Introducción a las Pruebas de Software

Objetivo

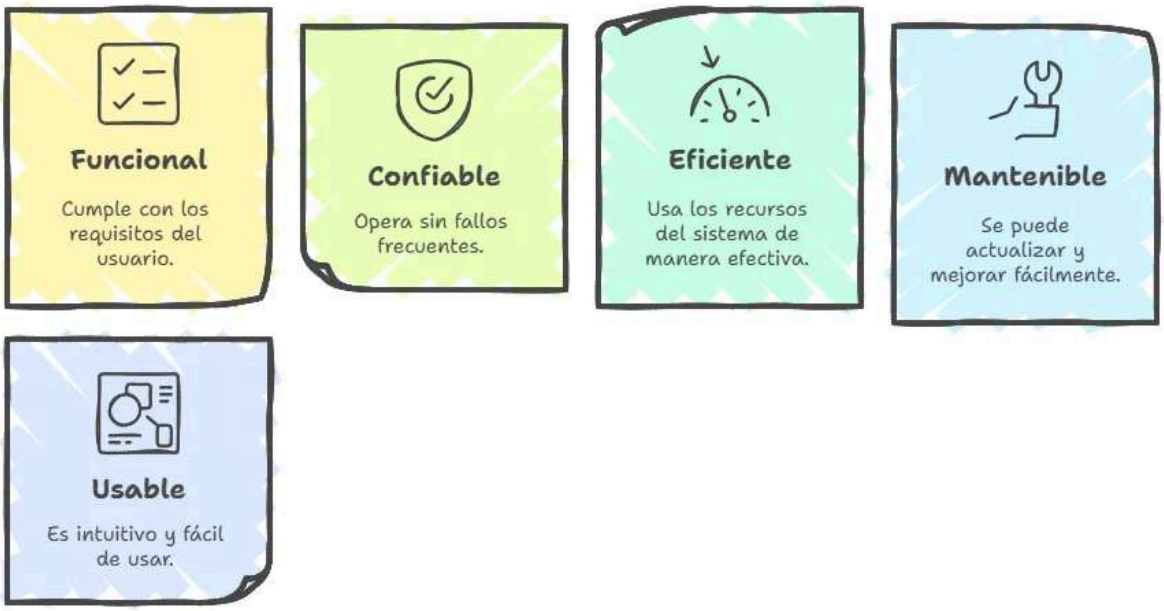
Comprender la importancia del testing en el ciclo de desarrollo de software y familiarizarse con los conceptos fundamentales de calidad y pruebas de software.

---

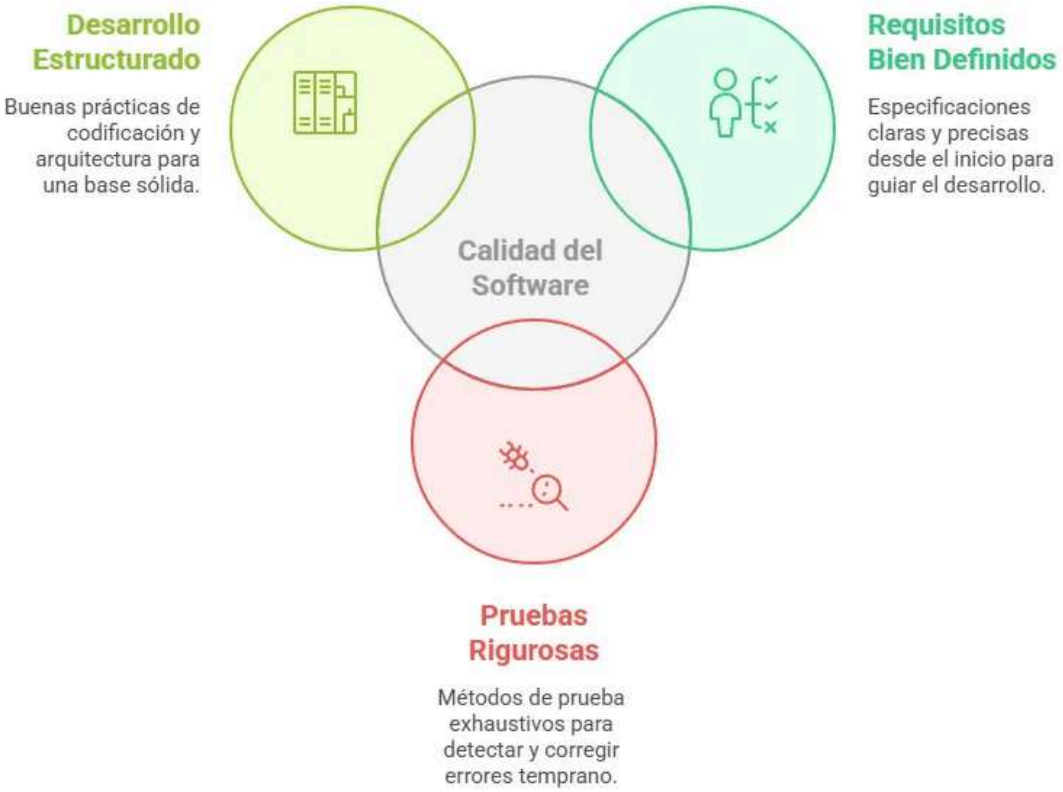
1. Fundamentos de la Calidad del Software

1.1. ¿Qué es la Calidad del Software?

La calidad del software se refiere a la capacidad de un sistema para cumplir con los requisitos especificados y satisfacer las necesidades del usuario. Un software de calidad debe ser:



1.2. Factores que influyen en la calidad del software



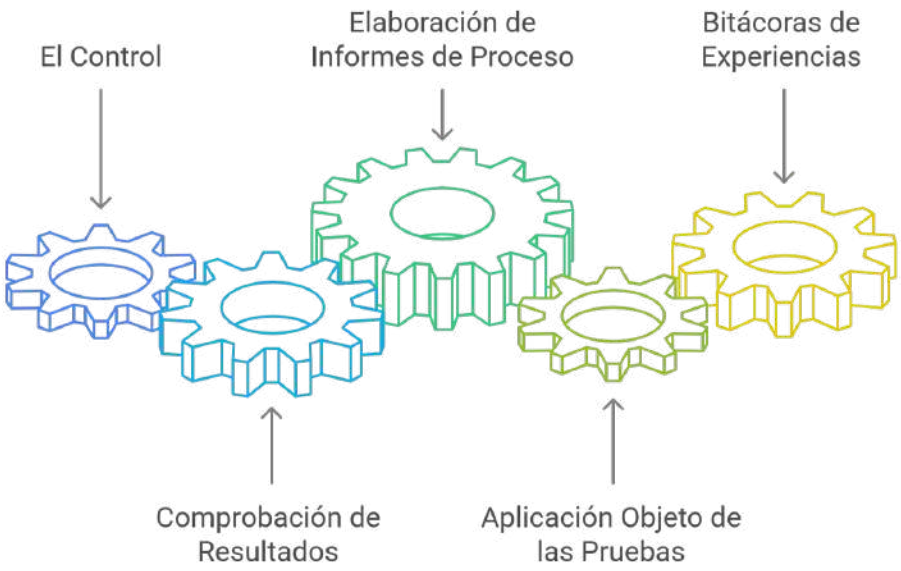
2. Definición y Conceptos Clave en Testing

2.1. Qué es Testing

El **Testing de software** es un proceso sistemático de evaluación y validación de un sistema o aplicación para garantizar que funcione correctamente y cumpla con los requisitos especificados. Su propósito es detectar errores, defectos o fallos antes de que el software llegue a producción, minimizando riesgos y mejorando la calidad del producto final.

Las pruebas *software* son parte fundamental del proceso de calidad y adopción de buenas prácticas, estas deben realizarse en todo el proceso de desarrollo de *software*, según Mera (2016) este proceso debe incluir: *Revisión de los requerimientos, realización de análisis documentales, identificación de defectos, pruebas funcionales y no funcionales, pruebas dinámicas y estáticas, pruebas de integración, informes de confianza en el nivel de calidad, información para la toma de decisiones, planes de mejora continua.*

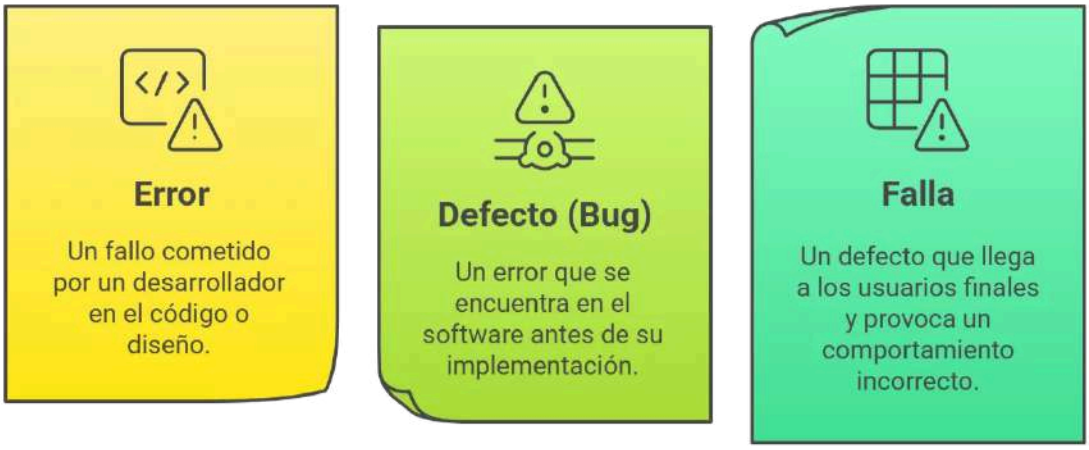
Este mismo autor afirma que dentro de las actividades de un proceso de prueba se debe incluir las siguientes fases:



- **El Control**: Se refiere a la planificación y gestión de las actividades de prueba para garantizar que se ejecuten de manera eficiente y organizada.

- **Comprobación de Resultados**: Implica la comparación de los resultados obtenidos con los resultados esperados para determinar si el software funciona correctamente.
- **Elaboración de Informes de Proceso**: Documentación de hallazgos, defectos y resultados de las pruebas, lo que permite una mejora continua en el desarrollo.
- **Aplicación Objeto de las Pruebas**: Se refiere al sistema o software que está siendo evaluado a través de pruebas.
- **Bitácoras de Experiencias**: Registro de incidencias, aprendizajes y mejoras detectadas durante el proceso de prueba para futuras referencias y optimización.

2.2. Error, Defecto y Falla



Ejemplo:
Un programador olvida inicializar una variable (
error), lo que causa un resultado inesperado en el código (*defecto*). Cuando el usuario final experimenta un fallo en la aplicación, se considera una *falla*.

---

3. Principios Fundamentales de las Pruebas de Software

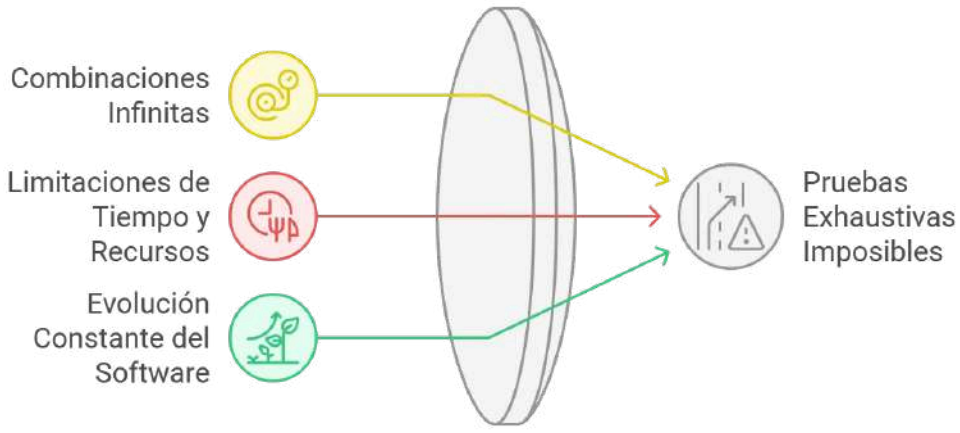
▼ 3.1. Las pruebas muestran la presencia de defectos, no su ausencia.

- Si una prueba detecta un defecto, es evidente que el software tiene problemas.
- Sin embargo, si una prueba no encuentra errores, no significa que el sistema esté libre de fallos, solo que no se han identificado en ese momento con los casos de prueba ejecutados.
- Existen defectos que pueden aparecer en condiciones específicas o que no fueron contempladas en las pruebas realizadas

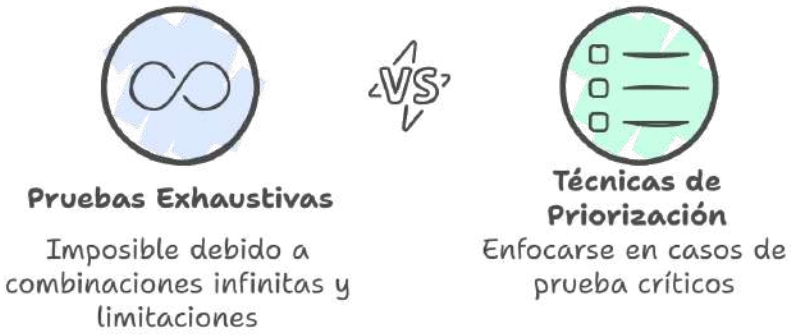
▼ 3.2. Las pruebas exhaustivas son imposibles.

es inviable probar absolutamente todas las combinaciones posibles de entradas, caminos de ejecución y escenarios en un software. Esto se debe a varias razones:

- **Cantidad infinita de combinaciones**: En sistemas complejos, la cantidad de combinaciones de datos de entrada y estados posibles es tan grande que probar todas sería impráctico.
- **Limitaciones de tiempo y recursos**: No se dispone de tiempo ni presupuesto infinito para ejecutar todas las pruebas posibles.
- **Evolución constante del software**: Las aplicaciones se actualizan constantemente, lo que hace que nuevas funcionalidades y cambios en el código requieran pruebas continuas.



En lugar de realizar pruebas exhaustivas, se utilizan técnicas como **partición de equivalencia**, **análisis de valores límite**, **pruebas basadas en riesgo** y **automatización**, priorizando los casos de prueba más críticos.



▼ 3.3. Las pruebas tempranas ahorran tiempo y dinero.

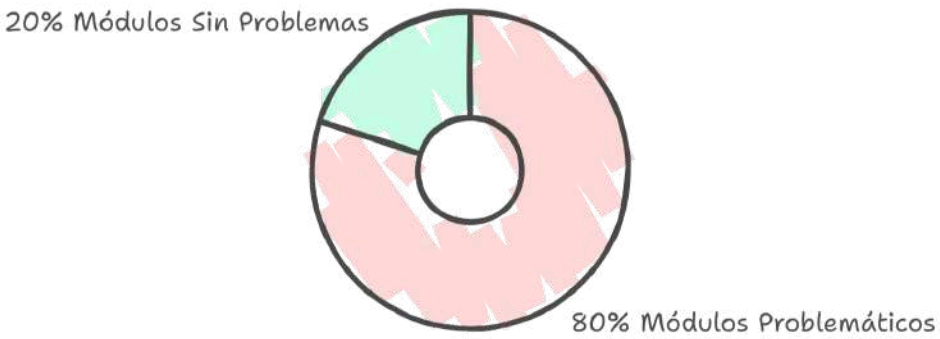
- **Corrección más económica:** Un defecto encontrado en la fase de requisitos es más barato de corregir que uno detectado en producción.
- **Menos retrabajo:** Evita que errores tempranos se propaguen a otras partes del sistema, reduciendo el esfuerzo de desarrollo.
- **Mayor calidad del producto:** Al identificar problemas desde el inicio, se mejora la estabilidad del software antes de que llegue a los usuarios finales.
- **Entrega más rápida:** Reducir errores desde el principio disminuye el tiempo dedicado a solucionar problemas en fases críticas.

Ejemplo:

Si en la fase de diseño se detecta que un requisito es ambiguo o incorrecto, se puede corregir con solo modificar la documentación. Sin embargo, si el problema se detecta en producción, podría requerir una reescritura de código, pruebas adicionales y un retraso en la entrega.

▼ 3.4. La acumulación de defectos sigue la ley de Pareto (80/20).

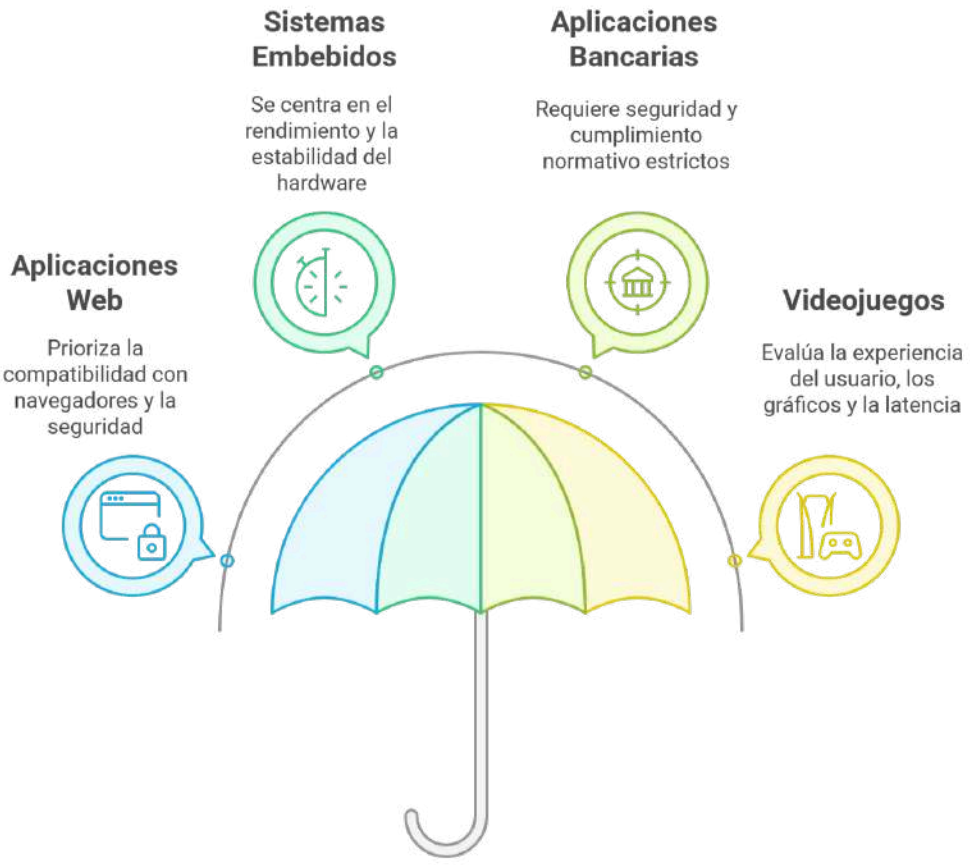
en la mayoría de los casos, **el 80% de los defectos provienen del 20% de los módulos o componentes del software**. Esto significa que ciertos módulos suelen concentrar la mayor cantidad de errores debido a factores como su complejidad, modificaciones frecuentes o mal diseño.



▼ 3.5. Las pruebas dependen del contexto del software.

el enfoque y los tipos de pruebas que se aplican a un sistema dependen de sus características específicas, su entorno de desarrollo y los objetivos del proyecto. No todas las pruebas son aplicables de la misma manera en todos los sistemas, ya que cada software tiene requisitos, riesgos y usuarios diferentes.

Ejemplos de cómo el contexto influye en las pruebas:



Implicaciones en el Testing:

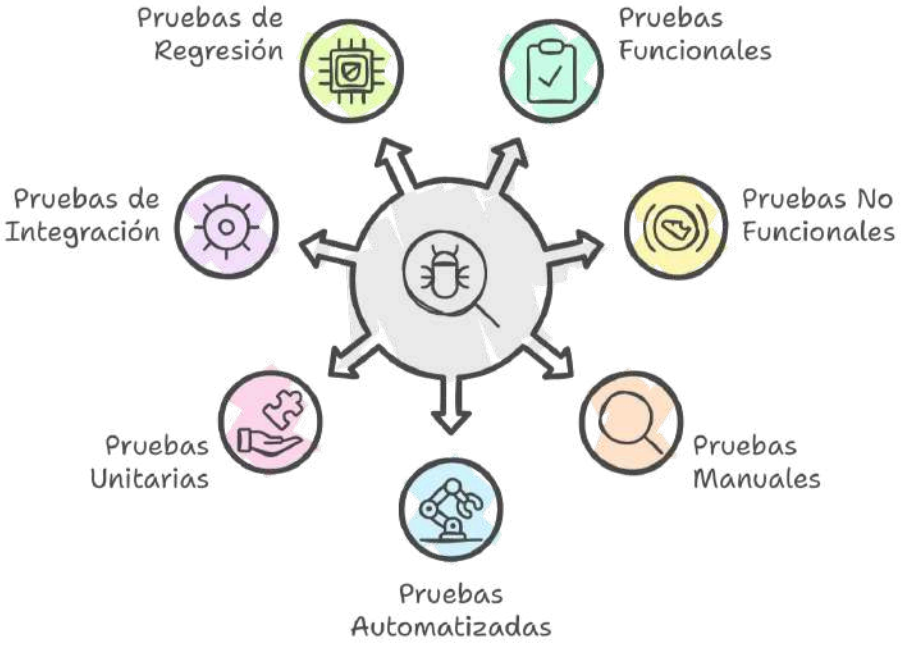
- Se deben adaptar las estrategias de prueba según el tipo de software.
- La elección entre pruebas manuales o automatizadas depende del contexto.
- En metodologías como Agile, el contexto requiere pruebas continuas y automatización frecuente.

▼ 3.6. La paradoja de las pruebas: si las pruebas se repiten demasiado, pierden efectividad.

Cuando los mismos casos de prueba se ejecutan una y otra vez sin cambios ni nuevas estrategias, es menos probable que detecten defectos adicionales.

Si un equipo de testing ejecuta repetidamente un conjunto de pruebas automatizadas sobre una aplicación web, pero nunca actualiza los casos de prueba para considerar nuevos flujos o interacciones del usuario, es probable que los errores recién introducidos pasen desapercibidos.

4. Tipos de Pruebas de Software



Tipo de Prueba	Descripción
Pruebas Funcionales	Verifican que el software cumpla con los requisitos funcionales.
Pruebas No Funcionales	Evalúan rendimiento, seguridad y usabilidad del software.
Pruebas Manuales	Se realizan sin automatización, verificando la aplicación manualmente.
Pruebas Automatizadas	Uso de herramientas para ejecutar pruebas rápidamente.
Pruebas Unitarias	Se prueban pequeñas unidades de código.
Pruebas de Integración	Se validan interacciones entre módulos del sistema.
Pruebas de Regresión	Se aseguran de que nuevas modificaciones no rompan funciones previas.

5. Estrategias de Pruebas en Desarrollo Web

Las estrategias de pruebas en desarrollo web tienen como objetivo garantizar que una aplicación web sea funcional, segura, eficiente y accesible para los usuarios. Algunas estrategias clave incluyen:

▼ 5.1. Pruebas de Compatibilidad

- Validan que la aplicación web funcione correctamente en diferentes navegadores (Chrome, Firefox, Edge, Safari) y dispositivos (PC, tabletas, móviles).
- Se prueban distintas resoluciones de pantalla y configuraciones de hardware/software.

▼ 5.2. Pruebas de Rendimiento

- Se realizan pruebas de carga para medir la capacidad del sistema bajo alta concurrencia de usuarios.
- Se aplican pruebas de estrés para evaluar la estabilidad ante condiciones extremas.
- Se realizan pruebas de escalabilidad para verificar el comportamiento con el aumento progresivo de usuarios y datos.

▼ 5.3. Pruebas de Seguridad

- Detectan vulnerabilidades como inyección SQL, ataques de cross-site scripting (XSS) y problemas de autenticación/autorización.
- Se aplican pruebas de penetración para evaluar la capacidad del sistema para resistir ataques.
- Se revisan configuraciones de seguridad en servidores y bases de datos.

▼ 5.4. Pruebas de Usabilidad y Accesibilidad

- Evalúan la experiencia del usuario para garantizar que la interfaz sea intuitiva y fácil de usar.
- Se validan estándares de accesibilidad (WCAG) para garantizar que la web sea usable por personas con discapacidad.
- Se aplican pruebas A/B para analizar la interacción de los usuarios con la interfaz.

▼ 5.5. Pruebas de Integración y API

- Se validan las interacciones entre el frontend y backend mediante pruebas de API.
- Se utilizan herramientas como Postman o SoapUI para verificar respuestas de servicios web.
- Se aseguran que las integraciones con terceros (como pasarelas de pago o servicios en la nube) funcionen correctamente.

---

6. Actividad Práctica: Identificación de Defectos en Aplicaciones Web

Objetivo:

Aplicar los conceptos aprendidos identificando errores en una aplicación web.

Instrucciones:

1. Accede a una página web de prueba o una aplicación en desarrollo.
2. Navega y busca defectos en la interfaz y en las validaciones.
3. Documenta los errores encontrados indicando:

- Descripción del defecto.
- Pasos para reproducirlo.
- Resultado esperado vs. resultado obtenido.

4. Reporta los defectos en un documento de texto.

Material de Apoyo

- Libro: "Software Testing Foundations" - Andreas Spillner.
- Curso gratuito de ISTQB Foundation Level (YouTube, Udemy).
- Herramientas sugeridas: Chrome DevTools, Postman, Jira (versión gratuita).
