• ¿Qué es testing?

El testing es la prueba de que todo funcione de acuerdo a lo esperado y en caso contrario avisar para que esto pueda solucionarse.

Testing = calidad

En Testing, calidad se trata del grado de satisfacción del cliente (cumplir sus expectativas y satisfacerlas sin excedernos en tiempo y presupuesto).

• Los siete principios del testing

1) La prueba muestra la presencia de defectos, no su ausencia◊no puede probar que no hay defectos. Incluso si no se encuentran, el proceso de prueba no demuestra una corrección.

2) La prueba exhaustiva es imposible◊ en su lugar debe utilizarse análisis de riesgos, técnicas de prueba y prioridades para centrar los esfuerzos de prueba.

3) La prueba temprana ahorra tiempo y dinero◊ las actividades de testing deben iniciarse lo antes posible en el ciclo de vida de desarrollo de software para ayudar a reducir o eliminar cambios costosos.

4) Los defectos se agrupan

5) Cuidado con la prueba del pesticida ◊ no deben repetirse las mismas pruebas, sino que se deben cambiar como también cambiar los datos de prueba existentes para evitar no poder encontrar nuevos defectos.

6) La prueba se realiza de manera diferente según el contexto

7) La ausencia de errores es una falacia ◊ el éxito de un sistema no depende solamente de encontrar errores y corregirlos, puede no haber errores, pero si otros problemas.

• Validación vs Verificación

La VERIFICACIÓN comprueba si los requisitos han sido cumplidos o no (¿construimos bien el software?). La VALIDACIÓN es una comprobación, pero de que el software cumpla el uso previsto, esperado (¿construimos el software adecuado?).

• Mesa de 3 patas

Se interpreta a cada actor como parte integrante de una mesa de 3 patas. A la mesa no le puede faltar ninguna de sus patas, porque de lo contrario, no podría estar de pie.
Si bien cada actor tiene un rol definido, es necesario un trabajo en comunión entre los 3 actores. Es decir, es necesario que trabajen como equipo.

- Business analyst / Analista de negocio

Se encarga de detectar los factores clave del negocio y es el intermediario entre el departamento de sistemas y el cliente final.

-QA

La principal función es probar los sistemas informáticos para que funcionen correctamente de acuerdo a los requerimientos del cliente, documentar los errores encontrados y desarrollar procedimientos de prueba para hacer un seguimiento de los problemas de los productos de forma más eficaz y eficiente.

- Software developer / Desarrollador de software

Su función es diseñar, producir, programar o mantener componentes o subconjuntos de software conforme a especificaciones funcionales y técnicas para ser integrados en aplicaciones.

- ¿Qué es un defecto?

Un error se produce por equivocación y causa un defecto que luego producirá un fallo al ejecutarse. Cuando se detecta un defecto —como parte de las pruebas estáticas—, o se observa un fallo —como parte de las pruebas dinámicas—, la persona implicada debería recopilar los datos e incluirlos en el informe de defectos. Esta información debería ser suficiente para tres fines:

- Gestión del informe durante el ciclo de vida de los defectos.
- Evaluación del estado del proyecto, especialmente en términos de calidad del producto y progreso de las pruebas.
- Evaluación de la capacidad del proceso.

Los datos necesarios para la gestión de los informes de defectos y el estado del proyecto pueden variar en función de cuándo se detecta el defecto en el ciclo de vida, siendo la información requerida menor en etapas anteriores.

- Ciclo de vida de un defecto

Es el proceso que gestiona un defecto desde su descubrimiento hasta su solución. En cada estado solo existe un responsable del defecto –excepto en estados terminales (cerrado, duplicado) debido a que no se van a realizar más acciones.

*Estados:

Nuevo-inicial◊se recopila la información y se registra el defecto.

Asignado: si es un defecto valido y debe solucionarse, se asigna al equipo de desarrollo, sino se puede rechazar o diferir.

-Duplicado: si el defecto se repite o existe otro con una misma causa raíz.

-Devuelto o rechazado: se solicita más información o el receptor rechaza el defecto.

-Diferido: el defecto no es prioritario y se solucionará en una próxima versión.

En progreso: se analiza y trabaja en la solución.

Corregido: se realizan los cambios de código para solucionar el defecto.

En espera de verificación: en espera de que sea asignado a un probador. El desarrollador está a la expectativa del resultado de la verificación.

En verificación: el probador ejecuta una prueba de confirmación.

-Reabierto: la prueba de confirmación indica que el defecto no se ha solucionado.

Verificado: se obtiene el resultado esperado en la prueba de confirmación.

Cerrado: el defecto fue corregido y se encuentra disponible para el usuario final.

- ## Gestión de defectos

1) Detectar
2) Registrar: varía según el contexto del componente o sistema, nivel de prueba y modelo de desarrollo elegido.
3) Investigación y seguimiento
4) Clasificación/ resolución

*Escribir un informe de defectos: Mientras mejor se reporte el defecto, más probabilidades de encontrar una solución rápidamente.

*Condiciones a tener en cuenta para ello:

-Los bugs deben tener identificadores únicos.

-Una falla debe ser reproducible para reportarla: si el defecto no es reproducible no es un defecto.

-Ser específico: incluir solo la información relevante para poder reproducir el defecto. (Ahorrarse las suposiciones o ideas de lo que está ocurriendo).

-Reportar cada paso realizado para reproducirlo: información necesaria para que el desarrollador pueda reproducir la falla. No debe obviarse ningún paso relevante para llegar al error en cuestión.

- ## Diseño de la prueba

Es un documento en el cual se incluirán todas las acciones que se ejecutan para verificar una característica o funcionalidad particular de una aplicación de software. Uno o más casos de prueba representarán las características de una aplicación de software. Es un documento escrito que proporciona información sobre qué y cómo probar.

*Contenido de un caso de prueba:

Identificador: numérico o alfanumérico.

Nombre del CP: conciso.

Descripción: que es lo que se va a probar, el ambiente de pruebas y datos necesarios para ejecutarlo.

Precondición: aquello que deba cumplirse antes de ejecutar el CP.

Pasos: acciones a realizar para obtener los resultados.

Resultados esperados: lo que indica al probador cual experiencia debería ser al ejecutar los pasos y determinar si el test falló o no.

*Estados de un CP:

De diseño: utilizado durante la etapa de diseño de la prueba.

-En diseño.

-Diseñado.

-En revisión.

-Revisado.

De ejecución: utilizado durante la etapa de ejecución de la prueba.

-No corrido.

-Pasado.

-Fallado.

-No aplica.

-Bloqueado.

-No completo.

-Diferido.

*Características de un buen CP

-ser simples -título fuerte.

-tener en cuenta al usuario final: que los CP cumplan con los requisitos del cliente y sean fáciles de usar.

-no asumir: funcionalidad y características de la aplicación al preparar el CP, sino que se debe ser fiel a los documentos de especificación y consultar ante la duda.

-asegurar la mayor cobertura posible: de todos los requisitos especificados.

-autonomía: generar los mismos resultados con independencia de quien lo pruebe.

-evitar la repetición de CP: si se necesita un CP para ejecutar otro, indicar el primero por su ID.

- Niveles y tipos de pruebas

Niveles:

- Prueba unitaria o de componente:

- **Objetivos específicos**:

  - ✓ Reducir el riesgo.
  - ✓ Verificar que los comportamientos funcionales y no funcionales del componente son los diseñados y especificados.
  - ✓ Generar confianza en la calidad del componente.
  - ✓ Encontrar defectos en el componente.
  - ✓ Prevenir la propagación de defectos a niveles de prueba superiores.

- **Bases de prueba**: Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba incluyen:

  - ✓ Diseño detallado.
  - ✓ Código.
  - ✓ Modelo de datos.
  - ✓ Especificaciones de los componentes.

- **Objeto de prueba**: Los objetos de prueba característicos para la prueba de componente incluyen:

  - ✓ Componentes, unidades o módulos.
  - ✓ Código y estructuras de datos.
  - ✓ Clases.
  - ✓ Módulos de base de datos.

- **Defectos y fallos característicos**: Ejemplos de defectos y fallos característicos de la prueba de componente incluyen:

  - ✓ Funcionamiento incorrecto —por ejemplo, no lo hace de la manera en que se describe en las especificaciones de diseño—.
  - ✓ Problemas de flujo de datos.
  - ✓ Código y lógica incorrectos.

- **Enfoques y responsabilidades específicas**:

✓ En general, el desarrollador que escribió el código realiza la prueba de componente. Los desarrolladores pueden alternar el desarrollo de componentes con la búsqueda y corrección de defectos. A menudo, estos escriben y ejecutan pruebas después de haber escrito el código de un componente. Sin embargo, especialmente en el desarrollo ágil, la redacción de casos de prueba de componente automatizados puede preceder a la redacción del código de la aplicación.

- Prueba de integración:

- **Objetivos específicos**: La prueba de integración se centra en las interacciones entre componentes o sistemas:

  - ✓ Reducir el riesgo.
  - ✓ Verificar que los comportamientos funcionales y no funcionales de las interfaces sean los diseñados y especificados.
  - ✓ Generar confianza en la calidad de las interfaces.
  - ✓ Encontrar defectos —que pueden estar en las propias interfaces o dentro de los componentes o sistemas—.
  - ✓ Prevenir la propagación de defectos a niveles de prueba superiores.

- **Bases de prueba:** Algunos ejemplos de productos de trabajo que pueden utilizarse como base de prueba incluyen:

  - ✓ Diseño de software y sistemas.
  - ✓ Diagramas de secuencia.
  - ✓ Especificaciones de interfaz y protocolos de comunicación.
  - ✓ Casos de uso.
  - ✓ Arquitectura a nivel de componente o de sistema.
  - ✓ Flujos de trabajo.
  - ✓ Definiciones de interfaces externas.

- **Objeto de prueba:** Los objetos de prueba característicos para la prueba de integración incluyen:

  - ✓ Subsistemas.
  - ✓ Bases de datos.
  - ✓ Infraestructura.
  - ✓ Interfaces.
  - ✓ Interfaces de programación de aplicaciones —API por sus siglas en inglés—.
  - ✓ Microservicios.

- **Defectos y fallos característicos:**

  - ✓ Datos incorrectos, datos faltantes o codificación incorrecta de datos.
  - ✓ Secuenciación o sincronización incorrecta de las llamadas a la interfaz.
  - ✓ Incompatibilidad de la interfaz.
  - ✓ Fallos en la comunicación entre componentes.
  - ✓ Fallos de comunicación entre componentes no tratados o tratados de forma incorrecta.
  - ✓ Suposiciones incorrectas sobre el significado, las unidades o las fronteras de los datos que se transmiten entre componentes.

- Enfoques y responsabilidades específicas:

- ✓ La prueba de integración debe concentrarse en la integración propiamente dicha. Se puede utilizar los tipos de prueba funcional, no funcional y estructural. En general es responsabilidad de los testers.

o Prueba de sistema:

- **Objetivos específicos**:

- ✓ Reducir el riesgo.
- ✓ Verificar que los comportamientos funcionales y no funcionales del sistema son los diseñados y especificados.
- ✓ Validar que el sistema está completo y que funcionará como se espera.
- ✓ Generar confianza en la calidad del sistema considerado como un todo.
- ✓ Encontrar defectos.
- ✓ Prevenir la propagación de defectos a niveles de prueba superiores o a producción.

- **Bases de prueba**: Algunos ejemplos de productos de trabajo que se pueden utilizar como base de prueba incluyen:

- ✓ Especificaciones de requisitos del sistema y del software —funcionales y no funcionales—.
- ✓ Informes de análisis de riesgo.
- ✓ Casos de uso.
- ✓ Épicas e historias de usuario.
- ✓ Modelos de comportamiento del sistema.
- ✓ Diagramas de estado.
- ✓ Manuales del sistema y del usuario.

- **Objeto de prueba**:

- ✓ Aplicaciones.
- ✓ Sistemas hardware/software.
- ✓ Sistemas operativos.
- ✓ Sistema sujeto a prueba (SSP).
- ✓ Configuración del sistema y datos de configuración.

- **Defectos y fallos característicos**:

- ✓ Cálculos incorrectos.
- ✓ Comportamiento funcional o no funcional del sistema incorrecto o inesperado.
- ✓ Control y/o flujos de datos incorrectos dentro del sistema.
- ✓ Incapacidad para llevar a cabo, de forma adecuada y completa, las tareas funcionales extremo a extremo.
- ✓ Fallo del sistema para operar correctamente en el/los entorno/s de producción.
- ✓ Fallo del sistema para funcionar como se describe en los manuales del sistema y de usuario.

- **Enfoques y responsabilidades específicas**:

- ✓ La prueba de sistema debe centrarse en el comportamiento global y extremo a extremo del sistema en su conjunto, tanto funcional como no funcional. Deben utilizar las técnicas más apropiadas para los aspectos del sistema que serán probados. Los probadores independientes, en general, llevan a cabo la prueba de sistema.

- Prueba de aceptación:

- **Objetivos específicos**: La prueba de aceptación, al igual que la prueba de sistema, se centra normalmente en el comportamiento y las capacidades de todo un sistema o producto. Los objetivos de la prueba de aceptación incluyen:

  - ✓ Establecer confianza en la calidad del sistema en su conjunto.
  - ✓ Validar que el sistema está completo y que funcionará como se espera.
  - ✓ Verificar que los comportamientos funcionales y no funcionales del sistema sean los especificados.

- **Bases de prueba**: Entre los ejemplos de productos de trabajo que se pueden utilizar como base de prueba se encuentran:

  - ✓ Procesos de negocio.
  - ✓ Requisitos de usuario o de negocio.
  - ✓ Normativas, contratos legales y estándares.
  - ✓ Casos de uso.
  - ✓ Requisitos de sistema.
  - ✓ Documentación del sistema o del usuario.
  - ✓ Procedimientos de instalación.
  - ✓ Informes de análisis de riesgo.

- **Objeto de prueba**:

  - ✓ Sistema sujeto a prueba.
  - ✓ Configuración del sistema y datos de configuración.
  - ✓ Procesos de negocio para un sistema totalmente integrado.
  - ✓ Sistemas de recuperación y sitios críticos —para pruebas de continuidad del negocio y recuperación de desastres—.
  - ✓ Procesos operativos y de mantenimiento.
  - ✓ Formularios.
  - ✓ Informes.
  - ✓ Datos de producción existentes y transformados.

- **Defectos y fallos característicos**: Entre los ejemplos de defectos característicos de cualquier forma de prueba de aceptación se encuentran:

  - ✓ Los flujos de trabajo del sistema no cumplen con los requisitos de negocio o de usuario.
  - ✓ Las reglas de negocio no se implementan de forma correcta.
  - ✓ El sistema no satisface los requisitos contractuales o reglamentarios.

- ✓ Fallos no funcionales tales como vulnerabilidades de seguridad, eficiencia de rendimiento inadecuada bajo cargas elevadas o funcionamiento inadecuado en una plataforma soportada.

- **Enfoques y responsabilidades específicas**:

- ✓ A menudo es responsabilidad de los clientes, usuarios de negocio, propietarios de producto u operadores de un sistema, y otros implicados también pueden estar involucrados. La prueba de aceptación se considera, a menudo, como el último nivel de prueba en un ciclo de vida de desarrollo secuencial.

Tipos de Prueba:

1) funcional:
Describen QUE HACE el sistema. Evalúan las funciones que el sistema debe realizar. Podemos hacernos preguntas sobre cómo funciona, qué debe estar haciendo y cómo están interactuando los usuarios.
-Implementación: observa el comportamiento del software.
-Niveles de prueba en que se puede realizar: todos.
-Alcance: los requisitos funcionales pueden estar detallados en: especificaciones de requisitos del negocio; épicas; historias de usuarios; casos de uso y/o especificaciones funcionales.
-Cobertura: es la medida en que algún tipo de elemento funcional ha sido practicado por pruebas, y se expresa como un porcentaje del tipo o tipos de elementos cubiertos.

2) no funcional:
Prueba COMO (qué tan bien) SE COMPORTA el sistema. Otro tipo de problemas que no son detectados por las pruebas anteriores. Factores como lentitud, problemas con la combinación de colores.
-Implementación: puede implicar competencias y conocimientos especiales (Ej.: conocimiento de debilidades inherentes a un diseño o tecnología).
-Niveles de prueba en que se puede realizar: todos.
-Alcance: evalúa características como la usabilidad, la eficiencia del desempeño o la seguridad.
-Cobertura: es la medida en que algún tipo de elemento no funcional ha sido practicado por pruebas, y se expresa como un porcentaje del tipo o tipos de elementos cubiertos.

3) estructural:

Se basan en la estructura interna del sistema o su implementación (puede incluir código, arquitectura, flujos de trabajo y/o flujos de datos dentro del sistema).

-Implementación: puede implicar competencias y conocimientos especiales (Ej.: forma en que se construye el código; como se almacenan los datos y como utilizar las herramientas de cobertura e interpretar correctamente sus resultados).

- Niveles de prueba en que se puede realizar: de componente y de integración.

-Alcance: puede basarse en la arquitectura del sistema como en las interfaces entre componentes.

-Cobertura: es la medida en que algún tipo de elemento estructural ha sido practicado mediante pruebas, y se expresa como un porcentaje del tipo de elemento cubierto.

4) asociada al cambio:

2 tipos:

*de confirmación: corregido un defecto, el software se puede probar con todos los casos de prueba que fallaron (debido al defecto) que se deben volver a ejecutar en la nueva versión de software. Busca confirmar que el defecto original se ha solucionado satisfactoriamente.

*de regresión: implica la realización de pruebas para detectar efectos secundarios no deseados (a raíz de cambios realizados en alguna parte del código al efectuarse una corrección o algún cambio).

-Implementación: particularmente en los ciclos de vida de desarrollos iterativos e incrementales (Ej.: agile) las nuevas características existentes y la refactorización del código dan como resultado cambios frecuentes en el código, lo que también requiere pruebas asociadas al cambio.

-Niveles de prueba en que se puede realizar: ambas (confirmación y regresión) en todos. ---Cobertura: los juegos de prueba de regresión se ejecutan muchas veces y generalmente evolucionan lentamente, por lo que la prueba de regresión es un fuerte candidato para la automatización. La cobertura crece a medida que se agregan más funcionalidades al sistema por lo tanto más pruebas de regresión.

- ## Técnicas de prueba

Son el punto de partida para el diseño de nuestros casos de pruebas. Permiten tener mayor cobertura a la hora de diseñarlos. Ayudan a identificar las condiciones, los casos y los datos de prueba.

-De caja negra: el elemento es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. Se basan en el comportamiento extraído del análisis de los documentos que son base de prueba (documentos de requisitos formales, casos de uso, historias de usuario).
*aplicables para pruebas funcionales y no funcionales.
*partición de equivalencia.
*análisis de valores límites.
*tabla de decisión.
*transición de estados.

-De caja blanca: se mira el código y la estructura del producto que se va a probar y se usa ese conocimiento para la realización de las pruebas. Se basan en la estructura extraída de los documentos de arquitectura, diseño detallado, estructura interna o código del sistema. Se concentran en el procesamiento dentro del objeto de prueba.

-Basadas en la experiencia: aprovechan el conocimiento de desarrolladores, probadores y usuarios para diseñar, implementar y ejecutar las pruebas.
*predicción de errores.
*prueba exploratoria.
*prueba basada en listas de comprobación.

- Implementación y ejecución de la prueba

-Implementación ¿Está todo preparado para realizar la prueba? Algunas de sus actividades principales son crear y/o completar los productos de trabajo necesarios para la ejecución de la prueba, preparar el entorno de prueba —incluidos datos o configuraciones—, organizar las suites de pruebas en un calendario de ejecución, entre otros.
-Ejecución de la prueba Al completarse la implementación de los cambios o nuevas funcionalidades solicitados por el cliente y ya diseñados los casos de pruebas correspondientes, estamos en la etapa de ejecutar las pruebas como parte de la fase de pruebas del ciclo de desarrollo de software (SDLC). Las tareas dentro de la ejecución se llevan a cabo en forma iterativa hasta conseguir un sistema lo más estable posible.

- Creación de suites

Son necesarias para el funcionamiento saludable del producto en construcción y relevantes para la calidad final del producto.

De humo: se ejecutan para evaluar la estabilidad de las compilaciones de software iniciales o desarrolladas recientemente. Cubren las funcionalidades principales. *son previas a las de regresión –Si se encuentra algún problema durante las pruebas de humo, la compilación no se encuentra estable por lo que retorna al equipo de desarrollo hasta que lo sea. Una vez que nos encontramos en una versión estable del sistema, se llevan a cabo las pruebas de regresión sobre las funcionalidades existentes siguiendo la planificación prevista.

De regresión: verificar y validar las funcionalidades existentes de la aplicación después de cada modificación o en la adición de nuevas funciones. Para poder verificar que todo se haga sin fallas. Termina de confirmar que todo lo que antes del despliegue funcionaba, funcionan a la perfección.

- Pruebas estáticas y dinámicas

Estas se complementan entre si u permiten entregar un software con la mejor calidad posible. Proporcionar una evaluación de calidad de los productos de trabajo e identificar defectos en forma temprana.

Estáticas: Se basa en la evaluación manual de los productos de trabajo (revisiones) o en la evaluación basada en herramientas del código u otros productos de trabajo (análisis estático). Este tipo de pruebas no requieren la ejecución del software que se está probando.

Dinámicas: Requieren la ejecución del software, componente o sistema. –se complementan con las pruebas estáticas debido a que encuentra diferentes tipos de defectos. Para la generación de casos de prueba se utilizan diferentes técnicas de caja negra, caja blanca o basadas en la experiencia de usuario.

- ## Organización de la prueba - Pruebas en entornos de desarrollo.

¿Qué pasa si luego se debe realizar un cambio imprevisto o si se visualiza un error que necesita ser corregido? No podemos hacer los cambios directamente sobre el ambiente en el que el cliente está utilizando el software, porque podríamos romperlo y dejarlo inoperativo. Deben existir diferentes ambientes de trabajo, donde se pueda desarrollar y probar los cambios antes de que llegue al ambiente del cliente.

Ambiente de trabajo: Entorno con todos los recursos necesarios para que se pueda ejecutar un sistema. Hacen referencia a un servidor con ciertos recursos asignados, software y librerías instalados, su propia base de datos y una configuración determinada. Permite desarrollar aplicaciones de forma segura y con entornos diferenciados para realizar la programación, realizar pruebas, compartir resultados con los clientes y permitirles realizar pruebas y prácticas; y finalmente publicar una aplicación robusta y estable. Cada ambiente es utilizado con un fin específico y presenta ciertas ventajas sobre otros ambientes, en determinado momento del proceso de trabajo.

1. Ambiente de desarrollo o DEV: este entorno es el espacio de trabajo donde el programador desarrolla el código de la aplicación, realiza pruebas iniciales y comprueba si la aplicación se ejecuta correctamente con ese código.
2. Ambiente de pruebas o QA: el entorno de pruebas suele estar ubicado en un servidor en la nube o en una granja de servidores locales (laboratorio). Permite minimizar incidencias en etapas posteriores, ya que el tester ejecutaría las primeras pruebas de funcionalidad en este ambiente.
3. Ambiente de UAT: el entorno de UAT (o de pruebas de aceptación de usuario) permite a los usuarios del cliente poder verificar que los cambios realizados son los que realmente se solicitaron, evaluando a su vez accesibilidad y usabilidad.
4. Ambiente de preproducción o STAGE: este entorno debería poseer una configuración técnica idéntica a la que nos encontraremos en el entorno de producción. El propósito principal de este entorno es emular al entorno de producción con el fin de probar las actualizaciones y asegurar que estas no corromperán la aplicación en los servidores en producción cuando sean desplegadas. De esta forma se minimizan las caídas del sistema y corte de los servicios en producción.
5. Ambiente de producción o PROD: este es el entorno donde finalmente se ejecuta la aplicación, donde acceden los usuarios finales y donde se trabaja con los datos reales de negocio. Es un servidor que posee las mismas características y configuración que tendrá el servidor de preproducción.

Aunque, en este caso, puede estar configurado por más de un servidor, para efectos de balanceo de carga en aplicaciones que requieren una infraestructura con capacidad de manejar un tráfico de usuarios pesado y miles de conexiones concurrentes