DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Pruebas de integración, regresión, volumen, estrés, seguridad y recursos.

Documentar pruebas

El desarrollo de aplicaciones y sus interfaces requiere de la elaboración de un conjunto de fases, prestando especial atención a la fase de pruebas, es decir, se debe establecer una estrategia de pruebas.

En este tema, veremos todos los **tipos de pruebas** que pueden llevarse a cabo y que forman parte de una estrategia de evaluación compleja, distribuida en múltiples fases. Algunas de las pruebas más importantes son:

- las de regresión, puesto que permiten evaluar si al introducir cambios para solucionar errores se han **provocado otros**;
- las pruebas de seguridad, que permiten **garantizar la integridad** de la aplicación,
- o las pruebas capacidad y uso de recursos, que permiten evaluar el **comportamiento** de una aplicación ante diferentes escenarios, puesto que puede funcionar ante un usuario, pero no con diez, lo que supone un grave problema.

Sin el desarrollo de una buena estrategia de pruebas, el resultado raramente será exitoso, por eso, en este capítulo, veremos en detalle todos los **tipos de evaluaciones** que existen en la actualidad, así como algunas **herramientas útiles**.

Fase de pruebas en el desarrollo del proyecto

Todas las fases establecidas en el desarrollo del software son importantes. La falta o mala ejecución de alguna fase puede provocar que el resto del proyecto tenga varios errores que serán decisivos para el buen funcionamiento del proyecto.

Una aplicación informática **no puede llegar al usuario final con errores**, y menos aún si han sido detectados previamente por los propios desarrolladores. Si se diese este supuesto, la impresión por parte de los usuarios sería de **falta de profesionalidad** y su confianza en el producto y la marca disminuiría.

Para prevenir esta situación, es importante en el desarrollo del proyecto no saltarse la fase de prueba de software antes de que llegue al usuario final.

Objetivo y valoración del proceso de prueba

Las pruebas son una fase indispensable en el desarrollo de cualquier aplicación, herramienta o sistema, puesto que permiten inspeccionar el software desarrollado atendiendo a todos los posibles escenarios que se pueden contemplar.

Casi todos los desarrollos de software están basados en dos grupos de pruebas claramente diferenciados.

- Por un lado, las pruebas de **caja negra** en las cuales se evalúa la aplicación desde un **punto de vista externo**, es decir, sin preocuparnos del "interior", son las habituales para la **prueba de interfaces**.
- Por otro lado, encontramos las pruebas de **caja blanca**, estas se basan en la evaluación del **código interno** del software. Un buen diseño para estas pruebas implica la evaluación de **todos los posibles caminos** que se han implementado en el diseño de un programa.

Son múltiples las **ventajas** que aportan a cualquier desarrollo el uso de un buen sistema de pruebas:

• Permiten **validar el funcionamiento** del software en base a las especificaciones de diseño.

• Permiten **detectar errores** en el software y **corregirlos** antes de la implantación del mismo.

• Un buen sistema de pruebas también permite **evaluar** el software en **distintos escenarios** probables para el uso de la aplicación desarrollada.

Podemos diferenciar entre:

- las pruebas realizadas a nivel de módulos,
- o las pruebas de funcionamiento general, que se realizan cuando todos los **módulos** del sistema se encuentren **integrados**, puesto que lo que se pretende es validar el **funcionamiento global** de la aplicación antes de ser entregada al cliente.

Para contemplar todos los posibles escenarios, es recomendable realizar la fase de pruebas desde múltiples puntos de vista, provocando **errores habituales**, o realizando pruebas de las **acciones más comunes**.

Tipos de pruebas

Además de la clasificación genérica de pruebas de caja negra y pruebas de caja blanca, es posible dividir las fases de pruebas en diferentes tipos **atendiendo al tipo de funcionalidad evaluada** en cada caso. Algunas de las más habituales son:

- Pruebas **unitarias**:

  Este tipo de pruebas evalúan **funcionalidades concretas**, examinando todos los caminos posibles implementados en el desarrollo de un **algoritmo, función o clase**.

- Pruebas de **integración**:

  Tras realizar las pruebas unitarias, se utilizan las de integración en toda la aplicación ya totalmente integrada.

- Pruebas de **regresión**:

  Este tipo de pruebas es muy común y muy importante. Implica volver a **verificar** aquello que ya había sido **evaluado previamente** y había **generado** algún tipo de **error**. La superación con éxito de este tipo de pruebas es imprescindible para validar que los **cambios han sido correctos**.

- Pruebas de **seguridad**:

  Este tipo de pruebas se centran, sobre todo, en la evaluación de los sistemas de **protección y autenticación** de una aplicación.

- Pruebas de **volumen y de carga**:

  En algunas ocasiones, es necesario manejar una **gran cantidad de datos**, puesto que puede que el software no soporte un **acceso masivo**, por lo que será necesario comprobar este tipo de acceso. Este tipo de pruebas son especialmente útiles en **tiendas virtuales** que pueden ver **sobrecargado el servidor** ante un gran número de accesos.

- pruebas **manuales:**

  Son realizadas por **un desarrollador**, experto o no en el software a testear.

- pruebas **automáticas**:

  Utilizan herramientas que permiten agilizar este proceso de evaluación.

Depuración del código

La depuración de código, como su propio nombre indica, se centra en la revisión del código para la **detección** de **posibles errores** y la **corrección** de los mismos.

Podemos encontrar **tres tipos de errores atendiendo al desarrollo** del código:

- Errores de compilación:

  Este tipo de errores se producen por un **fallo en la sintaxis** del lenguaje de programación utilizado. Dado que **no está escrito de forma correcta**, este no puede ser interpretado por completo por el compilador y devuelve este tipo de errores. Por ejemplo, si no utilizan los ';' (punto y coma) para concluir las instrucciones, se producirá un error de compilación.

- Errores de ejecución:

  Este tipo de errores se producen cuando la **sintaxis es correcta**, pero se ha implementado algún tipo de operación cuyo **resultado es erróneo**. Son errores en tiempo de ejecución.

- Errores de lógica:

  Por último, este tipo de errores son producidos por un fallo en el diseño de la lógica del programa, es decir, el **resultado no es el esperado**. Su detección es más compleja que los anteriores, puesto que mientras que los otros tipos devuelven algún mensaje de error, este tipo de errores no lo hace. Este tipo de errores **no devuelve mensaje alguno de error**.

El diseño de un buen programa de pruebas es clave para la detección del último tipo de errores. Los dos primeros serán detectados, habitualmente, en tiempo de desarrollo.

Diagrama completo para depuración de código

Resultados →

Localizar error →

Diseñar reparaciones →

Reparar errores →

Probar de nuevo el programa →

Casos de prueba

Pruebas de integración

Las pruebas de integración permiten evaluar el funcionamiento de todos los módulos desarrollados de manera conjunta. Hasta ahora es posible que solo se hayan realizado ciertos paquetes de pruebas que incluían la evaluación de los módulos y funciones como entes aislados.

Para garantizar el correcto funcionamiento de una aplicación tras la **integración de todos sus módulos**, serán necesarias las denominadas pruebas de integración.

Podemos distinguir entre **dos tipos de pruebas de integración**:

- Pruebas de integración **ascendente**:

  Estas pruebas **comienzan** con la evaluación de los **niveles más bajos**. Este tipo de pruebas se realizan **en grupo** y requieren de un **controlador** que se encarga de la coordinación de las **salidas y entradas** de los casos de prueba.

- Pruebas de integración **descendente**:

  Estas pruebas se realizan en el sentido inverso a las anteriores, **desde el módulo principal hasta los subordinados**.

  Es habitual encontrar dos subtipos:

  - una integración **primero en profundidad**,
  - otra integración **primero en anchura**.

Pruebas de integración y pruebas de humo

*Las **pruebas de integración** son un tipo de pruebas de software. Estas pruebas buscan detectar posibles nuevos errores o problemas que puedan surgir **por haber introducido mejoras o cambios** en el software. Por esta razón **hay que llevar a cabo pruebas de integración al finalizar el resto de pruebas**.*

*Las **pruebas de humo** (ver si no echa humo...) se usan para **describir la validación de los cambios** en el software **antes de** que los cambios en el código se registren en la **documentación** del proyecto. Su origen es bastante curioso, se encuentra relacionado con la fabricación de hardware, porque antiguamente, si después de reparar un componente, éste **no "echaba humo", significaba que el funcionamiento era correcto**.*

Pruebas de sistema

Son aquellas que evalúan **todo el sistema de forma conjunta**, integrando todas las partes, pero **sin diferenciar** las pruebas entre **módulos**.

Pruebas de regresión

Se denominan pruebas de regresión a la nueva ejecución de un conjunto de pruebas que ya había sido ejecutado con anterioridad para **evaluar si las** nuevas **modificaciones y cambios** que se hayan realizado sobre el código **han podido provocar fallos** que antes no existían.

Este tipo de pruebas implica la ejecución completa de **toda la batería de pruebas** o **solo de algunas** concretas.

Tipologías en pruebas de regresión:

- Pruebas de regresión locales:

  En este caso, se localizan errores que se han producido por la introducción de **últimos cambios y modificaciones**.

- Pruebas de regresión desenmascarada o al descubierto (de pura chorra):

  Este tipo se produce cuando la introducción de cambios muestra **errores que nada tienen que ver con las modificaciones** realizadas, pero que su inclusión los ha dejado al descubierto.

- Pruebas de regresión remota o a distancia:

  Reciben este nombre los errores producidos cuando al realizar la integración de las diferentes partes de un programa, se producen errores que **de forma individual no ocurrían**.

La superación con éxito de este tipo de pruebas es imprescindible para **validar que los cambios han sido correctos**.

Pruebas funcionales

Las pruebas funcionales se basan en la evaluación de todas las **funcionalidades especificadas en los requisitos de diseño** de una herramienta o aplicación software. Es aconsejable que el proceso quede **documentado**.

Tal y como se indica en la norma ISO 25010, son deseables ciertas **características relativas a la evaluación de la funcionalidad**:

Características de pruebas funcionales

1. **Exactitud**.
2. **Seguridad**.
3. **Idoneidad**.
4. **Interoperatividad**.

(nemotécnica: easy, esii...)

Este tipo de pruebas son llevadas a cabo por **expertos** capaces de interpretar los resultados obtenidos y realizar **propuestas de modificaciones y cambios** necesarios para que la funcionalidad obtenida se adecúe a las especificaciones de diseño.

Pruebas de capacidad, rendimiento, de recursos y seguridad

Este tipo de pruebas pertenecen a las llamadas **no funcionales**. Mientras que las funcionales se centran en el comportamiento interno de la aplicación, este tipo de pruebas se utilizan para **evaluar el comportamiento externo** de la aplicación. Podemos decir que las primeras son de caja blanca y las segundas, **de caja negra**.

Tipos de pruebas

importante

Prueba de **capacidad:**

> Se utilizan para la evaluación del software y su comportamiento ante un **aumento de peticiones**, es decir, ante un **incremento de la carga** de trabajo.

Prueba de **rendimiento**:

> Se utilizan para la evaluación del **tiempo de respuesta** y la **velocidad de procesamiento** del software.

Prueba de **estrés**:

> Se utilizan para la evaluación de la **capacidad de recuperación** del software ante una **sobrecarga** de datos.

Prueba de **volumen:**

> Se utilizan para la evaluación de la **capacidad de procesamiento** del software ante la llegada de una **cantidad grande de datos**.

Pruebas de **seguridad:**

> Este tipo de pruebas están diseñadas para dos claros propósitos:
>
> - Garantizar los aspectos relativos a la **integridad** de los datos.
> - Evaluar los diferentes mecanismos de **protección** que pueden estar incorporados en una determinada aplicación software.

Pruebas manuales

Son aquellas que se realizan **por el desarrollador** para probar el código que se está implementando. En este tipo de pruebas, el propio **programador será el que introduzca los valores de entrada**, y en función de estos, se evalúa si la **salida es la esperada** al desarrollo realizado hasta el momento.

Este tipo de pruebas **no utiliza herramientas concretas**, sino que será el propio desarrollador el que lleve a cabo la **depuración** y **prueba del código** en base a la situación que se esté evaluando.

Pruebas automáticas

Utilizan herramientas que permiten **agilizar** el proceso de evaluación de las aplicaciones implementadas. Este tipo de pruebas resultan especialmente **útiles para la realización de pruebas de regresión**, puesto que **optimizan el proceso**.

Algunas de las **herramientas más comunes** para la realización de este tipo de pruebas son:

- Jmeter:

  Se trata de un proyecto de **Apache** que permite realizar **pruebas de carga** para la evaluación del **rendimiento** de una determinada aplicación.

- Bugzilla:

  Herramienta **online** utilizada para el **seguimiento de errores y defectos** del software y sus módulos, a través de las diferentes **versiones** de una misma aplicación.

- JUnit:

  Conjunto de **librerías en Java** utilizadas para la realización de **pruebas unitarias** sobre las aplicaciones desarrolladas sobre este lenguaje de programación, como las interfaces.

Herramientas de pruebas automáticas, rendimiento y unitarias

https://bit.ly/3kGKV3L

La generación de **pruebas automáticas** es una herramienta esencial para **agilizar** la evaluación de aplicaciones ya implementadas. Estas pruebas son especialmente importantes para las **pruebas de regresión**, ya que **optimizan** la implementación de dichas pruebas:

> Las pruebas de regresión permiten evaluar si los **cambios introducidos** en la aplicación para corregir errores o realizar modificaciones han generado **nuevos errores** en el desarrollo previo que **funcionaba correctamente**.

En este capítulo, nos enfocamos en tres herramientas específicas para realizar pruebas automáticas, todas comunes en el ámbito de desarrollo. A continuación, se describen brevemente estas herramientas:

1. **JUnit**:

   - JUnit es un conjunto de librerías para Java, utilizado para implementar pruebas unitarias en aplicaciones desarrolladas en Java.

   - Sitio web: https://junit.org/junit5/

   - Ofrece una **guía de usuario en inglés** que explica qué es JUnit, los **tipos de soportes** necesarios para las versiones de Java, y cómo empezar con el proceso de **descarga**.

   - Proporciona **proyectos de ejemplo** y explica la **creación de casos de prueba**.

2. **Bugzilla**:

   - Bugzilla es una herramienta que permite realizar un **seguimiento de errores y defectos** de software de los módulos a través de **diferentes versiones** de una misma aplicación.

   - Proporciona información sobre **descargas y documentación** para cada versión.

     https://www.bugzilla.org/download/

3. **JMeter**:

   - JMeter es un proyecto de **Apache** que facilita la realización de **pruebas de carga** para evaluar el rendimiento de una aplicación.

   - Sitio web: https://jmeter.apache.org/

   - Incluye una sección de **descargas y documentación**, que abarca el manual de usuario, información sobre cómo comenzar y mejores **prácticas para mejorar el rendimiento** evaluado por la herramienta.

Es fundamental destacar que la mayoría de estos recursos están en inglés. No obstante, esta característica ofrece a los desarrolladores la oportunidad de adquirir **habilidades de búsqueda de**

información, repositorios y guías, competencias esenciales para el desarrollo informático. Es crucial dedicar tiempo a explorar el amplio universo del desarrollo y aprender a localizar información en los sitios web de librerías y frameworks.

Pruebas de usuario

Las pruebas de usuarios, en contraposición a las pruebas de expertos, se basan en el **análisis y evaluación** de una herramienta o aplicación software mediante un **grupo de usuarios** reales que pueden detectar errores que los **expertos no han** sido capaces de **encontrar**.

Los métodos de test con usuarios se basan en el uso de **cuestionarios** tipo. Según el **Diseño Centrado en el Usuario** (**DCU**), los test de usuario se basan en pruebas que **observan la forma de interacción** de los usuarios con el producto objeto del test.

Por ejemplo, a un experto le puede resultar intuitiva y fácil de aprender la forma de uso de una determinada funcionalidad, pero no dejan de ser expertos en desarrollo. Desde el punto de vista de un usuario, esta forma de acometer la acción puede no ser tan intuitiva.

Es aconsejable que el número de usuarios que participen en este test sea de **al menos quince** para poder garantizar una tasa de **detección de cerca del 100%**. La elección de estos se debe basar en los **perfiles** hacia los que está dirigida la aplicación, no tendrá sentido probar una aplicación para la gestión logística de un almacén con un grupo de usuarios que no tienen ninguna vinculación con este tipo de áreas. Las pruebas se realizan **de forma individual** y se deben tener en cuenta todas las observaciones que se tomen, **desde la primera toma de contacto** hasta la realización de la prueba completa. Algunos **criterios de diseño** son:

- Pruebas razonables: es decir, que un **usuario real** realizará.
- Pruebas específicas: es aconsejable realizar pruebas concretas y **no muy genéricas**.
- Pruebas factibles: que **pueden realizarse**, no se trata de un examen que los usuarios no deben superar.
- Tiempo de realización razonable.

Pruebas de aceptación

Las pruebas de aceptación se utilizan para comprobar si una aplicación, en el caso de desarrollo de software, **cumple con el funcionamiento** contenido en las **especificaciones de diseño**, tanto desde un punto de vista **funcional** como de **rendimiento**. Es importante tener en cuenta ambos aspectos, puesto que si el producto cumple con la funcionalidad deseada, pero el tiempo necesario para completarla es excesivo, no será aceptable. Igualmente, si no cumple con la funcionalidad requerida, aunque el tiempo empleado en su uso sea óptimo, tampoco cumplirá con los criterios de aceptación.

Características de las Pruebas de Aceptación:

- Las pruebas de aceptación son definidas por los clientes: esto es así puesto que el **grado de aceptación adecuado** para un desarrollo concreto viene **determinado por los clientes** o usuarios finales del producto.
- Se ejecutan antes de la implantación final de la aplicación: este tipo de pruebas se realizan antes de ser implantadas de forma definitiva, puesto que de lo contrario, las modificaciones serán más costosas.
- Los **planes** de pruebas de aceptación han de ser correctamente documentados.

Por lo tanto, el proceso de evaluación y análisis de cualquier desarrollo antes de ser implantado de forma definitiva está constituido por una variada tipología de pruebas. La realización de todas estas es importante para ofrecer la mayor garantía posible sobre el funcionamiento de una herramienta.

En el siguiente diagrama, podemos observar uno de los **flujos habituales de ejecución de pruebas** en el desarrollo de productos software.

Diagrama de diseño de estrategia de pruebas

Unitarias
→ Integración
→ Sistema
→ Regresión
→ Funcionales
→ Capacidad
→ Recursos
→ Seguridad
→ Usuario
→ Aceptación

Flujo de pruebas para el desarrollo software. Diseño estrategia de pruebas

https://bit.ly/32V0yym

Para desarrollar y definir un buen sistema y una buena estrategia de pruebas es necesario crear diferentes tipologías en cuanto a las pruebas. No basta con una única, sino que va a ser necesario realizar diferentes tipos de pruebas, diseñar una estrategia que desde la primera vayamos pasando por el resto de pruebas, además en un orden ya establecido para poder llegar al final y entregar ese producto, subir, publicar ese producto en producción con los requisitos y especificaciones que se nos habían solicitado desde el principio.

Vamos a analizar **en qué consisten todas esas pruebas** que forman parte de una estrategia de pruebas, de esta forma vamos a establecer todos los pasos necesarios para esta estrategia.

1. En primer lugar hablamos de las pruebas unitarias:

    Son las que tendríamos en primer lugar, son pruebas que van a evaluar **funcionalidades concretas**, por ejemplo vamos a tomar un fragmento de un código, una función concreta de un código, a eso se refieren las unitarias a pequeñas funciones, pequeños fragmentos y vamos a evaluar **todos los caminos posibles** que se podrían implementar, que se podrían ocurrir en el desarrollo de un determinado algoritmo, de una función o de una clase, esto es llevado a cabo **por los desarrolladores** y se trata de realizar pruebas, imprimir por pantalla, todo lo que el desarrollador puede llegar a hacer, pero a nivel de **función, de clase, de algoritmo**.

2. Cuando éstas se han concluido pasaríamos a las pruebas de integración:

    Las pruebas de integración son las cuales vamos a probar de **forma conjunta** toda la aplicación ya totalmente integrada, en este caso que es lo que queremos decir, ya no vamos a probar una función aislada, sino que vamos a probar todo el funcionamiento en su conjunto, en este caso recordamos que tendríamos pruebas de integración **ascendente** y pruebas de integración **descendente.**

3. Además cuando continuamos las pruebas de integración pasaríamos a las pruebas del sistema:

    En este sistema son similares a las de integración, lo que ocurre es que en el caso de integración, como vemos que tenemos ascendente y descendente, lo que hacemos es evaluar las diferentes ramificaciones de toda la funcionalidad de toda la aplicación en su conjunto, ahora cuando hablamos de las de sistema, lo que hablamos es que evaluamos el sistema de forma conjunta, integrando todas las partes pero **sin diferenciar las pruebas entre módulos**, aquí radica la **diferencia entre las pruebas de sistema y de integración**, en el caso de integración estamos probando el funcionamiento de la aplicación en su conjunto pero diferenciando por módulos y en el caso del sistema estamos probando su funcionamiento pero sin distinguir entre módulos.

4. Cuando hemos concluido estas pruebas vamos a pasar a las de regresión:

> ¿Por qué? Hasta ahora en las pruebas de unitaria, integración y sistema vamos a haber encontrado errores y lo vamos a haber cambiado, ahora necesitamos hacer las pruebas de regresión, estas pruebas lo que nos permiten es evaluar si antes los cambios que hemos realizado para solucionar ciertos errores hemos **producido nuevos errores**, en la capítulo tenéis los diferentes tipos de pruebas de regresión que podemos encontrar, esto es muy importante porque si resolvemos determinados problemas pero esa solución lo que va a hacer es generarnos nuevos problemas pues podríamos no terminar nunca, entonces hay que tener mucho cuidado, estas pruebas son imprescindibles, las de regresión son muy importantes.

5. Tenemos a continuación cuando ya hemos concluido esta, hemos resuelto errores y demás vamos a ver las pruebas funcionales:

> Se basan en la evaluación de todas las funcionalidades específicas que estaban recogidas en los **requisitos de diseño** de la herramienta o del software, ya no estamos hablando de una evaluación en su conjunto sino que además ahora nos vamos a centrar en comprobar si todo aquello que aparecía en los requisitos en las especificaciones de nuestro cliente se está cumpliendo porque podemos haber diseñado una aplicación que funciona perfectamente pero no tiene nada que ver con lo que se había solicitado por parte del cliente, entonces ahora deberíamos comprobar si lo que se ha **especificado es lo que nosotros hemos construido**.

6. A continuación, bueno, esto sí que las podemos, se van a ver un poco de forma conjunta, hablamos de las pruebas de capacidad:

> las pruebas de **rendimiento, de estrés, de volumen**, de recursos, por ejemplo en el caso de las pruebas de capacidad lo que nos hace es **comprobar**, evaluar si el **comportamiento** de un software es **igual de bueno ante pocas peticiones que ante muchas**, si necesitamos implementar una tienda online y funciona para 10 usuarios pero no para 100, bueno pues deberíamos, si se prevé que va a haber 100 usuarios deberíamos volver a revisar esa parte de la implementación, tenemos pruebas de rendimiento, tenemos pruebas de estrés, en las pruebas de estrés lo que se hace es ver cómo se va a evaluar la **capacidad de recuperación** de un software cuando se sobrecargamos con datos. Todas estas pruebas, **las de capacidad y las** de recursos, son **de forma conjunta** y casi siempre lo que vamos a evaluar en este caso es cómo responde nuestro sistema **ante muchas peticiones, ante muchos datos**, modificando todas estas casuísticas.

7. Podemos observar también las pruebas de seguridad:

> Se van a basar en garantizar todos los aspectos relativos a la **integridad de los datos**, no podemos realizar una aplicación en la que los usuarios almacenen sus datos y estos datos queden públicos, entonces también hay que prestar mucho cuidado al tema de las pruebas de seguridad.

8. A continuación vamos a pasar a las pruebas de usuario:

> Son las que ya **probamos la aplicación**, siempre aquí vamos a diferenciar dos tipos de usuarios, por un lado son los usuarios **expertos** que van a evaluar errores típicos en la aplicación para ver si se han implementado bien y luego también vamos a encontrar otro tipo de usuarios que van a ser más similares a los que nos vamos a encontrar ya en la implantación, no tanto desde el punto de vista del desarrollador que va a ir buscando unos errores, sino más bien del **usuario** que no tiene por qué conocer el funcionamiento de la aplicación interno y va a realizar diferentes acciones con ello, puede ser que al realizar estas funciones no habían sido valoradas por los desarrolladores, por lo tanto no se están implementando y volveríamos a necesitar modificar esa implementación de la aplicación y de nuevo volver a pasar por todo nuestro sistema de pruebas.

7. Si también superamos las pruebas de usuario, finalmente vamos a llegar a las pruebas de aceptación:

> Concluiríamos en este paso nuestra estrategia de pruebas; en las pruebas de aceptación lo que vamos a hacer es valorar si se cumplen todas las **características propias del diseño.** ¿Qué quiere decir esto? que esté funcionando y que además se contemplen todas las características que habían sido definidas por el cliente al inicio, tendríamos un combo formado por la aplicación **funcione perfectamente** y además todo lo que en ella se puede realizar **coincida** con lo que había sido **solicitado por un usuario**.

Podemos ver a través de este diagrama que la definición de una estrategia de pruebas no es trivial, tampoco se puede realizar en 5 minutos, sino que requiere de un algoritmo de pasos al que hay que prestar especial atención.

Versiones alfa y beta

Las pruebas alfa y beta son aquellas que permiten **encontrar aquellos errores propios en el cliente final**. Es decir, hasta ahora se han desarrollado pruebas desde el punto de vista del experto, del desarrollador o a través de los usuarios, pero siguiendo un guión pautado durante la prueba.

Existen errores que solo van a ser descubiertos cuando se simule el **funcionamiento habitual** de la aplicación, por lo tanto, en estos casos, lo que se hace es probar la herramienta al completo creando para ellos **versiones muy parecidas a las definitivas**, pero que van a servir como pruebas con las llamadas **versiones alfa y beta**.

Versión alfa:

> La versión alfa de un producto, desarrollo de herramientas o aplicaciones consiste en la **primera versión de la aplicación**. Esta será probada por un **grupo de individuos** que **simulan ser el cliente final**. Esta versión aún **no está preparada para** su implantación en **producción**, sino que debe ser evaluada previamente por un grupo de **expertos que simulan ser clientes**.
>
> Este tipo de pruebas se realizan, habitualmente, **desde las oficinas** de trabajo donde un producto ha sido desarrollado, puesto que **aún no ha sido entregado al cliente**.

Versión beta:

> La versión beta, al igual que la anterior, es una versión casi definitiva del producto, en este caso, se trata de la **primera** versión de un producto que será **probada por los clientes** que habían realizado el encargo del desarrollo. A diferencia de la versión alfa, las versiones beta sí serán evaluadas por los clientes y otros **usuarios ajenos al desarrollo**.
>
> Además, lo habitual será hacerlo fuera del entorno de desarrollo para evitar condicionamientos en el manejo de la aplicación por parte del grupo de desarrolladores. Aparecen los llamados **usuarios beta tester**, que son los encargados de **comprobar que todo funciona** correctamente, si no es así, también serán los que **indiquen los fallos** producidos, sobre todo **antes de hacer pública la aplicación** (cuando sería más costoso la corrección de estos fallos).

Estrategia de diseño

Estrategia de diseño completa para la creación de una aplicación.

- Fase de planificación: En esta, se definen los **criterios y necesidades** básicas de la aplicación.
- Fase de diseño: Durante esta fase se realizará el diseño **en base a los criterios** definidos en el apartado anterior.
- Fase de implementación: En esta fase, se escribe y desarrolla el **código** diseñado en los apartados anteriores.
- Fase evaluación: Fase de **pruebas** para evaluar el estado de la aplicación antes de ser entregada al cliente. Al concluir esta fase, será necesario **regresar a la fase anterior**, en el caso de haber aparecido **errores** o **cambios**.
- Fase de producción: En esta fase se **entrega** el desarrollo a los **usuarios** que la utilizarán para el fin que estaba diseñada.
- Fase de mantenimiento: Tras ser entregada la aplicación, es posible incorporar **nuevas funcionalidades** o **resolver casuísticas** que no se habían tenido en cuenta durante su diseño e implementación.

Por lo tanto, todas las fases son necesarias para optimizar la creación de una aplicación, tanto en tiempo como en coste económico.

Si, por ejemplo, omitimos la fase de evaluación, cuando esta suba a producción, es decir, esté en funcionamiento, podrán aparecer errores que no se habían detectado al no ser evaluada, y se requerirá volver a la fase de evaluación para determinar todos los errores o cambios necesarios, y, a continuación, regresar a la fase de implementación para solventarlos, incluso a la de diseño, puesto que puede que el cambio en una parte de la aplicación, implique el cambio en otros elementos del desarrollo .

Estrategia de pruebas

Tras describir la secuencia de fases necesarias para el desarrollo de una aplicación, nos centramos en la **fase de pruebas o fase de evaluación**, la cual es utilizada para evaluar el estado de la aplicación antes de ser entregada al cliente que solicitó su desarrollo.

Al concluir esta fase, será necesario regresar a la fase anterior en el caso de haber aparecido errores o cambios.

Se pide diseñar un plan de pruebas en el que indiques de forma estimada los tiempos que se van a asignar a cada una de ellas, puesto que siempre se va a trabajar con plazos de entrega que han de estar convenientemente acotados.

Se realiza un **listado de todas las pruebas** que se van a realizar para evaluar una aplicación.

Por ejemplo: pruebas unitarias, pruebas de integración, pruebas de sistema, pruebas de regresión, pruebas funcionales, pruebas de capacidades y recursos, pruebas de usuario y pruebas de aceptación.

Para el diseño de este calendario de tiempos, se debe tener en cuenta la **casuística de cada prueba**:

- Pruebas **unitarias**: Es conveniente que estas pruebas se realicen de **forma paralela al desarrollo** de la aplicación, por lo tanto, no se le asigna un intervalo específico durante la fase de evaluación, pero sí debe tenerse en cuenta en la **estimación del tiempo de desarrollo**.
- Pruebas de **integración**: Estas son muy importantes, ya que se evalúa el comportamiento de la aplicación **por módulos**, así que, de nuevo, sería recomendable realizarlas **en paralelo al desarrollo** de la aplicación.
- Pruebas de **sistema**: Estas pruebas evalúan **todo el comportamiento** de la aplicación de forma completa, sería conveniente dedicarle un **25% del tiempo** reservado para pruebas.
- Pruebas de **regresión**: Las pruebas de regresión evalúan si las modificaciones han creado **nuevos cambios**, y sería conveniente reservar un **15% del total**.
- Pruebas **funcionales**: Funcionamiento general en base a las **especificaciones**, en torno a un **15% del tiempo**.
- Pruebas de **capacidades y recursos**: En función del tipo de aplicación, sería necesario definir un intervalo concreto, entre **10% y el 20%**.
- Pruebas de **usuario**: Claves para cualquier desarrollo, **al menos un 20%**.
- Pruebas de **aceptación**: Se trata de volver a evaluar de nuevo la herramienta, hasta aquí debería funcionar, ahora solo debemos verificar si todo lo **especificado está implementado**, un **10% de tiempo**.

Además de la clasificación genérica de pruebas de caja negra y pruebas de caja blanca, es posible dividir las **fases de pruebas** en diferentes **tipos** atendiendo al tipo de funcionalidad evaluada en cada caso. Algunas de las **más habituales son**: pruebas **unitarias**, de **integración**, **regresión**, **seguridad**, **volumen** y **carga**.

En ocasiones, de manera errónea, se usan bucles en los que sus condiciones para su funcionamiento no tienen un final, por lo que el número de alternativas y combinaciones posibles para desarrollar las pruebas pasa a ser exponencial.

Cuanto más tiempo se tarda en detectar y solucionar un error en el código, más costes conllevará solucionarlo, y la gráfica puede llegar a ser exponencial. Además, la **mayor parte de los errores se concentran en las primeras fases** del proyecto de creación de aplicaciones, por lo que es importante **no descuidar** estas fases.