TESTING I

Fundamentos de testing y Gestión de Defectos

El **TESTING** se realiza en distintos niveles de prueba para asegurar la **calidad** del producto o sistema.

¿Qué son los niveles de prueba? → Usamos el Modelo en V para eso.

Modelo en V → Es un modelo empleado en distintos procesos de desarrollo. Este modelo también define los procedimientos y gestión de la calidad que lo acompañan. Este modelo compara las fases de desarrollo con las fases de calidad correspondientes.



Dentro de estas etapas hay diferentes niveles de control de calidad, o llamadas tambien de verificación.



**Verificación → ¿Estamos construyendo el producto correctamente?**

**Validación → ¿Estamos construyendo el producto correcto?**

El Método en V abarca pruebas de:

* Componente (se aísla cada parte del programa, para comprobar que cada una de las partes por separado, funciona bien)
* Integración (se realiza sobre un conjunto de módulos, se prueba la interacción de estos)
* Sistema (quiere demostrar que el sistema cumple con los requisitos funcionales y no funcionales, y con el diseño técnico).
* Aceptación (quiere demostrar que el sistema cumple las especificaciones funcionales y requisitos del cliente).

CALIDAD

La calidad es el grado de satisfacción del cliente. Implica cumplir con sus expectativas y satisfacerlas, sin excedernos en tiempo y presupuesto. La calidad es un valor en sí mismo y no un gasto que las empresas deben realizar para que su negocio prospere.

DIFERENCIA ENTRE VALIDACIÓN Y VERIFICACIÓN:

VALIDACIÓN:

Son aquellas cuestiones que se deben confirmar con el cliente para garantizar que se están entendiendo las necesidades, antes de pasar a la ejecución.

VERIFICACIÓN:

Es una salida contrastada con los requerimientos del lado del negocio, de modo de tener una comparación entre lo planeado y lo relevado, versus lo realmente logrado.

***7 principios del testing:***

* 1 - **La prueba muestra la presencia de defectos, no su ausencia** (nunca podemos decir que no hay defectos, hay que demostrar la presencia de defectos, no deberíamos enfocarnos en hacer una prueba y no querer ver defectos).
* 2 - **La prueba exhaustiva es imposible** ( no es posible probar absolutamente todo, es muy difícil hacer todas las combinaciones de datos, entonces hay que elegir, hay que saber que pruebas realizar y cuáles no, qué funcionalidades son las más críticas…etc. Hay que tratar de detectar lo más posible.)
* 3 - **La prueba temprana ahorra tiempo y dinero**
* 4 - **Los defectos se agrupan** (cuando estamos en una funcionalidad, y encontramos un defecto, en general hay más defectos en esa funcionalidad, si falla el carrito de compras, es normal que esté fallando algo más dentro de esa funcionalidad, entonces agrupamos estos defectos).
* 5 - **Cuidado con la paradoja del pesticida** (si hacemos muchas pruebas y siempre hacemos las mismas a lo largo del tiempo, no las actualizamos ni ampliamos, ya empiezan a ser inútiles, es posible que se necesite redactar nuevas pruebas).
* 6 - **La prueba depende del contexto** (cada prueba depende del proyecto y su contexto, al contexto, se lo llama por ejemplo, los dispositivos en los que se usa, para que funcionalidad, etc. )
* 7 - **La ausencia de errores es una falacia** (es mentira, nunca no hay errores, decir que es software completo no tiene errores es una falacia).

ASPECTO PSICOLÓGICO DEL TESTING

-El testing es el proceso de ejecución de un programa con la INTENCIÓN de encontrar errores.

PRUEBA INDEPENDIENTE:

**BENEFICIOS:**

* Es probable que los probadores independientes reconozcan diferentes tipos de fallos en comparación con los desarrolladores
* Un probador independiente puede verificar, cuestionar o refutar las suposiciones hechas por los implicados durante la especificación e implementación del sistema.

**DESVENTAJAS:**

* Los desarrolladores pueden perder el sentido de la responsabilidad con respecto a la calidad.
* Los probadores independientes pueden ser vistos como un cuello de botella o ser culpados por los retrasos en el lanzamiento o liberación.
* Los probadores independientes pueden carecer de información importante —por ejemplo, sobre el objeto de prueba.

MESA DE 3 PATAS

1. **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.
2. **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.
3. **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.

*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. Por eso, utilizamos la analogía con una mesa de 3 patas, pues si falta alguna de ellas, la mesa no podría estar de pie.*

GESTIÓN DE DEFECTOS

ERROR → Error del programador (equivocación de una persona)

DEFECTO → Defecto en el software

FALLA → En el sistema

*“El objetivo de los testers es brindar calidad dentro de un sistema de software”.*

**CICLO DE VIDA DE UN DEFECTO:**

* Nuevo/Inicial: Se recopila info y se registra el defecto
* Asignado: Si es un defecto válido y debe solucionarse se asigna al equipo de desarrollo, sino se puede rechazar o diferir (bug triage).
* Duplicado: Si el defecto se repite o existe otro con una misma causa raíz.
* Devuelto o rechazado: Se solicita más info o el receptor rechaza el defecto.
* Diferido: defecto no prioritario.
* En progreso: Análisis y se trabaja en la solución
* Corregido: Se cambia el código para solucionar el defecto
* En espera de verificación: El desarrollador espera que el tester lo pruebe.
* En verificación: El tester ejecuta una prueba de confirmación o reapertura del defecto.
* Verificado: Resultado de la prueba de verificación.
* Cerrado: defecto fue corregido y se encuentra disponible para el usuario final.

CONDICIONES A TENER EN CUENTA EN EL INFORME DE DEFECTOS:

1. El bug debe contener un ID.
2. Una falla debe ser reproducible para poder reportarla.
3. Ser específico
4. Reportar cada paso realizado para reproducirlo.

**PARTES DE UN INFORME DE DEFECTOS:**

| ID | Ej: 001-Test01 |
| --- | --- |
| Título : debe ser corto y específico | Ej: Login- Ingresa con campos en blanco |
| Descripción: Desarrollar… |  |
| Fecha del informe de defecto |  |
| Autor: Nombre del tester. |  |
| Identificación del elemento de prueba: Nombre de la aplicación o componente que estamos probando. | Ej: carrito de compras |
| Pasos a reproducir: Pasos para llegar al defecto |  |
| Resultado esperado: Lo que esperamos que suceda | Ej: No debería ingresar la app si hay campos en blanco. |
| Resultado actual: Lo que sucede realmente | Ej: Ingresa la app con campos en blanco. |
| Severidad: Cuán grave es el defecto.  Bloqueado/Crítico/Alto/Medio/Bajo/  Trivial | Ej: Crítico |
| Prioridad: Alta/Media/Baja | Ej: Alta |
| Estado del defecto: (Cualquiera de los pasos del ciclo de vida del defecto) |  |
| Referencias: Link al caso de prueba con el cual encontramos el error. |  |
| Imagen: Screenshot del error |  |

**CASO DE PRUEBA**

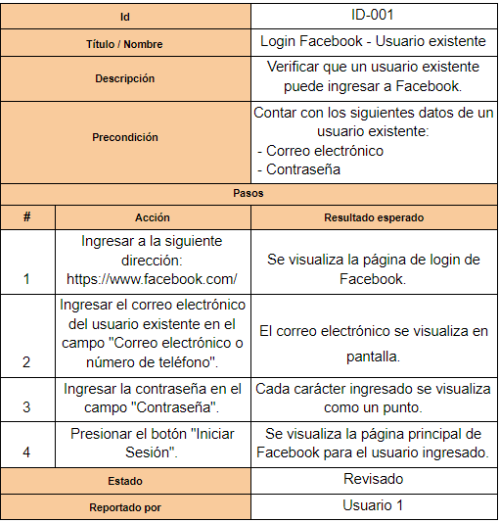
Conjunto de acciones que se ejecutan para verificar una característica o funcionalidad particular de una aplicación de software. Es decir, que todas las características de una aplicación de software van a ser representadas por uno o más casos de prueba.

CARACTERÍSTICAS DE UN BUEN CASO DE PRUEBA

* Ser simple: fácil comprensión
* Título fuerte
* Tener en cuenta el usuario final: debe cumplir los requisitos del cliente
* No asumir: no asumir funcionalidades ni características de la app
* Asegurar la mayor cobertura posible.
* Autonomía: debe generar los mismos resultados siempre, no importa quien lo pruebe.
* Evitar la repetición de casos de prueba

¿Qué debe contener?

* Identificador
* Nombre del caso de prueba (conciso)
* Descripción
* Precondición(que debe cumplirse antes de ejecutar el caso de prueba)
* Pasos(acciones a realizar para obtener los resultados)
* Resultados esperados(con los cuales se determina si el test falló o no).



**TESTING POSITIVO Y TESTING NEGATIVO**

TESTING POSITIVO

Son aquellos casos de prueba que validan el flujo normal de un sistema bajo prueba. Es decir, flujos que están relacionados a los requisitos funcionales del sistema bajo prueba.

TESTING NEGATIVO

Son aquellos casos de prueba que validan flujos no contemplados dentro de los requisitos de un sistema bajo prueba.

HAPPY PATH TESTING

Se prueba el flujo que realiza el usuario final cuando usa la aplicación de manera regular. Es la primera prueba que se realiza en la app y se incluye en la categoría de prueba positiva. Su propósito no es encontrar defectos, sino ver un producto o procedimiento que funcione como ha sido diseñado. Ayuda a identificar cualquier problema en una etapa temprana.

***CASO DE USO***

**Cuenta la historia de cómo un usuario interactúa con el software para lograr o abandonar un objetivo. Todos los caminos usados por el usuario para interactuar se llaman escenarios de caso de uso.**

***CASO DE PRUEBA***

**Cubre el software en profundidad y con mayor detalle que un caso de uso. Incluyen todas las funciones que el programa puede realizar, se deben tener en cuenta los datos de entrada/salida, cada comportamiento esperado y todos los elementos de diseño.**

**¿Cómo combinamos casos de uso y casos de prueba?**

Se puede empezar escribiendo casos de prueba para escenario principal y luego para escenarios alternativos. Se escribe uno o más casos de prueba para cada escenario de caso de uso.

*Las actividades que se desarrollan a lo largo del ciclo de vida de las pruebas de software (STLC), se las organiza y se las gestiona en distintos NIVELES DE PRUEBA.*

*Luego, las actividades de prueba van a ser agrupadas de acuerdo a características específicas que se necesitan probar en el software, estos grupos de pruebas con un objetivo específico son llamados TIPOS DE PRUEBA.*

El ciclo de vida de las pruebas de software consiste en las siguientes actividades principales:

1. Planificación
2. Seguimiento y control
3. Análisis
4. Diseño
5. Implementación
6. Ejecución
7. Conclusión
8. Planificación:

Se definen los objetivos y enfoque de la prueba.

1. Seguimiento y control

Se reúne información y se proporciona feedback y visibilidad sobre las actividades de prueba. Se pueden tomar acciones correctivas, ej, cambiar la prioridad de las pruebas, el calendario y reevaluar los criterios de entrada y salida.

1. Análisis

Durante esta prueba se determina **“qué probar”**.

Ej: se trata de identificar defectos de distintos tipos en las bases de prueba. Identificar los requisitos que se van a probar y definir condiciones de prueba para cada requisito.

1. Diseño

Durante esta prueba se determina **“cómo probar”**.

Ej: se diseña el entorno de prueba y se identifica la infraestructura y herramientas necesarias.

1. Implementación

Se completan los productos de prueba necesarios para la ejecución de la prueba, incluyendo la secuenciación de los casos de prueba en procedimientos de prueba.

Ej: se desarrolla y prioriza procedimientos de prueba. Se crean juegos de prueba.

1. Ejecución

Se ejecutan los casos de prueba.

Se comparan los resultados reales con los resultados esperados.

1. Conclusión

Se recopilan la información de las actividades completadas y los productos de prueba.

Se comprueba que todos los informes de defecto están cerrados. Se finaliza, archiva y almacena el entorno de prueba, la infraestructura y otros productos de prueba.

NIVELES DE PRUEBA

1. PRUEBA UNITARIA O DE COMPONENTE

En general, el desarrollador que escribió el código realiza la prueba de componentes. 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 componentes automatizados puede preceder a la redacción del código de la aplicación.

PRUEBA DE INTEGRACIÓN

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.

PRUEBA DE SISTEMA

La prueba de sistema debe concentrarse 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.

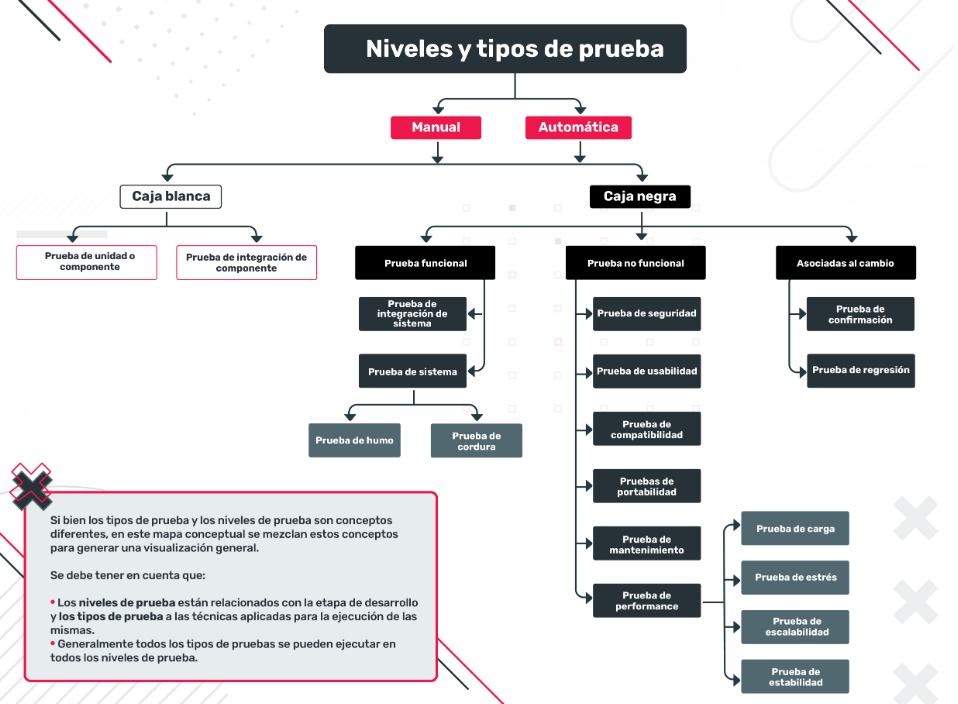
PRUEBA DE ACEPTACIÓN

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

TIPOS DE PRUEBA

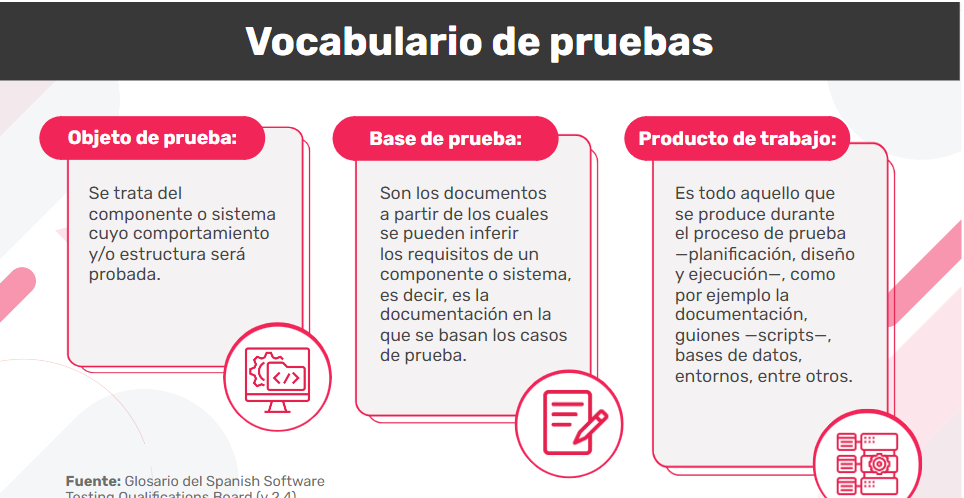
Un tipo de prueba es un grupo de actividades de pruebas destinadas a probar las características específicas de un sistema de software, o de una parte de un sistema, basados en objetivos de pruebas específicas.

|  | **1. Prueba Funcional** | **2. Prueba No Funcional** | **3. Prueba Estructurales** | **4. Prueba Asociada al Cambio** |
| --- | --- | --- | --- | --- |
| **Definición** | Pruebas que evalúan las funciones que el sistema debe realizar. Las funciones describen **qué hace** el sistema. | La prueba no funcional prueba “**cómo de bien**” se comporta el sistema. | Basadas en la estructura interna del sistema o en su implementación. La estructura interna puede incluir código, arquitectura, flujos de trabajo y/o flujos de datos dentro del sistema | Existen 2 tipos de prueba relacionadas al cambio:   * **Prueba de confirmación**: Una vez 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. * **Prueba de regresión:** La prueba de regresión implica la realización de pruebas para detectar efectos secundarios no deseados a raíz de algún cambio realizado previamente. |
| **Implementación** | La prueba funcional observa el comportamiento del software. | El diseño y ejecución de la prueba no funcional puede implicar competencias y conocimientos especiales, como el conocimiento de las debilidades inherentes a un diseño o tecnología, ej programación. | Este tipo de pruebas pueden implicar competencias o conocimientos especiales, como código. | Especialmente en los ciclos de vida de desarrollo iterativos e incrementales (por ejemplo, Agile), las nuevas características, los cambios en las 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** | Se pueden realizar en todos los niveles de prueba. | Se pueden realizar en todos los niveles de prueba | Se puede realizar en el nivel de componente y de integración. | Se realizan en todos los niveles de prueba |
| **Cobertura** | La cobertura funcional 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. | La cobertura no funcional 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. | La cobertura estructural 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. | 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 |



FORMAS DE PRUEBA :

* **MANUALES**: Son ejecutadas directamente por uno o más testers, simulando las acciones del usuario final, apoyándose de las herramientas necesarias.
* **AUTOMATIZADAS**: Son ejecutadas por testers con habilidades técnicas, y se apoyan en diversas herramientas para realizar scripts y así, ejecutar las pruebas automáticamente.



TÉCNICAS DE PRUEBA

El objetivo de una técnica de prueba es ayudar a identificar las condiciones, los casos y los datos de prueba.

1. TÉCNICAS DE CAJA NEGRA:

Se basan en el comportamiento extraído del análisis de los documentos que son base de prueba(docs de requisitos, casos de uso, historias de usuario). Son aplicables tanto para pruebas funcionales como no funcionales. Se concentran en las entradas y salidas sin tener en cuenta la estructura interna.

1. TÉCNICAS DE CAJA BLANCA:

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.

1. TÉCNICAS BASADAS EN LA EXPERIENCIA:

Aprovechan el conocimiento de desarrolladores, probadores, y usuarios para diseñar, implementar y ejecutar las pruebas.



TÉCNICAS DE CAJA NEGRA

PARTICIÓN DE EQUIVALENCIA

→ Se dividen los datos en particiones conocidas como **clases de equivalencia,** donde cada miembro de estas clases o particiones es procesado de la misma manera,

* La partición de equivalencia válida contiene valores que son aceptados por el componente o sistema.
* La partición de equivalencia no válida contiene valores que son rechazados por el componente o sistema.
* Se pueden dividir las particiones en subparticiones
* Cada valor pertenece a solo una partición de equivalencia



ANÁLISIS DE VALORES LÍMITES

* Consiste en datos numéricos o secuenciales.
* Se deben identificar los valores límites mínimo y máximo (o valores inicial y final).
* Se pueden usar 2 o 3 valores límites.



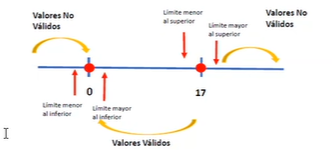


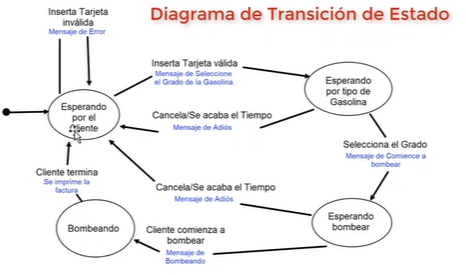
TABLA DE DECISIÓN

* Se usa para pruebas combinatorias formadas por reglas de negocio complejas que un sistema debe implementar.
* Las condiciones (entradas) y las acciones resultantes (salidas). Estas conforman las filas de la tabla.
* Las columnas de la tabla corresponden a reglas de decisión. Cada columna forma una combinación única de condiciones y la ejecución de acciones asociadas a esa regla.
* Los valores de la tabla pueden ser booleanos, discretos, numéricos, etc.



TRANSICIÓN DE ESTADOS

* Muestra los posibles estados del software, así como la forma en la que el software entra, sale y realiza las transiciones entre estados.
* Muestra las transiciones válidas y las transiciones potencialmente inválidas entre estados, se excluyen las transiciones NO válidas.
* Esta prueba se usa para apps basadas en menús y se usa en la industria de software embebido.





TÉCNICAS BASADAS EN LA EXPERIENCIA

PREDICCIÓN DE ERRORES

Esta técnica se usa para anticipar la ocurrencia de equivocaciones, defectos y fallos basados en el conocimiento del probador.

Se tiene en cuenta:

→ Cómo ha funcionado la app en el pasado

→ Errores comunes en los desarrolladores

→ Fallos en apps relacionadas

PRUEBA EXPLORATORIA

En esta técnica se diseñan, ejecutan, registran y evalúan de forma dinámica pruebas informales durante la ejecución de la prueba.

Estas pruebas sirven para aprender sobre el funcionamiento del componente o sistema.

Se usan para complementar técnicas formales.

PRUEBA BASADA EN LISTAS DE COMPROBACIÓN

En esta técnica se diseñan, implementan y ejecutan casos de prueba que cubren las condiciones que se encuentran en una lista de comprobación definida.

Se utiliza tanto en pruebas funcionales como no funcionales.

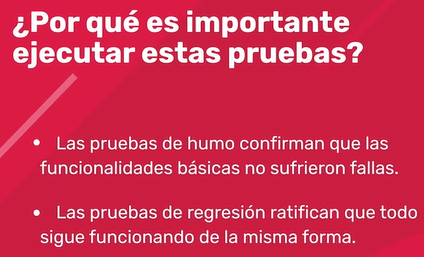
Introducción a la implementación y ejecución de la prueba

Una de las actividades es completar los productos de trabajo necesarios para la ejecución de la prueba, preparar el entorno de prueba —incluídos datos o configuraciones—, organizar las suites de pruebas en un calendario de ejecución, entre otros.

**EJECUCIÓN DE LA PRUEBA**

1. Registrar los identificadores y las versiones de los elementos u objetos de prueba, las herramientas de prueba y los productos de prueba.
2. Ejecutar pruebas de forma manual o utilizando herramientas de ejecución de pruebas.
3. Comparar resultados reales con resultados esperados.
4. Analizar las anomalías para establecer sus causas probables.
5. Informar sobre los defectos en función de los fallos observados.
6. Registrar el resultado de la ejecución de la prueba.
7. Repetir las actividades de prueba, ya sea como resultado de una acción tomada para una anomalía o como parte de la prueba planificada.

PRUEBAS DE HUMO Y PRUEBAS DE REGRESIÓN

Las **pruebas de humo(smoke test)** se ejecutan para evaluar la estabilidad de las compilaciones de software iniciales o desarrolladas recientemente. Ej: se agregan videollamadas grupales a WPP.

Las **pruebas de regresión** tienen la tarea de verificar y validar las funcionalidades existentes de la aplicación después de cada modificación o en la adición de nuevas funciones.

*Las pruebas de humo son previas a las de regresión.*

.

PRUEBAS ESTÁTICAS Y DINÁMICAS

Tienen el objetivo de proporcionar una **evaluación de calidad** de los productos de trabajo e **identificar defectos** en forma temprana.

**PRUEBAS ESTÁTICAS**

Se basa en revisiones de los productos de trabajo o en la evaluación basada en herramientas del código u otros productos de trabajo. Este tipo de pruebas no requieren de la ejecución del software que se está probando. Las pruebas estáticas permiten la detección TEMPRANA de defectos y por ende se reducen los costos.

Ejemplos de productos de trabajo en los que se usan pruebas estáticas:

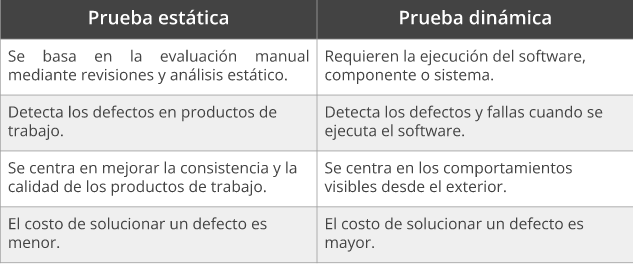
* Historias de usuarios
* Código
* Manuales de usuarios
* Planes de proyecto

**PRUEBAS DINÁMICAS**

Estas requieren la ejecución del software, componente o sistema. Se complementan con las estáticas ya que encuentran diferentes defectos. Durante estas pruebas se EJECUTA el software y se analiza y compara con el resultado esperado.

Fallas + comunes de pruebas dinámicas:

* Fallas de funcionalidad
* Fallas de interacción entre módulos
* Fallas de rendimiento y seguridad



PROCESOS DE REVISIÓN:

Dentro de las pruebas estáticas, una forma de detectar errores es mediante un **proceso de revisión.**

Las revisiones consisten en examinar cuidadosamente un producto de trabajo con el principal objetivo de encontrar y remover errores. Pueden ser realizadas por una o más personas.

Las revisiones pueden ser:

* **Revisiones formales:** Tienen roles definidos, siguen un proceso establecido y deben ser documentadas.
* **Revisiones Informales**: No siguen un proceso definido y no son documentadas formalmente.

El grado de formalidad del proceso de revisión está relacionado con factores, como el modelo del ciclo de vida del desarrollo del software, la madurez del proceso de desarrollo, la complejidad del producto del trabajo que se debe revisar, etc.

REVISIONES FORMALES- COMPONENTES:

**Roles** → Autor, Dirección,Facilitador, Líder de revisión, Revisores, Escriba.

**Tipos** → Guiada/Walkthrough : Dirigida por el autor del producto de trabajo, el escriba es obligatorio, Técnica: Se realiza entre pares técnicos del autor.Tienen que existir un facilitador y un escriba obligatoriamente, quien idealmente no será el autor. Se elaboran registros de defectos potenciales e informes de revisión, Inspecciones.

**Técnicas** → Ad hoc: Los revisores documentan los defectos, Basada en escenarios y ensayos: Fundada en el uso esperado del producto de trabajo descrito en un documento, por ejemplo, en un caso de uso, Basada en listas de comprobación: Los revisores detectan defectos a partir de un conjunto de preguntas basadas en defectos potenciales, Basada en Roles: Los revisores evalúan el producto de trabajo desde la perspectiva de usuarios experimentados, inexpertos, adultos, niños o roles específicos en la organización, Basada en Perspectiva: los revisores adoptan los diferentes puntos de vista del usuario final, del personal de marketing, del diseñador, del probador o del personal de operaciones.

**Actividades** → Planificar: Definir el alcance, establecer objetivos, roles, tiempo y plazos. Definir y comprobar el cumplimiento de criterios de entrada y salida para revisiones más formales. Iniciar revisión: Distribuir material e instruir a los participantes. Revisión individual: revisión del material y se toma nota de los defectos. Comunicar y analizar: comunicar defectos a los responsables. Corregir e informar: comunicar los defectos encontrados y elaborar informes de hallazgos.

Requisitos

Una de las revisiones que se realizan en las pruebas estáticas es examinar los requisitos del software. Pero ¿sabemos qué son los requisitos? ¿Qué tipos de requisitos existen?

Un requisito define las funciones, capacidades o atributos intrínsecos de un sistema de software, es decir, describe cómo debe comportarse un sistema. Para decir que un sistema tiene calidad deben cumplirse los requisitos funcionales y no funcionales.

**Requisitos funcionales**

Definen lo que un sistema permite hacer desde el punto de vista del usuario. Estos requisitos deben estar especificados de manera explícita. Por ejemplo: “El campo de monto acepta únicamente valores numéricos con dos decimales” (pruebas funcionales y de sistema).

**Requisitos no funcionales**

Definen condiciones de funcionamiento del sistema en el ambiente operacional. Ejemplos:

* **Requisito de usabilidad:** la usabilidad se define como el esfuerzo que necesita hacer un usuario para aprender, usar, ingresar datos e interpretar los resultados obtenidos de un software de aplicación (pruebas de usabilidad).
* **Requisito de eficiencia:** relacionado con el desempeño en cuanto al tiempo de respuesta, número de operaciones por segundo, entre otras mediciones; así como consumo de recursos de memoria, procesador y espacio en disco o red (pruebas de rendimiento, pruebas de carga, estrés y escalabilidad, pruebas de gestión de la memoria, compatibilidad e interoperabilidad).
* **Requisito de disponibilidad:** disposición del sistema para prestar un servicio correctamente (pruebas de disponibilidad). Requisito de confiabilidad: continuidad del servicio prestado por el sistema (pruebas de seguridad).
* **Requisito de integridad:** ausencia de alteraciones inadecuadas al sistema (pruebas de seguridad, pruebas de integridad). Requisito de mantenibilidad: posibilidad de realizar modificaciones o reparaciones a un proceso sin afectar la continuidad del servicio (pruebas de mantenimiento y de regresión).

ORGANIZACIÓN DE LA PRUEBA

Deben existir diferentes ambientes de trabajo, donde se pueda desarrollar y probar los cambios antes de que llegue al ambiente del cliente.

Entendemos como ambiente de trabajo al entorno con todos los recursos necesarios para que se pueda ejecutar un sistema. Aprenderemos cómo se organizan las diferentes pruebas a lo largo de los ambientes, entendiendo qué pruebas utilizar y dónde deben ejecutarse.

AMBIENTES DE TRABAJO

Los ambientes de trabajo hacen referencia a un servidor con ciertos recursos asignados, software y librerías instalados, su propia base de datos y una configuración determinada.

Esto nos permitiría 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 uno de los ambientes son utilizados con un fin específico y presentan ciertas ventajas sobre los otros en determinado momento del proceso de trabajo.

NIVELES DE AMBIENTES:

**AMBIENTE DE DESARROLLO O DEV:** el programador desarrolla el código de la aplicación, realiza pruebas iniciales y comprueba si la aplicación se ejecuta correctamente con ese código.

TIPOS DE PRUEBA:

* **Pruebas unitarias o de componente:** se prueban componentes x separado, busca encontrar defectos en el componente.
* **Pruebas de integración:** se centra en las interacciones entre componentes o sistemas. Buscan encontrar defectos en las interfaces.

**AMBIENTE DE PRUEBAS O QA:** Permite minimizar incidencias en etapas posteriores, ya que el tester ejecutaría las primeras pruebas de funcionalidad en este ambiente.

TIPOS DE PRUEBA:

* **Pruebas funcionales:** se prueban funciones que el sistema debe realizar.
* **Pruebas de casos de uso:** proporcionan pruebas transaccionales, basadas en escenarios, que deberían emular el uso de sistema.
* **Pruebas de exactitud:** comprenden el cumplimiento por parte de la app, los requisitos especificados o implícitos.
* **Pruebas de adecuación:** implican evaluar y validar la eficiencia de un conjunto de funciones para la consecución de las tareas especificadas previstas.
* **Pruebas de sistema:** se centra en el comportamiento y las capacidades de todo un sistema o producto de extremo a extremo.
* **Pruebas de regresión:** pruebas para detectar efectos secundarios no deseados luego de cambios en el código u actualizaciones que puedan afectar al resto de las partes del código.
* **Pruebas de confirmación:** se vuelve a ejecutar los pasos para confirmar que el defecto fue solucionado.
* **Pruebas de cordura:** similar a la de regresión pero se enfoca en unas pocas áreas de funcionalidad.
* **Pruebas de humo:** es un smoke test para asegurar que las funciones MÁS importantes del programa están funcionando, no se ocupa de los detalles finos.

**AMBIENTE DE UAT** (pruebas de aceptación del 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.

TIPOS DE PRUEBA:

* **Pruebas de aceptación:** producen info para evaluar el grado de preparación del sistema para su despliegue y uso por parte del cliente.
* **Pruebas exploratorias:** se usan para entender más sobre el componente o sistema y crear pruebas para las áreas que pueden necesitar ser probadas con mayor intensidad.
* **Pruebas de usabilidad:** evalúan la facilidad con la que los usuarios puedan acceder y utilizar el sistema.
* **Pruebas de accesibilidad:** evalúan la accesibilidad que presenta un software para aquellos con necesidades de uso(discapacitados).

**AMBIENTE DE PREPRODUCCIÓN O STAGE:** 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.

TIPOS DE PRUEBA:

* **Pruebas de mantenimiento:** se prueban los cambios en el sistema.
* **Pruebas de seguridad:** se usan para encontrar fallas en el sistema y se busca disminuir el impacto de ataques y pérdida de información importante.
* **Pruebas de rendimiento:** evalúan los perfiles de tiempo, flujo de ejecución, los tiempos de respuesta y la fiabilidad y los límites operativos.
* **Pruebas de carga, estrés y escalabilidad:** la prueba de carga garantiza que el sistema puede controlar el volumen de tráfico esperado, la prueba de estrés se somete al sistema a condiciones de uso extremas para garantizar su robustez y confiabilidad. Las pruebas de escalabilidad, prueba que el sistema pueda soportar el incremento de demanda en la operación.
* **Pruebas de infraestructura:** incluyen todos los sistemas informáticos internos, los dispositivos externos asociados, las redes de internet, la nube y las pruebas de virtualización.
* **Pruebas de gestión de memoria:** evalúan el estado y la integridad de la memoria del sistema para identificar problemas potenciales.
* **Pruebas de compatibilidad:** comprueba que el sistema es compatible con todos los navegadores de Internet y todos los sistemas operativos del mercado.
* **Pruebas de interoperabilidad:** evalúan la correcta integración entre distintos aplicativos, sistemas, servicios o procesos que conforman una plataforma o solución tecnológica.
* **Pruebas de migración de datos:** incluyen las pruebas realizadas al transferir datos entre tipos de dispositivos de almacenamiento, formatos o sistemas de cómputo.

**AMBIENTE DE PRODUCCIÓN O PROD:** entorno donde finalmente se ejecuta la aplicación, donde acceden los usuarios finales y donde se trabaja con los datos reales de negocio.

NO ANALIZAMOS LAS PRUEBAS DE ESTE AMBIENTE