



1. INTRODUCCIÓN

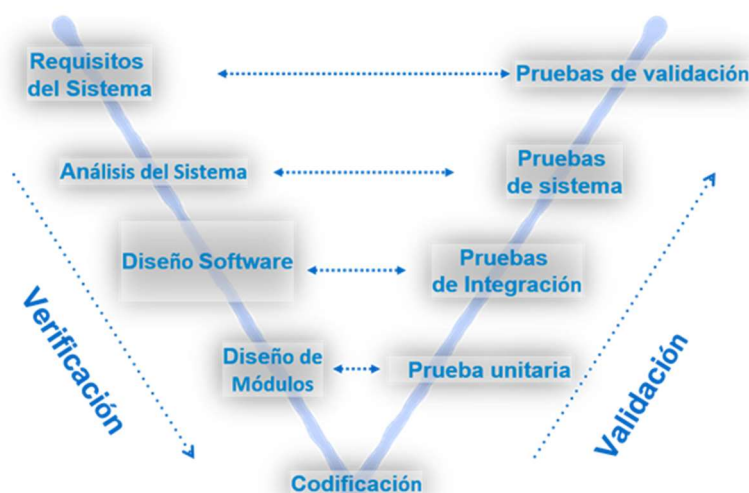
Las pruebas de software consisten en verificar y validar un producto software antes de su puesta en marcha. Constituyen una de las etapas del desarrollo de software, y básicamente consiste en probar la aplicación desarrollada. Se integran dentro de las diferentes fases del ciclo de vida del software dentro de la ingeniería de software.

Muchas veces se confunde “verificación” con “validación”:

- **Validación:** ¿Estamos construyendo el producto correcto? Se ocupa de controlar si el producto satisface los requerimientos del usuario.
En la *Validación* el resultado final del desarrollo software se debe ajustar a lo que el usuario quería (sus necesidades). En la mayoría de las ocasiones el producto desarrollado no coincide con la ideas del cliente, normalmente porque a éste suele faltarle capacidad técnica de expresión.
- **Verificación:** ¿Estamos construyendo correctamente el producto? implica controlar que el producto concuerda con su especificación inicial.
En la *Verificación* el código que estamos construyendo debe estar en armonía con la especificación que hemos tomado del usuario. El resultado final del desarrollo software debe concordar con la especificación (requisitos) del sistema, por lo que debemos asegurarnos que el desarrollo final coincida con dicha especificación.

Un sistema puede pasar la validación, sin embargo, no pasar la verificación. Cumple con la especificación del usuario, con lo que él quería, cubre sus necesidades pero internamente puede tener graves detalles como: un precario diseño en la base de datos; uso de un excesivo e innecesario nº de líneas de código, por desconocer las potencialidades del lenguaje de desarrollo o de técnicas avanzadas de programación como la POO y uso incorrecto en la BD de instrucciones propias del lenguaje de desarrollo, en lugar de las sentencias adecuadas de SQL.

La ejecución de pruebas de un sistema implica una serie de etapas como: planificación de pruebas, diseño y construcción de los casos de prueba, definición de los procedimientos de prueba, ejecución de las pruebas, registro de resultados obtenidos, registro de errores encontrados, depuración de los errores e informe de los resultados obtenidos.



2. PROCEDIMIENTOS DE PRUEBAS

Un procedimiento de prueba es la definición del objetivo que desea conseguirse con las pruebas, qué es lo que va a probarse y cómo.

El objetivo de las pruebas no siempre es detectar errores. Muchas veces lo que quiere conseguirse es que el sistema ofrezca un rendimiento determinado, que la interfaz tenga una apariencia y cumpla unas características determinadas, etc.

Por lo tanto, la ausencia de errores en las pruebas nunca significa que el software las supere, pues hay muchos parámetros en juego.


Cuando se diseñan los procedimientos, se deciden las personas que hacen las pruebas y bajo qué parámetros van a realizarse.

No siempre tiene que ser los programadores los que hacen las pruebas. No obstante, siempre tiene que haber personal externo al equipo de desarrollo, puesto que los propios programadores solo prueban las cosas que funcionan (si supieran dónde están los errores, los corregirían).

Hay que tener en cuenta que es imposible probar todo, la prueba exhaustiva no existe. Muchos errores del sistema saldrán en producción cuando el SW ya esté implantado, pero siempre se intentará que sea el mínimo número de ellos.

En los planes de pruebas (es un documento que detalla en profundidad las pruebas que se van a realizar), generalmente cubren los siguientes aspectos:

1. *Introducción.* Breve introducción del sistema describiendo objetivos, estrategias, etc.
2. *Módulos o partes del sw pr probar.* Detallar cada una de estas partes o módulos.

 <p>CIFP VIRGEN DE GRACIA</p>	<p>UT3-DISEÑO Y REALIZACIÓN DE PRUEBAS Entornos de desarrollo (1º DAW)</p>	<p>Dpto. INFORMÁTICA</p> <p>Curso: 2022-23</p>
--	---	---

3. *Características del SW por probar.* Tanto individuales como conjuntos de ellas.
4. Características del SW que no ha de probarse.
5. *Enfoque de las pruebas.* En el que se detallan, entre otros, las personas responsables, la planificación, la duración, etc.
6. *Criterios de validez o invalidez del sw.* En este apartado, se registra cuando el sw puede darse como válido o como inválido especificando claramente los criterios.
7. *Proceso de pruebas.* Se especificará el proceso y los procedimientos de las pruebas por ejecutar.
8. *Requerimientos del entorno.* Incluyendo niveles de seguridad, comunicaciones, necesidades HW y SE, herramientas, etc.
9. *Homologación o aprobación del plan.* Este plan deberá estar firmado por los interesados o sus responsables.

Las demás fases del proceso de pruebas, como puede entenderse, con el mero desarrollo del plan de pruebas anterior.

RECUERDA:

Probar es ejecutar casos de prueba uno a uno, pero que un SW pase todos los casos de prueba no quiere decir que el programa esté exento de fallos.

Como se ha observado, las pruebas solo encuentran o tratan de encontrar aquellos errores que van buscando, luego, es muy importante realizar un buen diseño de pruebas con buenos casos de prueba, puesto que se aumenta de esta manera la probabilidad de encontrar fallos.

Ejemplo

Caso de pruebas

Imagínese que se tiene la ventana anterior y desea realizarse un caso de pruebas. La descripción de un caso de pruebas Caso 1 para esta aplicación sería el que se observa en la figura 3.1.

- ✓ *Objetivo:* comprobar que un usuario correcto entra en el sistema y la fecha y hora de entrada queda registrada.
- ✓ *Entrada de datos:* en el campo User, se introduce "myfpschool" y, en el campo Password, "Troconne77".
- ✓ *Condiciones:* en la tabla Usuarios, existe el usuario myfpschool con la contraseña Troconne77 encriptada en MD5.
- ✓ *Resultado:* el usuario entra en el sistema y, en la tabla Log, deja un registro ("myfpschool", fecha, hora).
- ✓ *Procedimiento de la prueba:*
 1. Se comprueba en la tabla Usuarios que existe el usuario por introducir.
 2. Se comprueba que la contraseña esté codificada en MD5 correctamente.
 3. En los campos User y Password, se teclean los datos "myfpschool" y "Troconne77".
 4. Se pulsa Aceptar y se comprueba que se accede al sistema correctamente.
 5. Se revisa la tabla Log y se comprueba que se ha registrado el usuario, la fecha y la hora actual.



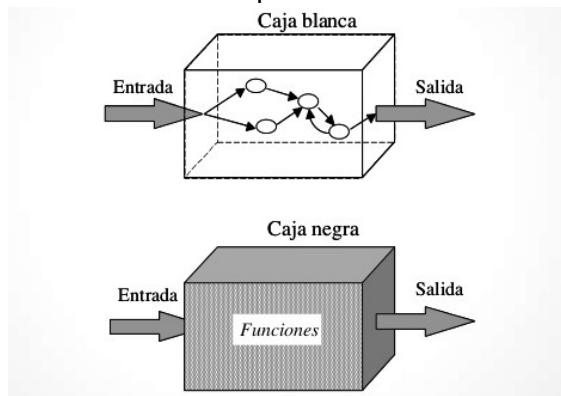
Figura 3.1
Ventana de acceso a una aplicación.

3. TÉCNICAS DE DISEÑO DE CASOS DE PRUEBA

Un **caso de prueba** es un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollado para conseguir un objetivo particular o condición de prueba. Para llevar a cabo un caso de prueba, es necesario definir las precondiciones y las post condiciones, identificar unos valores de entrada y conocer el comportamiento que debería tener el sistema ante dichos valores. Tras realizar ese análisis e introducir dichos datos en el sistema, se observa si su comportamiento es el previsto o no y por qué. De esta forma se determina si el sistema ha pasado o no la prueba (Ver GUÍA DE VALIDACIÓN Y VERIFICACIÓN. Inteco. Laboratorio Nacional de Calidad del Software <http://docplayer.es/19670553-Guia-de-validacion-y-verificacion.html>)

Para llevar a cabo el diseño de casos de prueba se utilizan dos técnicas o estrategias:

- Las **pruebas de caja blanca** se centran en validar la estructura interna del programa (necesitan conocer los detalles procedimentales del código) y
- Las **pruebas de caja negra** se centran en validar los requisitos funcionales sin fijarse en el funcionamiento interno del programa (necesitan saber la funcionalidad que el código ha de proporcionar). Estas pruebas no son excluyentes y se pueden combinar para descubrir diferentes tipos de errores.



3.1 Pruebas de caja blanca

También se las conoce como *pruebas estructurales* o *de caja de cristal*. Se basan en el minucioso examen de los detalles procedimentales del código de la aplicación. Mediante esta técnica se pueden obtener casos de prueba que:

- Garanticen que se ejecuten al menos una vez todos los caminos independientes de cada módulo.
- Ejecuten todas las sentencias al menos una vez.
- Ejecuten todas las decisiones lógicas en su parte verdadera y en su parte falsa.
- Ejecuten todos los bucles en sus límites.
- Utilicen todas las estructuras de datos internas para asegurar su validez.

Una de las técnicas utilizadas para desarrollar los casos de prueba de caja blanca es la *prueba del camino básico* que veremos más adelante.

Como se ha comentado, en las pruebas de caja blanca, se conoce o se tiene en cuenta el código que quiere probarse. Se denomina también **clear box testing** porque la persona que realiza las pruebas está en contacto con el código fuente. Su objetivo es probar el código, cada uno de sus elementos.

Existen algunas clases de pruebas de este tipo como, por ejemplo:

- Pruebas de cubrimiento
- Pruebas de condiciones
- Pruebas de bucles

3.1.1 Pruebas de cubrimiento

En este tipo de pruebas, el objetivo es ejecutar, al menos una vez, todas las sentencias o líneas del programa.

En ocasiones, es imposible cubrir el 100% del código porque puede haber condiciones que nunca se cumplan:

```
if (a > 20 && a < 10) { ... }
```

O también puede haber excepciones o notificaciones de error en un código que nunca va a fallar (código sin errores).

Para realizar las pruebas, habrá que generar el suficiente número de casos de prueba para poder cubrir los distintos caminos independientes del código. En cada condición, deberá cumplirse en un caso y en otro no.

3.1.2 Pruebas de condiciones

En este caso, se necesitarán varios casos de prueba. Una condición, puede hacer varias condiciones simples y habrá que genere un caso de pruebas por cada operando lógico o comparación. La idea es que, en cada expresión, se cumpla en un caso y en otro no.

```
if (videogames=1 && manga=1 && technology=1){ freaky = 1}
```

En el caso anterior, deberán de comprobarse todas las combinaciones de las tres variables anteriores.

En esta ocasión, son variables, pero podrían ser otro tipo de expresiones más complejas

De esta manera, habrá que cerciorarse de que cualquiera de las combinaciones de valores de la condición funcionará tal y como el programa fue concebido.

3.1.3 Pruebas de bucles

La prueba de bucles se basará en la repetición de un número especial de veces.

En el caso de un bucle simple, los casos de prueba deberían contemplar lo siguiente:

- Repetir el máximo, máximo -1 y 'máximo + 1 veces el bucle para ver si el resultado del código es el esperado.
- Repetir el bucle cero y una vez
- Repetir el bucle un numero x de veces



3.2 Pruebas de caja negra

Estas pruebas se llevan a cabo sobre la interfaz del software, no hace falta conocer la estructura interna del programa ni su funcionamiento. Se pretende obtener casos de prueba que demuestren que las funciones del software son operativas, es decir, que las salidas que devuelve la aplicación son las esperadas en función de las entradas que se proporcionen.

A este tipo de pruebas también se les llama *prueba del comportamiento*. El sistema se considera como una caja negra cuyo comportamiento solo se puede determinar estudiando las entradas y las salidas que devuelve en función de las entradas suministradas.

Con este tipo de pruebas se intenta encontrar errores de las siguientes categorías:

- Funcionalidades incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en acceso a bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y finalización.

Existen diferentes técnicas para confeccionar los casos de prueba de caja negra, algunos son: *clases de equivalencia*, *análisis de valores límite*, *métodos basados en grafos*, *pruebas de comparación*, etc.

Entre las pruebas de caja negra, pueden citarse las siguiente:

- Pruebas de clases de equivalencia de datos
- Pruebas de valores límite
- Pruebas de interfaces.

3.2.1 Prueba de clases de equivalencia de datos

Imagínese que se está probando un interfaz y se debe generar un código de usuario y una clave

- Usuario: mayúsculas y minúsculas, no puede tener caracteres que no sean alfabéticos y tiene que tener al menos 6 letras (máximos 12)
- Password: tendrán al menos 8 caracteres (máximo 10) y contendrán letras y números.

Para testear esta interfaz, lo que debe hacerse es establecer claves de equivalencia para cada uno de los campos . Tendrán que rgnerar clases válidas y clases inválidas por cada uno de los campos.

1. *Usuario:*

- *Clases válidas:* “Pelegrino” y “Rocinante”.
- *Clases inválidas:* “marrullero44”, “nene”, “Portaavionesgigante”, “Z&aratustra” y “Ventajoso12”.

2. *Contraseña:*

- *Clases válidas:* “5Entrevias” y “s8brino”.
- *Clases inválidas:* “corta”, “muyperoquemuylarguísima”, “oletugarbo” y “999999999”.

3.2.2 Prueba de valores límite

El objetivo es generar valores que puedan probar si la interfaz y el programa funcionan correctamente.

Ejemplo en un banco: “La cifra máxima que usted puede transferir hoy es de 10.000 euros”.

- Prvalores fuera de rango como -100 o 20.000
- Probar valores en los límites como 0,1,9.999,10.000,10.001
- Probar valores obar típicos e intermedios como 9.000 o 2.500.

El objetivo de esta prueba se halla en que, muchas veces, los programadores se equivocan al establecer los límites en la frontera.

3.2.3 Prueba de interfaces

Un interfaz de usuario o GUI (Graphical user interface)-> prueba de interfaces

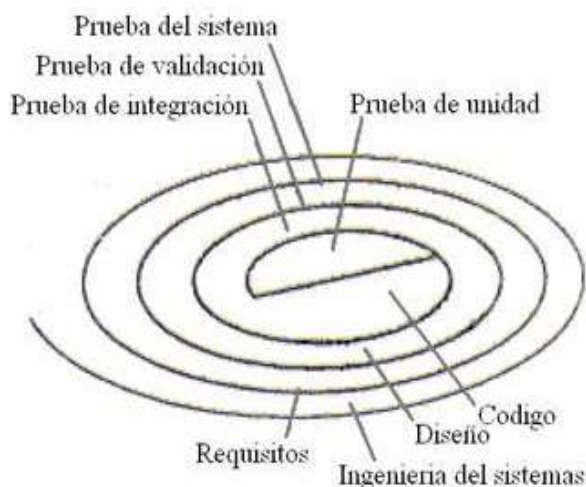
Una Interfaz es una serie de objetos con una serie de propiedades. Toda esta serie de objetos con sus propiedades en su conjunto formarán la interfaz.

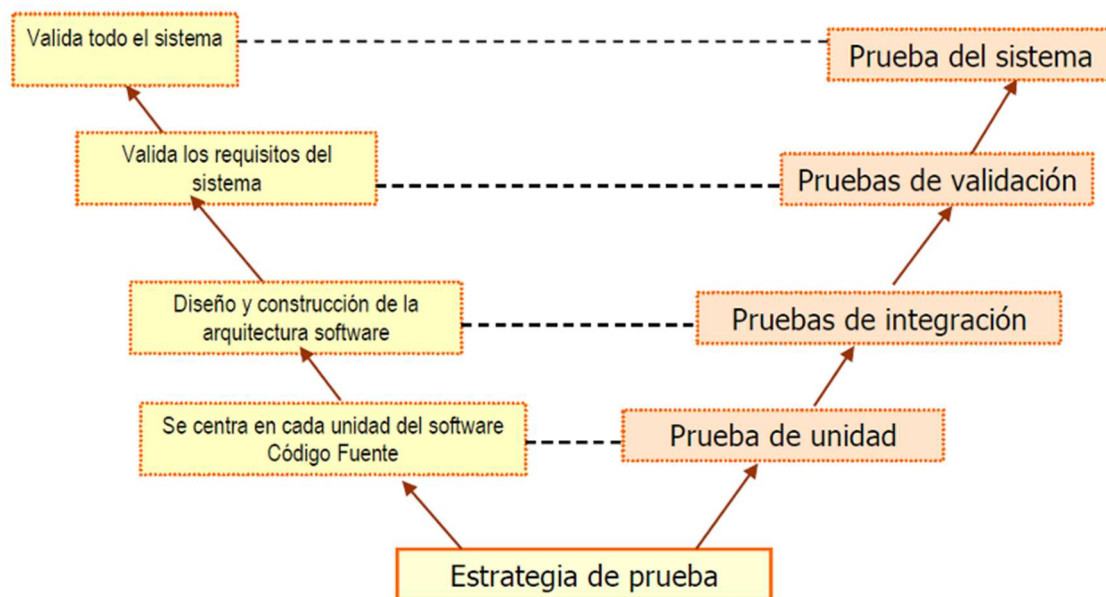
Dependiendo de las entradas, la interfaz proporcionará una salida determinada. Y esa salida debería ser la esperada. Muchas veces, cuando se testea un programa, hay que conocer su funcionalidad.

4. ESTRATEGIAS DE PRUEBAS DEL SOFTWARE

La estrategia de prueba del software se puede ver en el contexto de una espiral:

- En el vértice de la espiral comienza la **prueba de unidad**. Se centra en la unidad más pequeña de software, el módulo tal como está implementado en código fuente.
- La prueba avanza para llegar a la **prueba de integración**. Se toman los módulos probados mediante la prueba de unidad y se construye una estructura de programa que está de acuerdo con lo que dicta el diseño. El foco de atención es el diseño.
- La espiral avanza llegando a la **prueba de validación** (o de aceptación). Prueba del software en el entorno real de trabajo con intervención del usuario final. Se validan los requisitos establecidos como parte del análisis de requisitos del software, comparándolo con el sistema que ha sido construido.
- Finalmente se llega a la **prueba del sistema**. Verifica que cada elemento encaja de forma adecuada y se alcanza la funcionalidad y rendimiento total. Se prueba como un todo el software y otros elementos del sistema.



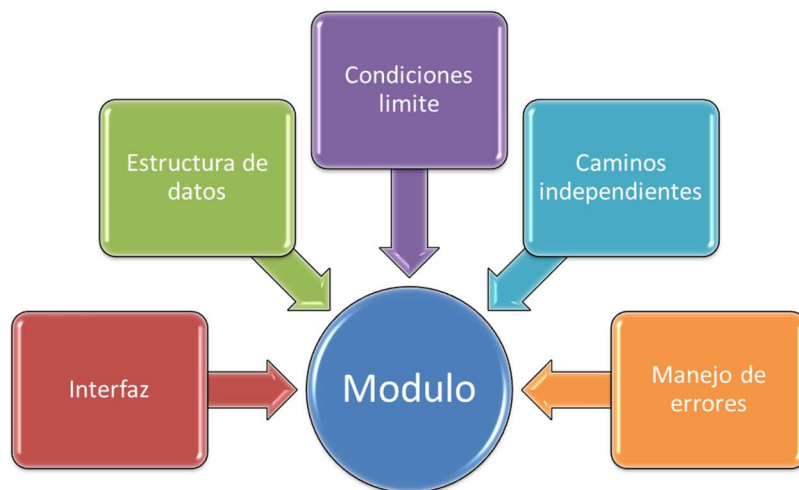


4.1 Prueba de unidad

En este nivel se prueba cada unidad o módulo con el objetivo de eliminar errores en la interfaz y en la lógica interna. Esta actividad utiliza técnicas de caja negra y caja blanca, según convenga para lo que se quiera probar. Se realizan pruebas sobre:

- La interfaz del módulo, para asegurar que la información fluye adecuadamente.
- Las estructuras de datos locales, para asegurar que mantienen su integridad durante todos los pasos de ejecución del programa.
- Las condiciones límite, para asegurar que funciona correctamente en los límites establecidos durante el proceso.
- Todos los caminos independientes de la estructura de control, con el fin de asegurar que todas las sentencias se ejecutan al menos una vez.

- Todos los caminos de manejo de errores.



Algunas de las herramientas que se utilizan para pruebas unitarias son *JUnit*, *CPPUnit*, *PHPUnit*, etc.

4.2 Prueba de integración

En este tipo de prueba se observa cómo interaccionan los distintos módulos. Se podría pensar que esta prueba no es necesaria, ya que si todos los módulos funcionan por separado, también deberían funcionar juntos. Realmente el problema está aquí, en comprobar si funcionan juntos.

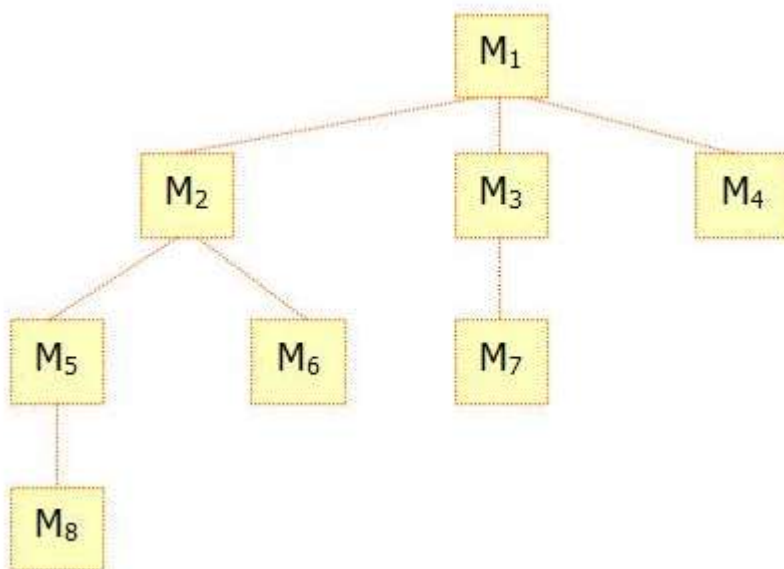
Existen dos enfoques fundamentales para llevar a cabo las pruebas:

- **Integración no incremental o *big bang*.** Se prueba cada módulo por separado y luego se combinan todos de una vez y se prueba todo el programa completo. En este enfoque se encuentran gran cantidad de errores y la corrección se hace difícil.
- **Integración incremental.** El programa completo se va construyendo y probando en pequeños segmentos en este caso los errores son más fáciles de localizar. Se dan dos estrategias *Ascendente* y *Descendente*. En la integración *Ascendente* la construcción y la prueba del programa empieza desde los módulos de los niveles más bajos de la estructura del programa. En la *Descendente* la integración comienza en el módulo principal (programa principal) moviéndose hacia abajo por la jerarquía de control.

La siguiente figura representa varios módulos y la interconexión entre ellos.

El módulo principal es el que está en la raíz M_1 .

En la integración Ascendente se empieza probando los módulos de más bajo nivel en la jerarquía módulos del sistemas: módulos M_8 , M_6 , M_7 y M_4 y luego se procede a probar la integración del abajo hacia arriba, en el M_5 , se prueba el M_8 y M_5 , en el módulo M_2 se prueban los módulos M_2 , M_5 , M_6 y M_8 y así hasta llegar al programa principal M_1 .



4.3 Prueba de validación


La validación se consigue cuando el software funciona de acuerdo con las expectativas razonables del cliente definidas en el documento de especificación de requisitos del software. Se llevan a cabo una serie de pruebas de caja negra que demuestran la conformidad con los requisitos. Las técnicas a utilizar son:

- **Pruebas Alfa.** Se lleva a cabo por el cliente o usuario en el lugar de desarrollo. El cliente utiliza el software de forma natural bajo la observación del desarrollador que irá registrando los errores y problemas de uso.
- **Prueba Beta.** Se lleva a cabo por los usuarios finales del software en su lugar de trabajo. El desarrollador no está presente. El usuario registra todos los problemas que encuentra reales y/o imaginarios, e informa al desarrollador en los intervalos definidos en el plan de prueba. Como resultado de los problemas informados el desarrollador lleva a cabo las modificaciones y prepara una nueva versión del producto.

4.4 Prueba del sistema

La prueba del sistema está formada por un conjunto de pruebas cuya misión es ejercitar profundamente el software. Son las siguientes:

- **Prueba de recuperación.** Este tipo de prueba se fuerza el fallo del software y se verifica que la recuperación se lleva a cabo apropiadamente.
- **Prueba de seguridad.** Esta prueba intenta verificar que el sistema está protegido contra accesos ilegales.
- **Prueba de resistencia (Stress).** Trata de enfrentar el sistema con situaciones que demandan gran cantidad de recursos, por ejemplo, diseñando casos de

 CIFP VIRGEN DE GRACIA	<p style="text-align: center;">UT3-DISEÑO Y REALIZACIÓN DE PRUEBAS Entornos de desarrollo (1º DAW)</p>	<p style="text-align: center;">Dpto. INFORMÁTICA</p> <p style="text-align: center;">Curso: 2022-23</p>
--	---	---

prueba que requieran el máximo de memoria, incrementando la frecuencia de datos de entrada, que den problemas en un sistema operativo virtual, etc.

5. DOCUMENTACIÓN PARA LAS PRUEBAS

El conjunto de documentos que pueden producirse durante el proceso de prueba son los siguientes:

- **Plan de pruebas.** Describe el alcance, los recursos y el calendario de las actividades de prueba. Identifica los elementos a probar, las características que se van a probar, las tareas que se van a realizar, el personal responsable de cada tarea y los riesgos asociados al plan.
- **Especificaciones de prueba.** Están cubiertas por tres tipos de documentos: la especificación del diseño de la prueba (se identifican los requisitos, casos de prueba y procedimientos de prueba necesarios para llevar a cabo las pruebas y se especifica la función de los criterios de pasa no-pasa), la especificación de los casos de prueba (documenta los valores reales utilizados para la entrada, junto con los resultados previstos), y la especificación de los procedimentitos de prueba (donde se identifican los pasos necesarios para hacer funcionar el sistema y ejecutar los casos de prueba especificados).
- **Informes de pruebas.** Se definen cuatro tipos de documentos: un informe que identifica los elementos que están siendo probados, un registro de las pruebas (donde se registra lo que ocurre durante la ejecución de la prueba), un informe de incidentes de prueba (describe cualquier evento que se produce durante la ejecución de la prueba que requiere mayor investigación) y un informe resumen de las actividades de prueba