# 6 Diseño y realización de pruebas de software

## 6.1 Introducción

Un **error** no detectado al inicio del desarrollo de un proyecto puede llegar a necesitar cincuenta veces más esfuerzos para ser solucionado que si es detectado a tiempo.

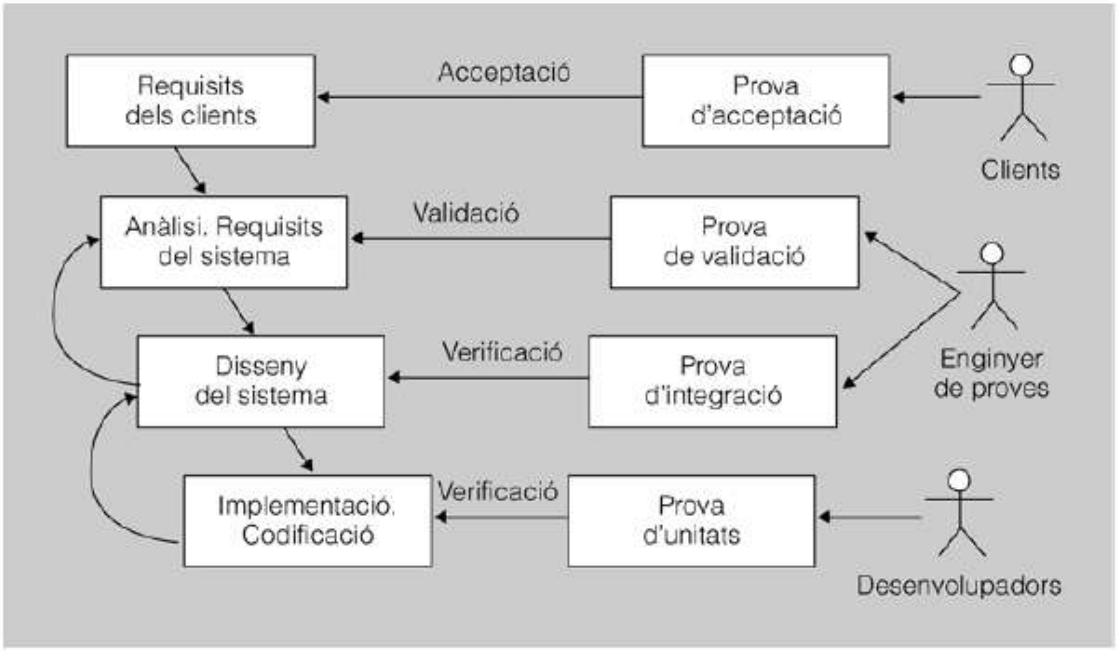
En un proyecto de desarrollo de software está estipulado que se dedica entre un **30% y un 50% del coste de todo el proyecto en la fase de pruebas**.

El **objetivo** de las pruebas es la evaluación de la calidad del software desarrollado durante todo su ciclo de vida, validando que hace lo que debe hacer y que lo hace tal y como se diseñó, a partir de los requerimientos.

## 6.2 Las pruebas en el ciclo de vida de un proyecto

En cada una de las fases del ciclo de vida de un proyecto, será necesario que el trabajo llevado a término sea validado y verificado.

En el esquema de la figura inferior podemos ver cómo encajan las pruebas en el ciclo de vida del software.



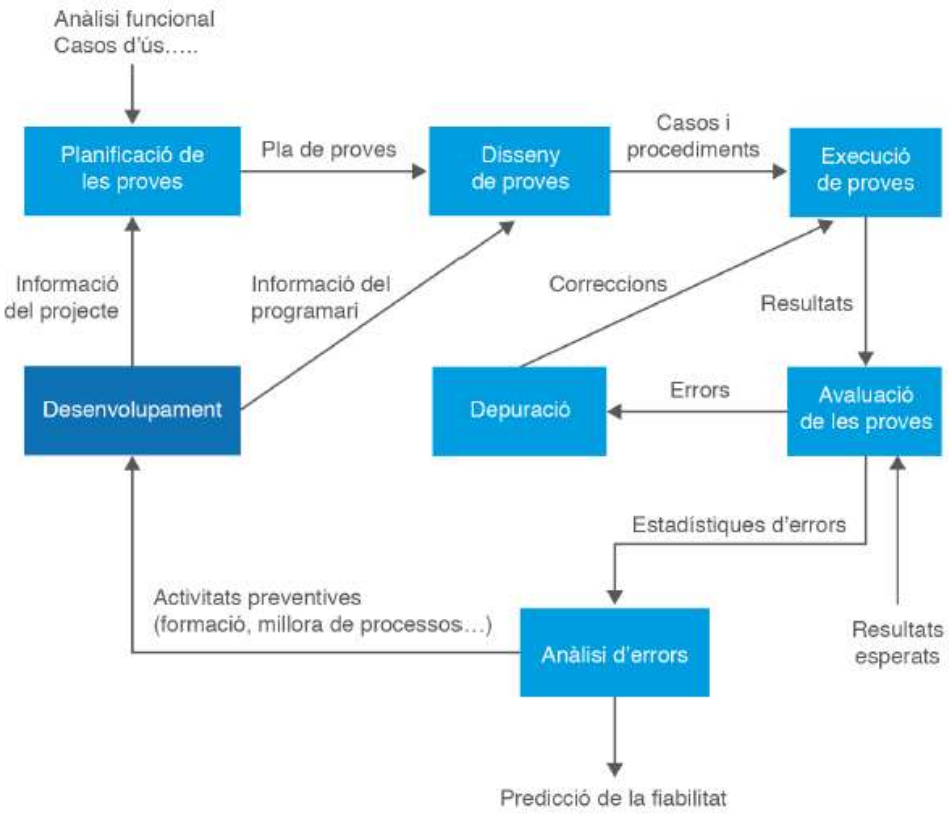

Al tiempo que se avanza en el desarrollo del software se van planificando las pruebas que se realizarán en cada fase del proyecto. Esta planificación se concretará en un plan de pruebas que se aplicará en cada producto desarrollado. Cuando se detectan errores en un producto debe volverse a la fase anterior para **depurarlo y corregirlo**; esto se indica con las flechas de vuelta de la parte izquierda de la figura.

Se conocen como **depuradores** los encargados de depurar errores en los programas.

## 6.3 Procedimientos, tipos y casos de pruebas

En cada una de las fases de un proyecto deberá dedicarse un tiempo considerable a desarrollar las tareas y procedimientos referentes a las pruebas.

Este proyecto de pruebas requerirá de una planificación, un diseño del plan de pruebas, una ejecución de las mismas y una evaluación de los resultados, con el fin analizar los errores y poder aplicar las acciones necesarias.




El esquema se inicia con una planificación de las pruebas, que tiene como punto de partida el análisis funcional, diagramas de casos de uso... del producto a desarrollar. En la planificación se estimarán los recursos necesarios para la elaboración de las pruebas y la posterior validación del software, obteniéndose un plan de pruebas como salida.

Partiendo del plan de pruebas y del código fuente que se haya desarrollado, se llevará a cabo el diseño de las pruebas identificando qué tipo de pruebas se efectuará para cada una de las funcionalidades, obteniéndose, como salida, los casos de prueba y procedimientos.

A partir de ese momento, se crea un bucle donde se ejecutarán las pruebas, se evaluarán los resultados de las pruebas efectuadas detectando los errores, se depurará el código aplicando las correcciones pertinentes y se volverán a ejecutar las pruebas.

Al finalizar el bucle, se realizará el análisis de la estadística de errores. Este análisis permitirá hacer predicciones de la fiabilidad del software, así como detectar las causas más habituales de error, con lo que se podrán mejorar los procesos de desarrollo.

### 6.3.1 Planificación de las pruebas

La planificación de las pruebas es una tarea que se debe ir desarrollando a lo largo de todas las fases del proyecto informático.

Es importante tener presente que cuanto antes se detecte un error en el proyecto informático, más fácil será contrarrestar y solucionar ese error. El **coste** de la resolución de un problema **crece exponencialmente** a medida que avanzan las fases del proyecto en las que se detecte.

Una buena guía para determinar qué contendrá un buen plan de pruebas puede obtenerse de la normativa IEEE 829-2008 “Standard for Software and System Test Documentación”. Este estándar establece cómo deberá ser la documentación y los procedimientos que se utilizarán en las diferentes etapas de las pruebas del software. Algunos de los contenidos del plan de pruebas son:

- **Identificador del plan de pruebas**. Es el identificador que se asignará al plan de pruebas.

- **Descripción del plan de pruebas**. Define el alcance del plan de pruebas, el tipo de prueba y sus propiedades, así como los elementos del software que se quieren probar.

- **Elementos del software a probar**. Determina los elementos del software que deben tenerse en cuenta en el plan de pruebas, así como las condiciones mínimas que deben cumplirse para llevarlo a cabo.

- **Elementos del software que no deben probarse**. Importante definir los elementos que no tendrán que tenerse en cuenta en el plan de pruebas.

- **Estrategia del plan de pruebas**. Define la técnica a utilizar en el diseño de los casos de prueba, como por ejemplo la técnica de caja blanca o de caja negra, así como las herramientas que se utilizarán o, incluso, el grado de automatización de las pruebas.

- **Documentos a entregar**. Define los documentos a entregar durante el plan de pruebas y al finalizarlo (resultados de los casos depruebas, especificación de las pruebas).

- **Responsables y Responsabilidades**. Se define el responsable de cada una las tareas previstas en el plan.

- **Calendario del plan de pruebas**. Diagrama de Gantt

### 6.3.2 Diseño de las pruebas. Tipo de pruebas

El diseño de las pruebas es el siguiente paso después de haber llevado a cabo el plan de pruebas. Este diseño consistirá en establecer los casos de prueba, identificando, en cada caso, el tipo de prueba que deberá efectuarse. Existen muchos tipos de pruebas:

- Estructurales o de caja blanca
- Funcionales o de caja negra
- De integración
- De carga y aceptación
- De sistema y de seguridad
- De regresión y de humo

**Casos de prueba y procedimientos**

Un **caso de prueba** define cómo se llevarán a cabo las pruebas, especificando, entre otros: el tipo de pruebas, las entradas de las pruebas, los resultados esperados o las condiciones bajo las que tendrán que desarrollarse.

Los casos de pruebas tienen un objetivo muy marcado: **identificar los errores existentes al software para que éstos no lleguen al usuario final**.

A la hora de diseñar los casos de prueba, no sólo debe validarse que la aplicación hace lo que se espera ante entradas correctas, sino que también debe validarse que tenga un comportamiento estable ante entradas no esperadas, informando de el error.

Para desarrollar y ejecutar los casos de prueba en un proyecto informático, podemos identificar dos enfoques:

- **Pruebas de caja negra**: Su objetivo es validar que el código cumple la funcionalidad definida.
- **Pruebas de caja blanca**: Se centran en la implementación de los programas para escoger los casos de prueba.

Los casos de prueba siguen un ciclo de vida clásico:

- Definición de los casos de prueba.
- Creación de los casos de prueba.
- Selección de los valores para los test.
- Ejecución de los casos de prueba.
- Comparación de resultados obtenidos con los resultados esperados.

**Tipo de pruebas**

- **Tipo de pruebas unitarias**:
  - Tienen como objetivo la detección de errores en los datos, en los algoritmos y en la lógica de éstos.
  - El método utilizado en este tipo de pruebas es el de la caja blanca o el de caja negra.

- **Tipo de pruebas funcionales:**
  - Son las encargadas de detectar los errores en la implementación de los requerimientos de usuario.

- **Tipo de pruebas de integración:**
  - Se encargan de detectar errores de las interfaces y en las relaciones entre los componentes.

- **Tipo de pruebas de carga:**
  - Se comprueba el rendimiento y la integridad de la aplicación ya terminada con datos reales y en un entorno que también simula el entorno real.

- **Tipo de pruebas de aceptación:**
  - Su objetivo es la validación o aceptación de la aplicación por parte de los usuarios.
  - **pruebas alfa y las pruebas beta**

- **Tipo de pruebas de sistema:**
  - comprobar que la integración es correcta.

- **Tipo de pruebas de regresión:**
  - Su finalidad es detectar posibles errores introducidos al haber realizado cambios en el sistema, bien para mejorarlo, bien para corregir otros errores.

- **Tipo de pruebas "de humo".** Son pruebas rápidas de las funciones básicas de un software que normalmente se realizan después de un cambio en el código antes de registrar este código modificado en la documentación del proyecto.

**Pruebas unitarias**

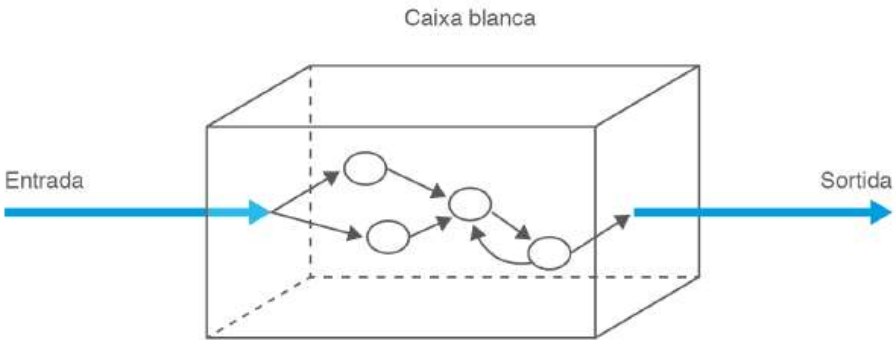
**Las pruebas unitarias, también conocidas como pruebas de componentes, son las pruebas que se harán a más bajo nivel, sobre los módulos o componentes más pequeños del código fuente del proyecto informático.**

Estas pruebas pueden desarrollarse bajo dos enfoques:

- El **enfoque estructural** (o pruebas de **caja blanca**) es la parte de las pruebas unitarias encargadas de la estructura interna del código fuente, desde el que se analizan todos los posibles caminos.
- El **enfoque funcional** (o pruebas de **caja negra**) es la parte de las pruebas unitarias encargadas del correcto funcionamiento de las funcionalidades del software.

**Enfoque estructural o de caja blanca**

**Las pruebas de caja blanca** se centran en la implementación de los programas para escoger los casos de prueba. Lo ideal sería buscar casos de prueba que recorrieran todos los caminos posibles del flujo de control del programa. Estas pruebas se centran en la estructura interna del programa, analizando todos los caminos de ejecución



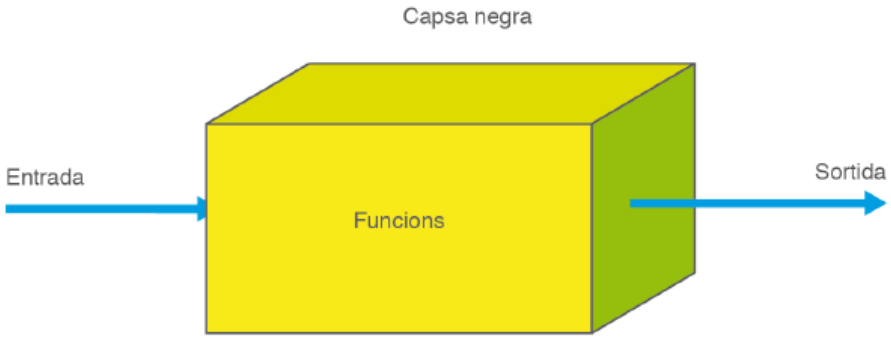

Las pruebas se llevarán a cabo con datos que garanticen que han tenido lugar todas las posibles combinaciones. Para decidir qué valores tendrán que tomar estos datos es necesario saber cómo se ha desarrollado el código, buscando que no quede ningún rincón sin revisar.

**Al menos se pasa una vez por cada camino del programa.**

**Enfoque funcional o pruebas de caja negra**

**Las pruebas de caja negra** prueban la funcionalidad del programa, para el que se diseñan casos de prueba que comprueben las especificaciones del programa.

Las técnicas de prueba de caja negra pretenden encontrar errores en funciones incorrectas o ausentes, errores de interfaz, errores de rendimiento, inicialización y finalización. Se centra en las funciones y en sus entradas y salidas.




Habrá que escoger cuidadosamente los casos de prueba, de modo que sean tan pocos como sea posible para que la prueba se pueda ejecutar en un tiempo razonable y, al tiempo, que cubran la variedad de entradas y salidas más amplia posible.

Para ello, se han diseñado diferentes técnicas:

- **Clases de equivalencia**: se trata de determinar los distintos tipos de entrada y salida, agruparlos y escoger casos de prueba para cada tipo o conjunto de datos de entrada y salida.
  - Las clases deben recoger tanto datos válidos como erróneos, ya que el programa debe estar preparado y no bloquearse bajo ninguna circunstancia.
  - Crear los casos de prueba a partir de las clases de equivalencia detectadas. Para ello se deben seguir los siguientes pasos:
    - Escoger un valor que represente cada clase de equivalencia.
    - Diseñar casos de prueba que incluyan los valores de todas las clases de equivalencia identificadas.

- **Análisis de los valores límite**: estudian los valores iniciales y finales, puesto que estadísticamente se ha demostrado que tienen mayor tendencia a detectar errores.
  - En los rangos de valores, tomar los extremos del rango y el valor intermedio.
  - Si se especifican una serie de valores, tomar el superior, el inferior, el anterior en el inferior y el posterior al superior.
  - Si el resultado se mueve en un determinado rango, debemos escoger datos en la entrada para provocar las salidas mínima, máxima y valor intermedio.
  - Si el programa elige una lista o tabla, tomar el elemento primero, el último y el intermedio.

**Pruebas de integración**

Las pruebas de integración sirven para validar que las partes de código que ya han sido probadas de forma independiente sigan funcionando correctamente al ser integradas.

**Pruebas de carga y aceptación**

Las pruebas de **carga** son pruebas que tienen como objetivo comprobar el rendimiento y la integridad de la aplicación ya terminada con datos reales. Se trata simular el entorno de explotación de la aplicación.

El objetivo de la prueba de **aceptación** es obtener la aprobación del cliente sobre la calidad de funcionamiento del sistema desarrollado y probado.

Las **pruebas alfa** consisten en invitar al cliente que venga al entorno de desarrollo a probar el sistema. Se trabaja en un entorno controlado y el cliente siempre tiene un experto a mano para ayudarle a usar el sistema y para analizar los resultados.

Las **pruebas beta** vienen después de las pruebas alfa, y se desarrollan en el entorno del cliente, un entorno que está fuera de control para el desarrollador y el equipo de trabajo. Aquí el cliente se queda solo con el producto y trata de encontrar los errores, de los que informará el desarrollador.

La experiencia muestra que estas prácticas son muy eficaces.

**Pruebas de sistema y seguridad**

Las pruebas de **sistema** servirán para validar la aplicación una vez ésta haya estado integrada con el resto del sistema del usuario. Aunque la aplicación ya haya sido validada de forma independiente, en las pruebas de sistema se llevará a cabo una segunda validación con la aplicación ya integrada en su entorno de trabajo real.

A continuación se enumeran algunos tipos de pruebas a desarrollar durante las pruebas del sistema:

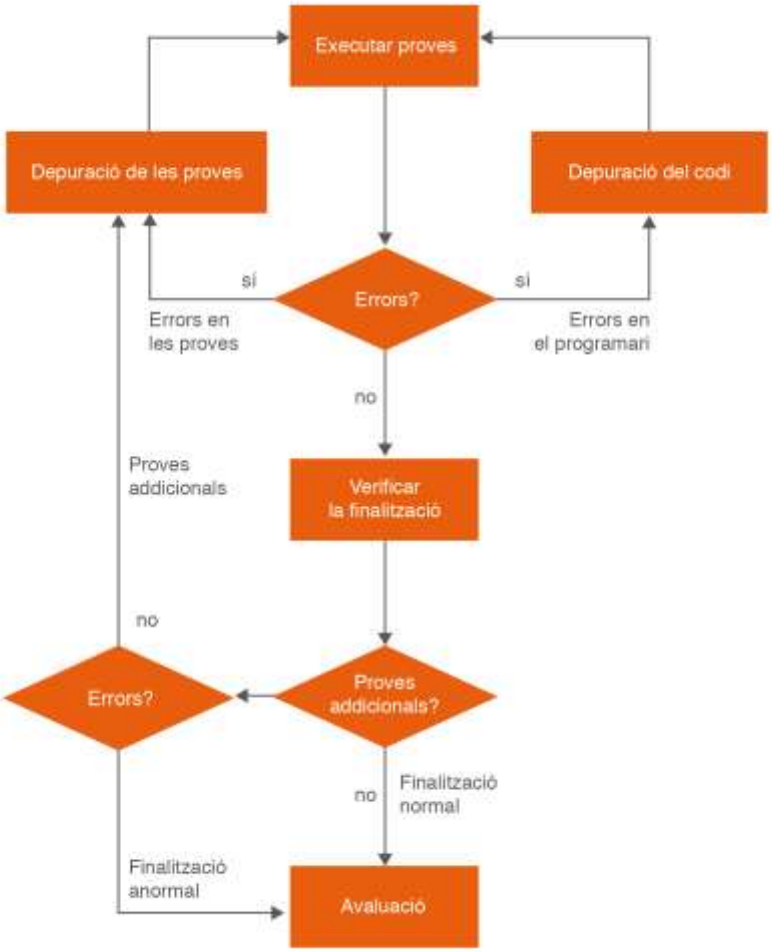
- **Pruebas de robustez**: valorarán la capacidad de la aplicación para soportar varias entradas no correctas.
- **Pruebas de seguridad**: ayudarán a determinar los niveles de permisos de los usuarios, las operaciones que podrán llevar a cabo y las de acceso al sistema y los datos.
- **Pruebas de usabilidad**: determinarán la calidad de la experiencia de un usuario en la forma de interactuar con el sistema.

**Pruebas de regresión y pruebas de humo**

Las **pruebas de regresión** buscan detectar posibles nuevos errores o problemas que puedan salir al haber introducido cambios o mejoras en el software.

Las **pruebas “de humo”** se utilizan para describir la validación de los cambios de código en el software, antes de que los cambios en el código se registren en la documentación del proyecto. Son pruebas de ejecución rápida y comprueban las funciones básicas del software.

### 6.3.3 Ejecución de las pruebas




### 6.3.4 Finalización: evaluación y análisis de errores

Efectuar una evaluación y un análisis de los errores localizados, tratados, corregidos y reevaluados.

En caso de tener que rehacer los **procedimientos de pruebas**, es muy importante la creación de nuevos casos de pruebas y no la readaptación de los ya existentes
Por último, es conveniente escribir un informe que ayude a almacenar la experiencia que se ha recogido a lo largo del procedimiento de prueba. Esta información será muy importante para futuros proyectos, ya que ayudará a no volver a repetir los mismos errores detectados. El informe deberá dar respuesta a :

- Número de casos de prueba generados.
- Número de errores detectados en cada fase del proyecto.
- Tiempo y recursos dedicados a los procedimientos de pruebas.
- Tipo de pruebas llevadas a cabo.
- Tipo de pruebas que más errores han detectado.
- Nivel de calidad del software.
- Módulos en los que más errores se han detectado.
- Errores que han llegado a los usuarios finales.
- Número de casos de prueba erróneos detectados.