- Introducción a la unidad 1

En la primera unidad del curso de Tester de Software, exploraremos los conceptos fundamentales asociados al testing de software. Comprenderemos la evolución histórica del testing de software, desde sus inicios en las primeras décadas de la informática hasta su importancia actual en el mundo tecnológico.

Analizaremos la relevancia del testing en el desarrollo de software, su integración en el ciclo de desarrollo, y su funcionalidad para garantizar la calidad y confiabilidad de las aplicaciones y sistemas informáticos.

El estudio del testing de software es esencial para desarrollar aplicaciones robustas y confiables. Los testers desempeñan un papel crucial al identificar, corregir errores y garantizar la satisfacción del cliente. Un testing efectivo ayuda a reducir costos relacionados con correcciones tardías y retrasos en el lanzamiento de productos. Además, el testing de seguridad protege la integridad de datos y la información confidencial, especialmente en sectores regulados como la salud y las finanzas.

En esta unidad, profundizaremos en temas como la definición y alcance del testing de software, tipos de pruebas, metodologías de testing y herramientas fundamentales para pruebas efectivas. A medida que avancemos en el curso, adquirirás habilidades y conocimientos avanzados para destacar en un entorno tecnológico en constante evolución.

- Historia del Testing de Software

La historia del testing de software es fascinante y ha evolucionado a lo largo de décadas, convirtiéndose en un pilar fundamental en el mundo de la tecnología. Desde las pruebas manuales en el ENIAC en 1947 hasta la incorporación de la inteligencia artificial en las pruebas de software en 2020, hemos presenciado una serie de hitos clave que han dado forma a esta disciplina.

En 1947, el ENIAC experimentó pruebas de errores manuales, donde los operadores examinaban los cables y conexiones para identificar problemas. En 1956, Grace Hopper acuñó el término "debugging" al encontrar una polilla atascada en un relé de un computador, marcando el comienzo de la depuración de errores en el software.

En la década de 1960, se desarrollaron las primeras herramientas de pruebas automáticas, como el IBM Testar y el Test-Maker de Honeywell. En los años 70 surgió el término "Quality Assurance" en el campo de desarrollo de software, estableciendo procesos para garantizar la calidad del software.

En la década de 1980, el IEEE publicó el estándar 829 para la documentación de pruebas de software, proporcionando directrices sólidas. En los 90, las metodologías ágiles como Scrum y Extreme Programming popularizaron la integración de pruebas en el ciclo de desarrollo, centrando en la calidad desde el principio.

Para el año 2000, las pruebas de software en la nube se volvieron más accesibles y escalables con la computación en la nube. En 2010, hubo un enfoque creciente en la automatización de pruebas, destacando el uso de marcos como Selenium.

Finalmente, la década de 2020 ha visto la incorporación de inteligencia artificial y aprendizaje automático en el testing de software, permitiendo la detección automática de errores y la generación de casos de prueba. A medida que la tecnología avanza, el campo de las pruebas de software sigue evolucionando para garantizar la calidad y la confiabilidad de los sistemas modernos.

-Testing de Software

El testing de software es una disciplina fundamental para identificar posibles defectos y problemas en el software y garantizar su correcto funcionamiento. Es crucial para asegurar la calidad del software en la era digital actual, ya que los software defectuosos pueden tener consecuencias graves. Se trata de un conjunto de procesos, métodos de trabajo y herramientas diseñados meticulosamente para abordar una pregunta tan profunda como relevante: ¿Está el software que sustenta nuestra vida digital libre de defectos y es capaz de funcionar sin problemas?
La respuesta a esta interrogante reside en el corazón mismo del testing de software.

Su objetivo principal es garantizar que el software funcione de manera estable, cumpla con sus especificaciones y satisfaga las necesidades de los usuarios finales.
Los profesionales del testing Su misión es simple pero esencial: **asegurar que el software que llega a las manos de los usuarios sea confiable, seguro y cumpla con las expectativas.**

Algunos beneficios clave del testing de software incluyen la identificación y corrección de defectos antes de llegar a los usuarios finales, la mejora continua de la calidad del software, el cumplimiento de requisitos, la generación de confianza en el producto final y el ahorro de costos asociados con la corrección de errores.

Los testers de software juegan un papel crucial en la creación de aplicaciones y sistemas informáticos que cumplen con las expectativas de los usuarios. **En resumen, el testing de software es esencial para garantizar la calidad, estabilidad y confiabilidad de los software que utilizamos a diario, siendo la base para el desarrollo exitoso de software de alta calidad.**

- **Conociendo Las Pruebas -** Video 02

Una vez finalizado un proyecto, se sigue trabajando en las pruebas del sistema , luego en la implantación del sistema y el mantenimiento del sistema.

- Pruebas del sistema: Evalúa que hace el software, es decir que se ejecuta correctamente y de manera eficiente
- Implantación del sistema: Deja la aplicación en funcionamiento en las maquinas del cliente
- Mto de sistema: Consiste en una series de pruebas para verificar su funcionamiento una vez implementado el software en el cliente.

¿Qué son las pruebas de software?

Son pasos (pruebas de menor a mayor), realizamos pruebas de lógicas y datos encapsulados en los componentes. Se realizan pruebas de orden superior y llevamos un proceso de depuración para diagnosticar posibles errores.

Las pruebas de software es la definición del producto final y para esto se realiza una especificación de pruebas que permite valorar si se incluyen todos las casos de pruebas y el plan de pruebas debe estar orientado de forma ordenada la construcción del software y la detección de errores en cada etapa

Para realizar la pruebas hay que realizar una estrategia de prueba que define los pasos de las pruebas a realizar para esto se necesita una planificación de las pruebas (QUE, CUANDO, COMO, QUE RECURSOS, CUANTO DURA, etc=).

Necesario los diseños de casos de prueba, la recolección y evaluación de los resultados.

¿Para que sirven las pruebas? Para detectar posibles errores y corregirlos. Las pruebas pueden ser flexibles o rígidas según el proceso que utilizemos.

- **El proceso del plan de pruebas:** para llevarlo a cabo se debe

☑ Generar un plan de pruebas
☑ Diseño específicos de pruebas
☑ Ejecutar pruebas
☑ Analizar y evaluar la salida esperada, acá detectamos si es la salida deseada o si debemos pasar a una depuración de errores
☑ Analizar si la falla del software tiene un defecto o si tuvimos un error al crear el plan de pruebas
☑ Realizar una estadísticas de los errores.

-Estrategias de las pruebas

Según el momento de realización:

- ***prueba de unidad****:* verifica los pequeños módulos del software
- ***Prueba de integración****: verifica la integración de todos los componentes que ya se han ido probando.*
- ***Pruebas de validación****:* Se centran en realizar acciones, de lo que el usuario final ve.

Según la forma que se realizan:

- ***Pruebas de caja blanca***: Se centran en la estructura interna del programa, es decir se generan casos de prueba por caminos diferentes independientemente de cada módulo y probando las decisiones lógicas del lado verdadero y falso. Y ejecutando los bucles en sus límites operacionales. Por último se prueban las estructuras internas para asegurar su validez.
- ***Pruebas de caja negra:*** Contrastan los requisitos del software como deben de ser las funciones y cómo los diferentes módulos han sido implementados , sus entradas y salidas.
- ***Análisis de valores límites:*** Exploran las condiciones límites del software, es decir cuando se prueban con los valores extremos permitidos en el software.

Testing de Software: Un fundamento Clave

El testing de software es una disciplina fundamental en la ingeniería de software que emplea procesos, metodologías y herramientas para identificar posibles defectos en el software y garantizar su correcto funcionamiento. Este proceso es crucial para la creación de software de calidad, ya que permite la detección y corrección de errores antes de que el software llegue a manos de los usuarios, asegurando así su estabilidad y satisfacción del cliente.

Las pruebas de software desempeñan un papel importante en la garantía de calidad, el ahorro de tiempo y recursos, la optimización del rendimiento y el cumplimiento de requisitos. Es fundamental entender que las pruebas no deben ser realizadas al final del desarrollo, sino que deben llevarse a cabo en paralelo para evitar problemas y corregir posibles fallos a tiempo.

Las pruebas de software surgieron en la década de 1960 en respuesta a la crisis en el desarrollo de software. Desde entonces, se han convertido en una parte integral del proceso de desarrollo de software, siendo esencial para la entrega de un trabajo bien realizado con la menor cantidad de problemas posibles. Es importante recordar que las pruebas deben llevarse a cabo en todas las etapas del desarrollo y que cada miembro del equipo desempeña diferentes roles en estas actividades.

En resumen, el testing de software es un proceso clave en el desarrollo de software que ha evolucionado con el tiempo y seguirá evolucionando a medida que avance la tecnología. Comprender este proceso es esencial para el curso que sigue, ya que nos permitirá explorar en profundidad los conceptos y prácticas que lo rodean en las próximas clases.

- **Conceptos generales del manejo de pruebas de softwares Video 03**

Términos:

- **Error**: Cuando hay una equivocación del desarrollador, ej=
    - Error en una expresión lógica
    - Error al retornar una excepción
    - Error a redireccionar a una pagina web
    - etc
- **Defecto**: Diferencia entre el valor esperado y el obtenido
- **Falla**: Problema presentado después de desplegado el software
- BUG: Una inconsistencia del software encontrada en etapa de pruebas.

Objetivos al realizar pruebas:

- ☑ Encontrar defectos
- ☑ Proporcionar información sobre la calidad (buena o mala) del software, que este se comporte como el usuario quiere y que no genere fallas.
- ☑ Prevenir defectos
- ☑ Asegurar que el resultado cumple con los requerimientos
- ☑ Aumentar la confianza en la calidad del producto

Niveles de las pruebas: Del más bajo al más alto.

- **Unidad:**
  - Se prueban los componentes individuales o módulos (funciones, métodos, clases, etc).
  - Detección de errores lógicos, algoritmos o manejos de datos realizados por el desarrollador.
- **Integración**:
  - Las unidades se integran a este nivel y se prueban en grupos con componentes
  - Se prueban como interaccionan entre si y son hechas por el desarrollador y las prueba el tester.
- **Sistema:**
  - Probar el sistema como un todo, este nivel no es tán técnico
  - Errores a nivel de requisitos (funcionales y no funcionales)
  - Las pruebas la realizan el analista y tester.
- **Aceptación:**
  - Prueba que el sistema esta listo para su entrega, son similares a las del sistema
  - Las pruebas las realizan el analista o tester y son denominadas pruebas “Alfa”
  - Las pruebas Betas las realizan usuarios o clientes seleccionados.

- **Estrategias de pruebas:**

Caja Negra: Realizamos pruebas fuera del software, es decir una evaluación de datos de entrada y compararlos con los de salida.

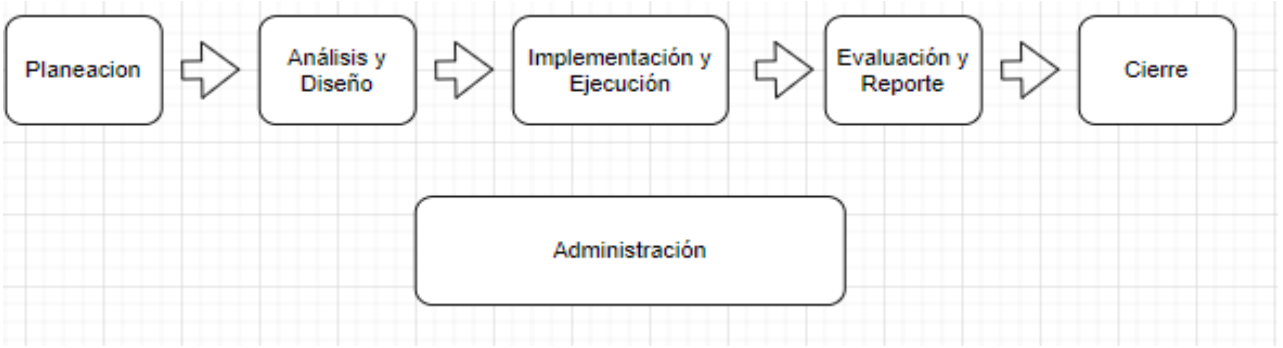
- Enfoque exhaustivo
- No importa la estructura interna
- Los datos de entrada son derivados de las especificaciones de requisitos
- No puede revelar errores debidos a codificación

Caja Blanca: Realizan pruebas internas dentro del software.

- Examinar la estructura interna del programa
- Deriva los datos de entrada de la lógica del software, se mira en detalle cómo se ejecutan las instrucciones.
- Asegurar que cada instrucción es ejecutada al menos una vez
- Se enfoca más en el código

Cuando uno va a diseñar las pruebas lo que se debe hacer es un conjunto de **casos de prueba:** Consiste en un grupo de datos de entrada que derivan de requisitos o la lógica del algoritmo, se obtendrán los datos de salida que evaluaran si causan fallas en el sistema y ayudan a detectar errores que hay que corregir para evitar fallas y defectos.

Proceso de pruebas :



Planeación: Que componentes vamos a probar, en que fechas, registros de como se comportan las pruebas
Análisis y diseño: Analizar pruebas, aca el rol tiene que ser muy creativo ya que debemos tratar de que falle el software.
Implementación y Ejecución de las pruebas
Evaluación y reporte
Cierre

Administración: es una actividad que se encarga durante todo el proceso de administrar las pruebas.

Conocimientos necesarios para empezar como Tester **Video 04**

Organizado, detallista, intuitivo, capacidad de comunicación, persistente, Estos elementos son importantes.

Actividades que el tester está involucrados

- Planificación de las pruebas
- Monitoreo y control de pruebas
- Análisis
- Diseño
- Implementación
- Ejecución
- Finalización del proceso de pruebas

Hay varios factores que influyen en el proceso de pruebas de un proyecto y uno de ellos es la metodología que se utilice EJ:
Metodologías Ágiles : Estas actividades se van a ejecutar juntas de forma simultánea, combinando entre ellas. No siguen un patrón secuencial (es decir, no hay 1 fase primero y luego la que sigue), utilizan un enfoque iterativo, es decir una pequeña iteración y dentro de esta tiene varias actividades que serán ejecutadas y después pasará a la siguiente iteración donde hay otras actividades y así sucesivamente hasta el final del proyecto.

- Planificación de las pruebas : Elementos que se definen
    - Objetivos de las pruebas
    - Técnicas de pruebas a utilizar
    - Tipos de pruebas
    - Niveles de pruebas adecuado para cada proyecto
    - Calendario de pruebas con fechas límites. (En metodologías ágiles las fechas pueden estar definidas por los sprints
    - Riesgos de pruebas
    - Criterios de entrada y salida (cuando comenzamos las pruebas y cuando terminamos)
    - Las métricas que vamos a medir en el proceso de prueba

★ **Estos elementos se describen en el plan de pruebas**

- **Monitoreo de las pruebas :** Control de lo que se planificó, implica la comparación del progreso real con el plan de pruebas, utilizando las métricas que se definieron en la planificación de las pruebas

Documento que se crea a partir del monitoreo de las pruebas es : el informe de progreso de las pruebas, donde vamos a reportar las desviaciones respecto al plan y las medidas que se fueron tomando respecto a las desviaciones (DE MANERA GENERAL).

- **Análisis de las pruebas :** Se analizan las bases de pruebas buscando omisiones, inconsistencias, ambigüedades.

  ¿Qué son las bases de prueba ? Aquellos documentos o productos de trabajo a partir de los cuales se pueden inferir los requisitos de un componente o del sistema. EJ = Requerimientos, historias de usuarios, diagramas de diseño, modelos de datos, o otros similares que especifique cómo debe ser el comportamiento del sistema que estamos probando.

Los análisis de prueba se refieren a la pregunta de ¿QUÉ PROBAR?.

- **Diseño de prueba: Responde a la pregunta ¿Cómo realizamos las pruebas ?**
  - Se incluyen actividades como:
  - Diseñar y priorizar casos de pruebas
  - IDentificar los datos de las pruebas (Datos que vamos a necesitar para ejecutar cada uno de los casos de prueba que habremos diseñado previamente.)
  - Diseñar el entorno de pruebas (Identificar la estructura y herramientas necesarias)
  - Verificar y actualizar la trazabilidad bidireccional entre los elementos de las pruebas.

- **Implementación de las pruebas : Actividades relacionadas con el entorno de las pruebas, muchos proyectos se realizan en la fase del diseño esta implementación de las pruebas.**

- **Ejecución de las pruebas : Hay tareas como**
  - **Ejecutar pruebas (manual o automatizadas)**
  - **Comparar resultados reales contra los esperados**
  - **Reportar los defectos encontrados**

    - **Registrar el resultado de la ejecución de prueba (si se aprobo o desaprobó).**
    - **Repetir actividades de pruebas (pruebas de confirmación o regresión).**

- **Finalización del proceso: ¿Qué tareas hay ?**
    - **Verificar si todos los informes de defectos están cerrados**
    - **Crear un informe de resumen de las pruebas**
    - **Finalizar y archivar el entorno de prueba, datos e infraestructura que utilizamos en el proceso**
    - **Entregas el software de pruebas a equipos de mantenimiento o partes interesadas**
    - **Analizar las lecciones aprendidas de las actividades de las pruebas. de forma que pueda ser utilizadas para mejorar la madurez del proceso de prueba**

Consejos para progresar en Testing **Video 05**

10 tips:

1. **Estudiar, mantenerse estudiando, webinar, suscripción de grupos, mantenerse al tanto de las noticias.**
2. **No tengas miedo a exponer nuevas ideas, con diferentes enfoques agregando valor, presentándote al equipo y haciendo pruebas pilotos.**
3. **No responder de forma inmediata. Investiga sobre lo que vamos a reportar, de esta forma daremos una mejor descripción al problema y más elementos al equipo de desarrollo como encontrar el problema. explicar un camino de como llegar ahí.**
4. **Evidencias de los test, imágenes, videos.**
5. **Buena comunicación con los desarrolladores.**
6. **No tomar decisiones de la gerencia de forma personal, debemos alertarnos y estar pendiente de los errores del sistema como TESTER QA que somos.**
7. **Provee todo el tiempo toda la información que puedas sobre los tests y el estado del sistema,en las reuniones o todo momento anunciar esos errores siempre.**
8. **Debemos saber crear reportes técnicos, gerencias y para el usuario. Hay que saber a qué audiencia van los mensajes.**
9. **Dominar el negocio de la app que estamos probando. ¡Explorar toda la app siempre !**
10. **Mostrar el trabajo que hacemos.**
11. **Aprender a ejecutar varios tipos de pruebas y a la vez especializarse en una de ellas**

Cosas que necesitas saber si queres estudiar Testing **Video 06**

1. **Que es un tester de Software, a que se dedica ?** El objetivo es detectar fallas para poder reportarlas y que se corrijan antes de que la app sea utilizada por los usuarios finales. También se encarga de prevenir defectos, el tester se debe involucrar desde el principio antes de desarrollar el código, es decir desde la planificación de lo que se requiere hacer, casos de usos a implementar. Según la experiencia del tester va a hacer preguntas, detectando posibles riesgos. El tester también va a ayudar a mejorar el proceso y de esta forma mejorará la calidad de la aplicación.

2. **Se suele llamar tester a la persona o al rol que se ubica en el final del proceso. QA: Abarca todo el proceso.**

3. **Tester QA Manual: Tareas**
    - Participar en distintas reuniones donde se planifica y se idea el producto a desarrollador (Prevención) aportando una idea o mirada crítica con los conocimientos.
    - Participar en el diseño de la estrategia de pruebas.
    - Analizar los requerimientos de la estrategia de pruebas.
    - Diseñar casos de pruebas o escenarios a probar.
    - Explorar la app y ejecutar las pruebas diseñadas.
    - Reportar defectos de forma efectiva y hacerle un seguimiento.
    - Colaborar con los desarrolladores.
    - Colaborar con el QA Automation en seleccionar cuáles pruebas deben automatizarse.
    - Pensar y sugerir posibles mejoras en el proceso de la calidad.

4. **Tester QA Automation: Tareas**
    - Implementar el framework de pruebas automatizadas desde 0.
    - Programar las pruebas manuales que se indicaron para automatizar.
    - Monitorear la ejecución de las pruebas automatizadas.
    - Mantenimiento: Corregir pruebas en caso de cambios en la funcionalidad.
    - Soporte/ayuda al tester Manual para programar tareas repetitivas y complejas de hacer de forma manual.

¿Qué se necesita saber para ser tester ?

Soft skill:

- ***Comunicación oral y escrita.***
- ***Pensamiento crítico, indagar, ir un poco más allá, con el foco en construir.***
- ***Curiosidad, preguntar.***
- ***Creatividad.***
- ***Organización.***
- ***Trabajo en equipo.***
- ***Prolijidad.***

Hard skill:

- ***Entender como es el ciclo de vida del desarrollo del software.***
- ***Fundamentos de las pruebas (Que es caso de prueba, que son los requerimientos, que es calidad, qué es un defecto).***
- ***Diseñar casos de pruebas (como se diseñan, que debe tener un caso de prueba, como se estructura).***
- ***Reporte de bugs y su ciclo de vida (estructura y seguimiento, ciclo de vida)***
- ***Modelos de trabajo: Ágiles y cascada***
- ***Concepto de severidad VS prioridad, saber priorizar en base a riesgos.***
- ***Conocer herramientas de gestión de proyectos y pruebas (EJ: Jira + Zephyr)***
- ***Saber consultar bases de datos.***
- ***Herramienta de automatización.***
- ***Ingles poder entender, hacerte entender.***

Si queres ser Tester Automation tenes que saber aparte de lo anterior..

- POO
- Implementar un framework con una herramienta en particular

Temas destacados de unidad 1 y 2

Las pruebas manuales y automatizadas tienen sus propias ventajas y desventajas en diferentes escenarios. A continuación, te presento un análisis considerando aspectos como el tiempo de ejecución, la precisión, el costo y la facilidad de implementación:

Pruebas Manuales

Ventajas:

1. **Flexibilidad y Adaptabilidad:** Las pruebas manuales son más adaptables para casos complejos, nuevos o inusuales que no están bien definidos y pueden requerir juicio humano.
2. **Menor Costo Inicial:** No requieren inversión en herramientas de automatización o programación, lo que puede ser beneficioso para proyectos pequeños o con presupuesto limitado.
3. **Facilidad de Implementación:** No requieren habilidades técnicas avanzadas; cualquier tester con el conocimiento del producto puede realizar las pruebas.
4. **Mejor para Experiencias del Usuario:** Permiten evaluar la interfaz de usuario y la experiencia de uso desde una perspectiva humana, lo cual es esencial para detectar problemas que una máquina podría pasar por alto.

Desventajas:

1. **Tiempo de Ejecución:** Las pruebas manuales son lentas y laboriosas, especialmente para pruebas repetitivas o regresiones.
2. **Menor Precisión y Consistencia:** Las pruebas manuales están sujetas a errores humanos y pueden variar en resultados debido a la fatiga o descuidos del tester.
3. **No Escalables:** A medida que el proyecto crece, las pruebas manuales se vuelven insostenibles debido al tiempo y esfuerzo requeridos.
4. **Limitada Cobertura de Pruebas:** Dificultad para cubrir todas las combinaciones posibles de pruebas, lo que puede dejar huecos en la detección de errores.

Pruebas Automatizadas

Ventajas:

1. **Velocidad y Eficiencia:** Las pruebas automatizadas se ejecutan mucho más rápido que las manuales, lo que permite realizar miles de pruebas en poco tiempo.
2. **Precisión y Consistencia:** Eliminan el error humano y garantizan la repetición exacta de los pasos de prueba cada vez que se ejecutan.
3. **Costos a Largo Plazo:** Aunque la inversión inicial es alta, los costos se reducen con el tiempo al automatizar pruebas repetitivas y frecuentes.

4. **Escalabilidad y Cobertura:** Permiten ejecutar pruebas en múltiples entornos y plataformas simultáneamente, aumentando la cobertura de prueba sin incrementar significativamente los recursos necesarios.

Desventajas:

1. **Alto Costo Inicial y Complejidad de Implementación:** Requieren inversión en herramientas de automatización, infraestructura, y capacitación del personal en scripting y mantenimiento de pruebas.
2. **Rigidez:** Las pruebas automatizadas pueden fallar si el software cambia, ya que los scripts deben actualizarse para reflejar los cambios en la aplicación.
3. **No Adecuadas para Pruebas Exploratorias:** No pueden replicar la intuición humana necesaria para pruebas exploratorias y de experiencia de usuario.
4. **Mantenimiento Costoso:** Los scripts de prueba automatizados requieren mantenimiento continuo para mantenerse relevantes y efectivos, especialmente en aplicaciones con cambios frecuentes.

Conclusión

- **Pruebas Manuales:** Son ideales para entornos dinámicos, pruebas exploratorias y cuando se necesita la perspectiva del usuario. Sin embargo, no son escalables y son propensas a errores humanos.
- **Pruebas Automatizadas:** Son excelentes para pruebas repetitivas, regresiones y para garantizar la consistencia y precisión en aplicaciones estables. No obstante, requieren una inversión inicial significativa y un mantenimiento continuo.

La elección entre pruebas manuales y automatizadas dependerá de los requisitos específicos del proyecto, presupuesto y recursos disponibles.

Unidad 2 - Aspectos claves para las pruebas

Introducción a la unidad 2

En la unidad anterior, exploramos el testeo de software, su definición, propósito, objetivos, usos y su importancia en la industria del desarrollo de software. Ahora, en esta nueva unidad, profundizaremos en aspectos clave para comprender mejor el testeo de software. Es fundamental formar profesionales competentes y garantizar la calidad del producto final.

Destacamos la importancia de estudiar los temas siguientes en el curso:

- Calidad del Software: Esencial para garantizar que el producto final cumpla con estándares y expectativas de usuarios en funcionalidad, eficiencia, seguridad y usabilidad.

- Causas de los Defectos: Crucial para prevenir futuros defectos colaborando con desarrolladores para mejorar la calidad del código.

- Necesidad de Pruebas de Software: Proceso crítico para detectar y corregir defectos antes de que lleguen a los usuarios, reduciendo costos y problemas.

- Ciclo de Vida de un Defecto: Permite a los tester rastrear y gestionar los problemas encontrados durante las pruebas de manera efectiva. Incluye la identificación, documentación, resolución y verificación de los defectos.

- Escenarios de Prueba de Software: Esencial para evaluar funcionalidad y confiabilidad del software en diversas situaciones.

Estudiar estos temas es esencial para formar profesionales altamente capacitados, asegurar estándares de calidad, prevenir defectos, realizar pruebas efectivas y gestionar problemas correctamente. En esta segunda unidad, nos enfocaremos en la calidad del software y los procesos de pruebas, explorando las funciones y aplicaciones de las pruebas de software, y su importancia en el desarrollo de productos confiables y de calidad.

Principios de las pruebas del software **Video 01**

1. ***Las pruebas demuestran la presencia de defectos, no su ausencia.***
    a. Es decir, las pruebas demuestran la presencia de defectos, pero no pueden probar que no los hay.
2. ***Las pruebas exhaustivas no existen.***
3. ***Las pruebas tempranas ahorran tiempo y dinero.***

4. ***Agrupación de defectos.***
    - ***a.*** Un software puede ser grande pero la concentración de los defectos son en funciones.
5. ***Tener cuidado con la paradoja del pesticida. (Pruebas realizadas una y otra vez).***
    a. Para detectar nuevos defectos, debemos actualizar las pruebas existentes, los datos de las pruebas y también escribir pruebas nuevas.
6. ***Las pruebas dependen del contexto.***
    - ***a. Ejemplo = diferentes una prueba en un sitio web que muestra información contra una app de avión de pasajeros.***
7. ***Falacia de ausencia de errores.***

Calidad del software **Video 02**

¿Qué es ?

- **A.** Proceso eficaz de software que se aplica de manera que crea un producto útil que proporciona valor medible a quienes lo producen y a quienes lo utilizan.
- **B.** Puntos de vista de David Garvin 84’
    a. **Transcendental**: Se reconoce pero es difícil de definir
    b. **Usuario**: cumple los requerimientos y funcionalidad
    c. **Fabricante**: cumple las especificaciones originales.
    d. **Producto**: Implementación de funciones y características.
    e. **Valor**: lo que el cliente está dispuesto a pagar.

¿Quién lo consigue ? Todos los involucrados en el proceso.

¿Por qué es importante? Reduce costos por repetición y mejora entrada al mercado.

Atributos de Calidad ISO 9126

- Funcionalidad: Adaptabilidad, exactitud, interoperabilidad, cumplimiento y seguridad.
- Confiabilidad: madurez, tolerancia a fallas y robustez.
- Usabilidad: entendible, intuitiva y operable.

- Eficiencia: Comportamiento del tiempo y de los recursos
- Facilidad de recibir mantenimiento: analizable, cambiable, estable y susceptible de someterse a pruebas.
- Portabilidad: adaptable, instalable, conformidad y sustituible

Test de comprobación de atributos: lo vamos a realizar para que nuestro software sea de calidad.

- Prototipo de interfaz de usuario -
    - **Intuitiva**
        - ¿todas las operaciones son fáciles de localizar e iniciar ?
        - La interfaz usa patrones esperados de uso ?
    - **Eficiencia** (localizar información o iniciar operaciones)
        - Economía de movimientos para entradas de datos y operaciones?
        - Datos de salida están presentados para facilitar su legibilidad?

Si estas preguntas son afirmativas, el software probablemente es de alta calidad.

Para cada factor de la calidad que se debe evaluar se desarrollarán preguntas similares.

Causa de los defectos **Video 03**

Que es un defecto ? Cuando el resultado esperado es DISTINTO al resultado actual. Esta presente cuando el software no hace lo que tiene que hacer funcionalmente o en sus atributos de calidad.

Campos básicos para reportar el defecto/ Estructura.

- Título del defecto (Una sola línea)
- Descripción (detalle del defecto)
- Ambiente (en que sistema operativo, que hardware)
- Versión del software
- ID del caso de prueba
- Procedimiento o script de prueba.
- Severidad
- Prioridad
- Evidencia del defecto

Tipos de defecto

- ***Funcionales (ejecutando pruebas de sistemas)***
- ***Seguridad***
- ***Estáticos (Documentos)***
- ***Desempeño (pruebas de rendimiento, estrés o carga)***

Principales tipos de defectos

Los defectos de software pueden clasificarse en varias categorías según la naturaleza del problema. Aquí tienes una descripción detallada de cada tipo mencionado en la imagen:

1. **Defectos Funcionales**:
    - Se refieren a errores que afectan las funcionalidades principales del software, es decir, cuando el sistema no cumple con los requisitos funcionales especificados. Estos defectos ocurren cuando una función no produce el resultado esperado o se comporta de manera incorrecta. Ejemplos comunes incluyen errores en cálculos, fallos en el manejo de

datos, o funciones que no responden adecuadamente a las acciones del usuario.

2. **Defectos de Seguridad**:
    - Son defectos relacionados con vulnerabilidades en el sistema que podrían ser explotadas para comprometer la confidencialidad, integridad o disponibilidad de la información. Estos defectos incluyen errores que permiten el acceso no autorizado a datos sensibles, fallos en la autenticación y autorización, o la falta de encriptación de datos críticos. Ejemplos incluyen inyección de SQL, Cross-Site Scripting (XSS), y fallos en la gestión de sesiones.
3. **Defectos Estáticos**:
    - Se detectan sin ejecutar el código, a través de la revisión del código fuente, análisis estático, o inspección de diseño. Estos defectos incluyen errores de sintaxis, violaciones de convenciones de codificación, errores de declaración y tipificación de variables, y otros problemas que pueden identificarse mediante la inspección visual o herramientas automáticas de análisis de código.
4. **Defectos de Desempeño**:
    - Relacionados con problemas que afectan la velocidad, estabilidad y capacidad del sistema. Estos defectos ocurren cuando el software no cumple con los criterios de rendimiento, como tiempos de respuesta lentos, uso excesivo de memoria o fallos bajo alta carga de usuarios. Ejemplos incluyen tiempos de carga prolongados, bloqueos durante el uso intensivo, y fallos en pruebas de estrés y carga.

Cada uno de estos tipos de defectos puede impactar negativamente la experiencia del usuario y la calidad general del software, por lo que es fundamental identificarlos y corregirlos durante el ciclo de desarrollo.

Necesidad de las pruebas de software **Video 04**

Ciclo de vida de un defecto **Video 05**

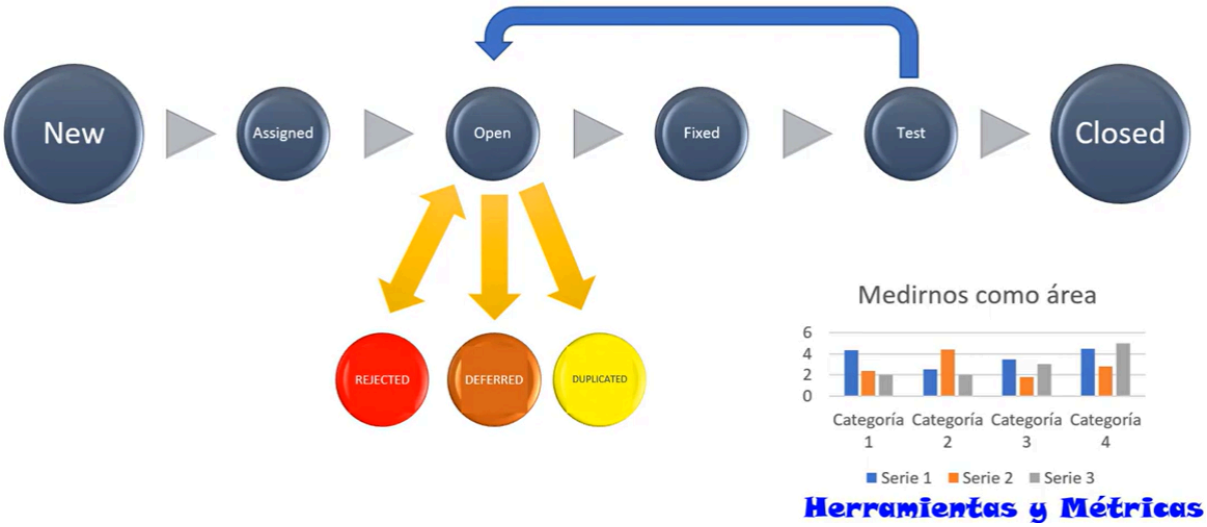
Si encontramos un defecto debe ser registrado y reportado al área de desarrollo para su correción.
Este defecto al ser registrado se va a poner con un ***STATUS NUEVO,*** este va a ser enviado al área de desarrollo y pasa a ponerse en un **STATUS** de ***ASIGNADO,*** ya no pertenece al área de QA sino al área de desarrollo. El lider de desarrollo asigna al equipo de desarrollador y cambia de ***STATUS a ABIERTO*** *para que el desarrollador* comience a evaluarlo, analizarlo y decidir si es un defecto o no.

Si este defecto lo corrige pasa a otro **STATUS FIXED,** este status pertenece al área de QA y el lider lo cambia a un **STATUS TEST** y el equipo de prueba se encarga de la validaciones de este defecto y si este fue corregido el **STATUS** cambio a **CLOSED.**

Que sucede si este defecto no fue corregido ? Si es así los de QA vuelven a enviar el defecto al ***STATUS OPEN / REOPEN*** y cae al lado de desarrollo nuevamente.
Puede pasar 3 cosas en este caso :

- Que el equipo de desarrolladores lo ***Rechace, si está rechazado queda del lado de QA y entra una negociación con los desarrolladores***
- ***El equipo de desarrolladores lo DIFIERE, es decir quizás lo corrigen para otra versión del software***
- ***El desarrollador lo puede colocar como DUPLICADO, es decir este defecto ya fue levantado con anterioridad de otros tester***

Aca se negocia con el desarrollador y se puede llevar a CERRADO



ESCENARIOS DE PRUEBAS DE SOFTWARE ***VIDEO 06***

Son una descripción simple de lo que vamos a probar (Descripción en una oración de la funcionalidad a probar), “Verificar el proceso de registro”.

Diferencias con los casos de pruebas.
Escenario: Es corto, nos dice QUE probar, al ser más genérico podemos realizarlo en menos TIEMPO para escribirlo y ejecutarlo.

Caso de prueba: Es más detallado, aca nos dice QUE probar y COMO probar, ya que nos detalla los paso a seguir, tener un mayor TIEMPO para escribirlo y ejecutarlo porque deberíamos de seguir el paso a paso el caso que está diseñado.

¿Por qué quisiéramos usar casos de pruebas ? Los escenarios de prueba ofrecen flexibilidad, permitiendo variaciones en las pruebas y exploración de diferentes áreas funcionales. Esto es especialmente útil en entornos de trabajo ágiles.

Situaciones al utilizar los escenarios de pruebas:

1. Una situación donde los escenarios son particularmente útiles es cuando el tiempo es acotado, ya que son muchos más rápidos de ejecutar y es posible cubrir un mayor alcance con menos tiempo. Esto es crucial en metodologías ágiles.
2. Cuando las funcionalidades cambian muy rápido. Esto se da cuando la funcionalidad cambia y los escenarios de prueba no apliquen y haya que actualizar los pasos llevando un costo y tiempo de actualización.
3. Cuando tenemos un equipo “MADURO”, si el equipo lo es y conoce la funcionalidad y ya viene trabajando veremos que cosas probar en cada escenario.
    a. Los CASOS DE PRUEBA es más adecuado cuando hay equipo nuevo o si hay muchas personas nuevas en el proyecto.

¿Cómo identificar los escenarios de prueba ?

- Lo primero que debemos hacer es ponernos en el lugar del usuario

Nos muestra ejemplos

EJEMPLO: MercadoLibre

Verificar la BÚSQUEDA DE PRODUCTOS

● Verificar el autocompletado

● Verificar búsqueda desde sugerencias
● Verificar búsqueda con Enter
● Verificar búsqueda desde botón

Pregunta del QUIZ

Desarrollar escenarios de prueba efectivos requiere una comprensión profunda de los requisitos del usuario y del sistema. Al basar los escenarios de prueba en esta comprensión, se asegura que todas las funcionalidades y posibles errores sean evaluados exhaustivamente, lo que conduce a un software más robusto y de mayor calidad.

Analisis Causa Raiz **VIDEO 07**

Objetivos:

- Entender el porque se genera el problema o defecto
- Determinar los factores del problema
- Eliminar causas del problema

¿Cómo se realiza este análisis?

1. ***Identifique el problema:*** Descripción del problema sencillo y ser objetivo

2. ***Defina la raíz del problema:*** Usando herramientas como tormenta de ideas, 5 porque, entender la razón porque se origina el problema
3. ***Desarrolle la solución:*** plan de acción, describiendo la implementación de la solución, indicando tareas, responsables y duración de las mismas.
4. ***Monitoreo:*** Seguir bien de cerca el nuevo proceso, con la intención de asegurarse que se ejecuta bien
5. ***Validación/Solución:*** Comprobación de la solución, demostrar que el problema se eliminó.

Pregunta del QUIZ

El análisis causa raíz es una técnica efectiva para identificar la fuente fundamental de los defectos en el software. Al entender y abordar la raíz del problema, se pueden implementar soluciones duraderas que prevengan la recurrencia de defectos similares en el futuro, mejorando así la calidad y la confiabilidad del software a largo plazo.

Bug reportado : La aplicación se bloquear cuando un usuario intenta cargar una imagen en su perfil

PASOS:

1. Identificación del problema:
    a. Descripción: Al intentar cargar una imagen, la aplicación se cierra inesperadamente.
    b. Frecuencia: sucede siempre que se suben imágenes mayores a 5MB.
2. Recolección de datos:
    a. Revisar los registros (logs) de la app.
    b. Replicar el error para observar los síntomas exactos.
3. Análisis del problema:
    a. Los logs muestran un error de “Out of Memory” al cargar imágenes grandes.
    b. Identificación del patrón: El problema ocurre con archivos grandes y en dispositivos con poca memoria disponible.
4. Identificación de la Causa Raiz:
    a. Causa Raíz: La app no limita el tamaño de las imágenes cargadas y no maneja adecuadamente la memoria durante la operación de carga.

5. Acciones correctivas:
    a. Implementar validaciones para limitar el tamaño de los archivos que pueden ser subidos.
    b. Optimizar el manejo de la memoria al procesar imágenes grandes.
6. Verificación:
    a. Probar la solución con diferentes tamaños de imágenes para asegurar que el bug no vuelva a ocurrir.
7. Prevención:
    a. Actualizar los casos de prueba para incluir la validación del tamaño de los archivos.
    b. Revisar otras funcionalidades que manejen archivos para aplicar mejoras similares.

Resumen:

El RCA ayudó a descubrir que el verdadero problema no era solo la imagen grande, sino la falta de validación de tamaño y gestión ineficiente de memoria. La implementación de estas correcciones no solo solucionó el bug reportado, sino que también mejoró la estabilidad general de la app.
Este proceso asegura que no solo se solucione el síntoma del bug, sino que se aborden las causas subyacentes para evitar problemas futuros.