsuma.so.0.0.0 libsuma.so)
ar cru .libs/libsuma.a .libs/suma.o
ranlib .libs/libsuma.a
creating libsuma.la
(cd .libs && rm -f libsuma.la && ln -s ../libsuma.la libsuma.la)
$ ls
libsuma.la  suma.c  suma.lo  suma.o
```

- Libtool permite crear librerías estáticas sin llamar al compilador, se usa la opción *static*.
- Se necesita enlazar la librería a un ejecutable.
- Es común enlazar librerías para añadir funcionalidad y maximizar la portabilidad de un programa.
- Libtool usa el propio sistema del destino para la interdependencia de librerías. También ofrece una emulación propia.

```
$ libtool --mode=link gcc -static -o libsuma.la suma.lo
rm -fr  .libs/libsuma.a .libs/libsuma.la
ar cru .libs/libsuma.a  suma.o
ranlib .libs/libsuma.a
creating libsuma.la
(cd .libs && rm -f libsuma.la && ln -s ../libsuma.la libsuma.la)
```

```
$ libtool --mode=link gcc -o suma main.c libsuma.la
gcc -o .libs/suma main.c  ./.libs/libsuma.so -Wl,--rpath -Wl,/usr/local/lib
creating suma
$ ./suma
3
```

USAR LIBRERÍAS DE CONVENIENCIA

INSTALACIÓN DE LIBRERÍAS Y EJECUTABLES

- Librerías parcialmente enlazadas usadas como agrupaciones intermedias de objetos.
- No son ni ejecutables ni librerías.
- Usadas solo para ser enlazadas dentro de otras librerías o ficheros ejecutables.

```
$ libtool --mode=install cp libsuma.la /usr/local/lib/libsuma.la
cp libsuma.la /usr/local/lib/libsuma.la
cp .libs/libsuma.a /usr/local/lib/libsuma.a
ranlib /usr/local/lib/libsuma.a
```

Para finalizar la instalación usaremos `--mode=finish`:

```
libtool --mode=finish /usr/local/lib
PATH="$PATH:/sbin" ldconfig -n /usr/local/lib
```

Desinstalación

- La desinstalación podemos realizarla por medio de --mode=uninstallcon el comando rm:

```
$ libtool --mode=uninstall rm -f /usr/local/lib/libsuma.la
rm -f /usr/local/lib/libsuma.la /usr/local/lib/libsuma.a
rm -f /usr/local/lib/libsuma.la /usr/local/lib/libsuma.so.0.0.0 /usr/local/lib/libsuma.so.0 /usr/
local/lib/libsuma.so /usr/local/lib/libsuma.a
```

GNU Libtool, configure.in y Makefile.am

- Para utilizar libtool con autoconf y automake sólo hace falta definir su uso en configure.ac.
- La macro que nos ofrece autoconf para ello es la AC_PROG_LIBTOOL. Del resto de librerías se encargará libtool: AC_PROG_LIBTOOL
- Finalmente se ejecuta autoconf y automake.

Integración con configure.in, opciones extra y marcos para Libtool

- Enable-shared y disable-shared para la activación de librerías compartidas
- Enable-static y disable-statica para las librerías estáticas.
- With-pic fuerza a construir las librerías con código independiente,
- En el configure.ac : AC_DISABLE_SHARED y AC_DISABLE_STATIC

Integración con Makefile.am, creación de librerías con Automake y linkado contra librerías Libtool

- Automake compila y enlaza librerías por medio del destino lib y la primaria LIBRARIES.

```
lib_LIBRARIES = libsuma.a
libsuma_a_SOURCES = suma.c
```

- La construcción de librerías de libtools se hace con la primaria LTLLIBRARIES

```
lib_LTLIBRARIES = libsuma.la
libsuma_la_SOURCES = suma.c
```

- El enlazado con la librería libtool es mediante la primaria LDADD

```
bin_PROGRAMS = suma
suma_SOURCES = main.c
suma_LDADD = libsuma.c
```

Utilización de libtoolize

configure.ac

```
AC_INIT(suma, 0.1)
AM_INIT_AUTOMAKE(suma, 0.1)
AC_CONFIG_SRCDIR([suma.c])

# Checks for programs.
AC_PROG_CC
AC_PROG_LIBTOOL

AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Makefile.am

```
bin_PROGRAMS = suma
suma_SOURCES = main.c
suma_LDADD = libsuma.la

lib_LTLIBRARIES = libsuma.la
libsuma_la_SOURCES = suma.c
```

Versionado de librerías

interfaz se establecen por medio de --ver-sion-info, en actual : revisión : antigüedad

- actual: establece el número de la interfaz más moderna soportada.
- revisión: permite distribuir librerías actualizadas con la misma interfaz.
- antigüedad: número de interfaces anteriores soportadas.