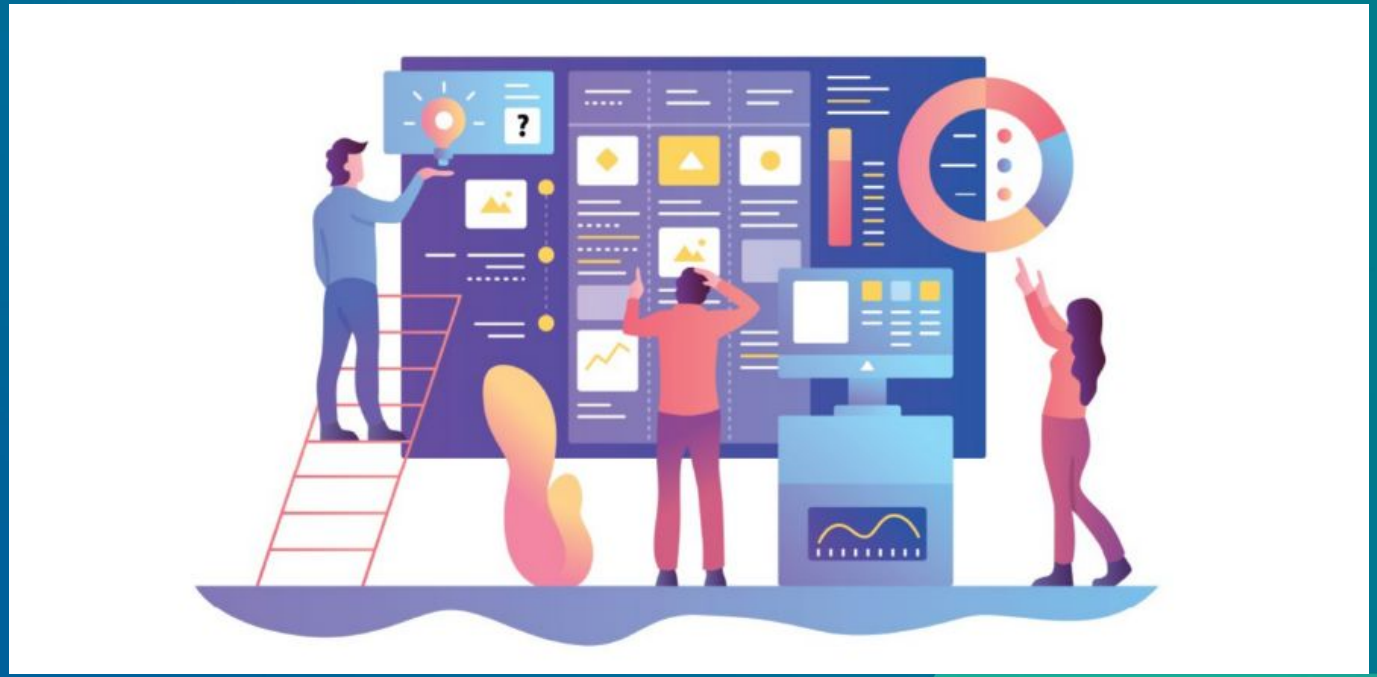
UTN AVELLANEDA
Tecnicatura Universitaria en Programación

# Metodología de Sistemas I (Plan 2024)

Gustavo Julián RIvas



Gustavo Julián RIvas

# Pruebas de Software

# Introducción a las Pruebas de Software

Las pruebas de software son el proceso de evaluar y verificar que un programa o aplicación funcione como se espera. Esto implica comprobar su funcionamiento contra los requerimientos y asegurar que no presente errores críticos.

**Objetivos**:

- **Detección de errores**: Identificar fallos en las funcionalidades.
- **Mejorar la calidad del software**: Asegurar que el producto sea robusto y confiable para el usuario final.
- **Validación de requerimientos**: Confirmar que el software cumple con los requisitos especificados.

# Errores, Defectos y Fallos en Software, diferencias entre conceptos:

**Error**: Una acción incorrecta o equivocada en el código por parte del desarrollador.
Ejemplo: el uso de una fórmula errónea para calcular impuestos.

**Defecto**: Una anomalía en el software que puede causar un fallo en ciertas condiciones.
Ejemplo: un error de lógica que impide completar un pedido en un e-commerce.

**Fallo**: La manifestación visible de un defecto que impide que el sistema funcione como se espera.
Ejemplo: un sistema de reservas de hotel que muestra habitaciones como disponibles cuando en realidad están ocupadas

**Impacto en el desarrollo**: los errores iniciales pueden convertirse en fallos visibles si no se detectan a tiempo.

Es más económico y efectivo detectar y corregir errores en etapas tempranas (fase de desarrollo) que cuando el software está en producción.

## Beneficios de Verificación y Validación (V&V)

**Verificación**: Permite asegurar que el producto cumple con las especificaciones técnicas antes de pasar a producción.

**Validación**: Asegura que el software cumple con las expectativas y necesidades del cliente final.

**Casos de éxito**: Empresas de la industria financiera que aplican V&V para garantizar seguridad y precisión en sus aplicaciones.

**Ventajas**: Asegura que el producto no solo esté libre de defectos técnicos, sino que también sea funcional y útil para el usuario final.

# Roles en Pruebas de Software

## Tester / Ingeniero de Pruebas

**Responsabilidad**: Los testers son responsables de diseñar, ejecutar y mantener pruebas. Se aseguran de que el software cumpla con los requisitos y se comporte según lo esperado. Sus tareas incluyen la redacción de casos de prueba, la ejecución de pruebas manuales y automatizadas, y la documentación de defectos.

**Ejemplo de tareas**:
Crear y mantener planes de prueba.
Realizar pruebas de regresión y pruebas funcionales.
Colaborar con desarrolladores para reproducir errores y validar correcciones.

## Analista de Calidad (QA Analyst)

**Responsabilidad**: Los analistas de calidad se centran en la calidad general del producto y trabajan para identificar mejoras en el proceso de pruebas. Evalúan los requisitos y ayudan a definir los criterios de aceptación. También son responsables de garantizar que los estándares de calidad se mantengan a lo largo del ciclo de vida del software.

**Ejemplo de tareas**:
Revisar requisitos y especificaciones para asegurarse de que sean claros y completos.
Definir métricas de calidad y realizar auditorías de calidad.
Colaborar con otros equipos para promover la calidad desde el inicio del proyecto.

## Analista de Calidad (QA Analyst)

**Responsabilidad**: Los analistas de calidad se centran en la calidad general del producto y trabajan para identificar mejoras en el proceso de pruebas. Evalúan los requisitos y ayudan a definir los criterios de aceptación. También son responsables de garantizar que los estándares de calidad se mantengan a lo largo del ciclo de vida del software.

**Ejemplo de tareas**:
Revisar requisitos y especificaciones para asegurarse de que sean claros y completos.
Definir métricas de calidad y realizar auditorías de calidad.
Colaborar con otros equipos para promover la calidad desde el inicio del proyecto.

## Automatizador de Pruebas (QA Automation)

**Responsabilidad**: Los automatizadores de pruebas son responsables de la creación de scripts y herramientas de automatización para facilitar las pruebas. Su enfoque está en mejorar la eficiencia del proceso de pruebas mediante la automatización de tareas repetitivas y la creación de entornos de prueba.

**Ejemplo de tareas**:
Desarrollar y mantener frameworks de automatización de pruebas.
Escribir scripts de prueba automatizados utilizando lenguajes de programación y herramientas de automatización.
Colaborar con testers para identificar oportunidades de automatización.

## Desarrollador de Software

**Responsabilidad**: Los desarrolladores son responsables de escribir el código y, en muchos casos, de realizar pruebas unitarias para asegurar la calidad de su propio trabajo. A menudo colaboran estrechamente con testers para abordar defectos y mejorar la calidad del producto.

**Ejemplo de tareas**:
Escribir y ejecutar pruebas unitarias.
Corregir defectos encontrados por el equipo de pruebas.
Participar en revisiones de código y sesiones de pruebas para mejorar la calidad del producto.

## Product Owner / Gerente de Producto

**Responsabilidad**: Aunque no son testers en sí, los Product Owners son clave en el proceso de pruebas, ya que definen los requisitos y criterios de aceptación del producto. Su responsabilidad es asegurarse de que el producto final cumpla con las expectativas del cliente y del mercado.

**Ejemplo de tareas**:
Priorizar las historias de usuario en el backlog.
Definir criterios de aceptación claros y medibles para cada historia de usuario.
Revisar y validar el trabajo del equipo de pruebas en función de los requisitos del cliente.

# Ciclo de Vida de los Defectos

**Fases**:

- **Identificación**: El defecto es encontrado y reportado.
- **Clasificación**: Se analiza el impacto y se clasifica por severidad.
- **Asignación**: Se asigna al equipo adecuado para su resolución.
- **Corrección**: Los desarrolladores implementan los cambios necesarios para resolver el defecto.
- **Verificación**: Los testers verifican si el defecto ha sido solucionado sin introducir nuevos errores.

**Ejemplo**: Un defecto detectado en la fase de pruebas de un sistema de ventas se clasifica como crítico, se asigna al equipo de backend para corregirlo, y luego es validado por los testers.

# Clasificación de Defectos por Severidad

**Definición y tipos**:

**Crítico**: Impide completamente el funcionamiento del sistema o de una función clave (ej. sistema de pagos no funciona).
**Mayor**: Funcionalidad limitada, pero el sistema puede seguir operando con restricciones (ej. problemas en el procesamiento de facturas).
**Menor**: No afecta funciones principales, pero causa inconvenientes en la usabilidad (ej. errores de formato o visuales).
**Trivial**: Errores sin impacto directo en la funcionalidad (ej. errores tipográficos).

**Ejemplo detallado**: Clasificación de un defecto que impide que un cliente realice un pago como "Crítico", ya que afecta directamente los ingresos.

# Clasificación de Defectos por Prioridad

**Definición y criterios de clasificación**:

**Alta**: Defectos que requieren resolución inmediata (ej. defectos que afectan la funcionalidad en el login).
**Media**: Defectos importantes pero que no requieren atención inmediata (ej. problemas en funciones menos usadas del sistema).
**Baja**: Defectos de menor impacto, sin urgencia de corrección (ej. problemas en elementos secundarios de la interfaz).

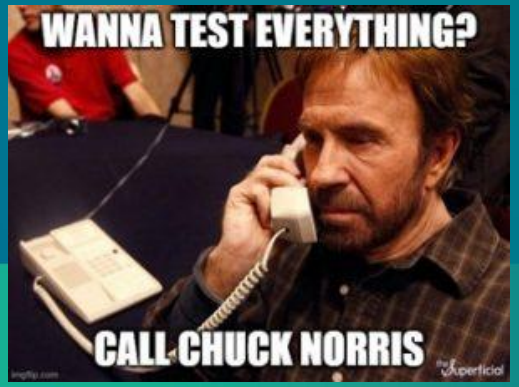
**Ejemplo práctico**: En una app de ecommerce, un defecto de prioridad alta sería uno que afecta la función de pago, mientras que un defecto de baja prioridad podría ser un problema visual en la página de perfil.

# Limitaciones de las Pruebas de Software

## Aspectos a considerar

**Cobertura total**: No es posible cubrir todos los posibles escenarios de uso en la mayoría de los sistemas complejos.

**Restricciones de tiempo y recursos**: La ejecución de pruebas extensivas puede ser costosa y consumir mucho tiempo.

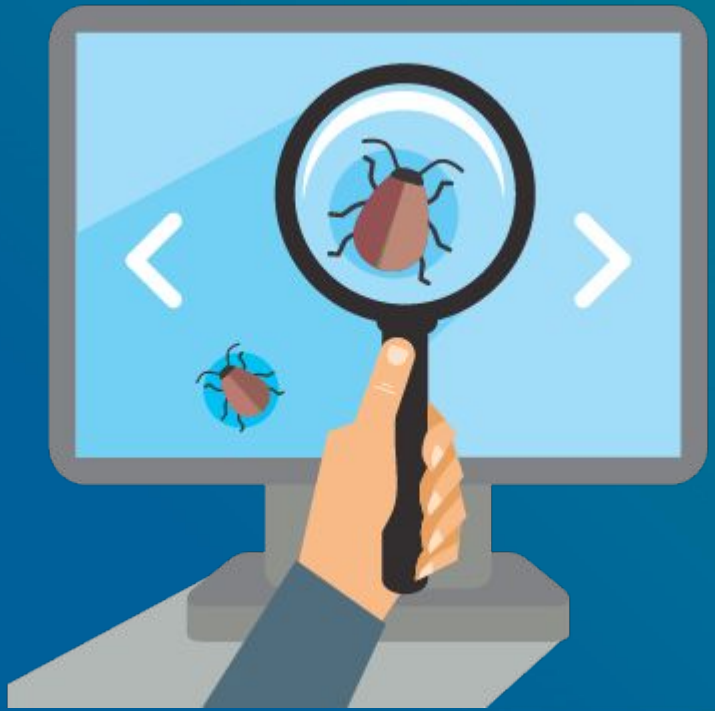


# Diseño y Ejecución de Casos de Prueba

# Diseño de Casos de Prueba – Introducción

**Estructura de un Caso de Prueba**:

**ID**: Identificador único del caso.
**Precondiciones**: Condiciones previas que deben cumplirse antes de la prueba.
**Pasos**: Instrucciones detalladas para ejecutar la prueba.
**Resultados esperados**: Lo que se espera obtener si el sistema funciona correctamente.



| Nombre de Proyecto: Xebee | | ID Caso de Prueba: CP-001 | |
|---|---|---|---|
| **Ambiente de Prueba:** Access/Web | | ID Historia de Usuario: HU-001 | |
| **Autor Caso de Prueba:** Mariam Rodríguez | | | |
| **Propósito** | | | |
| Verificar que los tipos de documentos cargados desde Access como Invoice for attorney se puedan visualizar en la aplicación web en el Tab correspondiente. | | | |
| **Descripción de las Acciones y/o condiciones para las Pruebas** | | | |
| **#** | **Acciones** | **Salida Esperada** | **Salida Obtenida** |
| 01 | Agregar un documento a un request desde Acess | El documento se carga como tipo solo para uso de Xebee | El documento se cargó con el tipo solo para uso de xebee |
| 02 | Cambiar el tipo de Documento a INVOICE FOR ATTORNEY desde Access | Actualizar el tipo de documento, “solo para uso interno” a Invoice for attorney | Se actualizó el tipo de documento, “solo para uso interno” a Invoice for attorney |
| 03 | Buscar request desde la Web | Debe existir en el listado de request. | Se encontró el request. |
| 04 | Consultar Tab Invoincing de la Web | Ver el documento que fue cargado desde Access de tipo Invoice for attorney | El documento etsaba en la pestaña tab invoincing |
| 05 | Consultar Tabs (Inf. General/documentación de soporte) de la Web | El documento de tipo Invoice for attorney no debe verse en ninguno de estos tab | El docuemnto no es visible desde estos tab. |
| **Resultados Obtenidos** | | | |
| **Resultado:** Aprobado | | | |
| **Seguimiento:** No Aplica | | **Severidad**: NO Aplica | |
| **Evidencia:** | | | |

## Ejemplo Completo de Caso de Prueba

**Caso de Prueba**: Función de añadir productos al carrito de compras en un e-commerce.

**Componentes**
**Precondiciones**: Usuario registrado, producto disponible en stock.
**Pasos detallados**: Seleccionar producto, añadir al carrito, revisar carrito.
**Resultados esperados**: El producto aparece en el carrito y el sistema muestra el total actualizado.

**Ejemplo de error y solución**: Producto no aparece en el carrito; se identifica el error en la base de datos de inventario.

## Beneficios de un Diseño Completo de Caso de Prueba

**Reducción de ambigüedades**: Mejora la claridad en el proceso de pruebas y reduce los errores de interpretación.

**Facilita la comunicación**: Los desarrolladores pueden comprender y corregir fácilmente los errores gracias a una descripción detallada.

**Ejemplo práctico**: En una app de salud, casos de prueba bien documentados permiten identificar rápidamente fallos en funcionalidades críticas, como el registro de medicamentos o signos vitales.

## Casos de Prueba Basados en Criterios de Aceptación

**Definición de Criterios de Aceptación**: Condiciones que el software debe cumplir para ser considerado funcional y aprobado.

**Ejemplo**: Un criterio de aceptación para una app bancaria podría ser "El usuario puede ver sus transacciones de los últimos seis meses".

**Impacto en la planificación de pruebas**: Define qué funcionalidades se deben verificar en las pruebas de aceptación.

# Tipos de Pruebas de Software

# Pruebas Unitarias

**Definición**: Son pruebas que verifican el correcto funcionamiento de unidades individuales del código (funciones, métodos, clases). Se realizan generalmente por los desarrolladores.

**Objetivo**: Detectar errores en la lógica de la programación antes de que el software se integre.

**Ejemplo**: Verificar que una función que calcula el IVA retorne el resultado correcto dado un valor de entrada. Por ejemplo, para un precio de 100, el IVA debería ser 21, y la prueba confirmaría que la salida es correcta.

# Ventajas de las Pruebas Unitarias

**Detección temprana de errores**: Permite encontrar errores de manera temprana en el ciclo de vida del desarrollo.

**Facilita la refactorización**: Los desarrolladores pueden realizar cambios en el código con confianza, sabiendo que si algo falla, se detectará rápidamente.

**Ejemplo de éxito**: En un proyecto de software, tras implementar pruebas unitarias, el equipo descubrió que un 30% del código estaba funcionando incorrectamente, lo que se corrigió antes de la integración.

# Pruebas de Integración

**Definición**: Se realizan para verificar la interacción entre diferentes módulos o componentes del software. Se aseguran de que funcionen correctamente en conjunto.

**Objetivo**: Identificar problemas que surgen de la integración de componentes que previamente funcionaban bien de forma aislada.

**Ejemplo**: En una aplicación de ecommerce, integrar la función de pago con el módulo de gestión de inventario para asegurar que el stock se actualiza correctamente después de una compra.

## Tipos de Pruebas de Integración

**Pruebas de integración de arriba hacia abajo(Top Down)**: Se prueba el sistema comenzando desde el nivel más alto, simulando las interacciones de los módulos inferiores.

**Pruebas de integración de abajo hacia arriba(Bottom Up)**: Se comienza desde los módulos de nivel inferior hacia arriba, integrando gradualmente.

**Pruebas de integración de Big Bang**: Todos los módulos se integran al mismo tiempo y se prueban juntos, lo cual puede ser riesgoso debido a la dificultad de aislar errores.

# Pruebas de Regresión

**Definición**: Se realizan para asegurar que los cambios en el código (nuevas características, corrección de errores) no hayan introducido nuevos fallos en funcionalidades existentes.

**Objetivo**: Verificar que el sistema siga funcionando correctamente después de que se han realizado cambios.

**Ejemplo**: Tras corregir un bug en la función de búsqueda, se ejecutan pruebas de regresión para verificar que la búsqueda de productos aún retorna resultados correctos y que no se han visto afectadas otras partes del sistema.

# Importancia de las Pruebas de Regresión

**Mantenimiento de la calidad**: Asegura que nuevas modificaciones no comprometan la estabilidad del software.

**Ejemplo de aplicación**: En un software de gestión de proyectos, cada vez que se añade una nueva función, se ejecutan pruebas de regresión sobre todas las funcionalidades existentes para garantizar que no se afecten entre sí.

# Pruebas de Aceptación

**Definición**: Realizadas para validar si el software cumple con los requisitos y expectativas del cliente o usuario final. Son generalmente realizadas por los testers o el equipo de calidad.

**Objetivo**: Asegurar que el sistema es funcional y que se satisface la necesidad del cliente.

**Ejemplo**: En una aplicación de gestión de tareas, un criterio de aceptación puede ser "El usuario debe poder marcar tareas como completadas", y se realizan pruebas para verificar este comportamiento.

# Tipos de Pruebas de Aceptación

**Pruebas de aceptación del usuario (UAT)**: Realizadas por los usuarios finales para validar el software en un entorno real.

**Pruebas de aceptación del sistema**: Verificación de que todo el sistema funciona como un todo, de acuerdo con los requisitos y especificaciones acordadas.

**Ejemplo**: Un cliente realiza pruebas en un software de facturación para verificar que todas las funciones operan según lo prometido antes de aceptar el software.

# Pruebas de humo (Smoke Testing)

**Definición**: Son pruebas preliminares realizadas para verificar si las funcionalidades básicas del software funcionan sin errores críticos, antes de proceder a pruebas más exhaustivas.

**Objetivo**: Identificar fallos graves que impidan el uso del software en su estado actual.

**Ejemplo**: En un sistema de gestión de clientes, realizar pruebas de smoke podría incluir verificar que los usuarios pueden iniciar sesión, que pueden crear nuevos registros y que la base de datos se carga correctamente.

## Beneficios del Smoke Testing

**Ahorro de tiempo y recursos**: Al realizar pruebas rápidas que verifican las funcionalidades críticas, se evita gastar tiempo en pruebas exhaustivas si el sistema está en un estado no funcional.

**Ejemplo de implementación**: En un ciclo de desarrollo ágil, después de cada integración, se ejecutan pruebas de smoke para asegurarse de que el sistema está listo para ser probado en profundidad.

# Estrategias y Herramientas de Pruebas

# Estrategias de Pruebas

**Definición de Estrategia de Pruebas**: Un plan detallado que describe el enfoque que se tomará para las pruebas en un proyecto específico.

**Elementos Clave**:

- Objetivos de prueba
- Tipos de pruebas a realizar
- Recursos y herramientas necesarias
- Roles y responsabilidades del equipo

**Ejemplo**: Una estrategia para una nueva aplicación de finanzas personales podría incluir pruebas unitarias exhaustivas, seguidas de pruebas de integración y pruebas de aceptación del usuario.

# Herramientas de Automatización de Pruebas

**Definición**: Software diseñado para facilitar la ejecución de pruebas automatizadas, reduciendo el esfuerzo manual y mejorando la cobertura de pruebas.

**Ejemplos de herramientas**:

- **Selenium**: Herramienta para pruebas automatizadas de aplicaciones web.
- **JUnit**: Framework para realizar pruebas unitarias en Java.
- **Postman**: Para probar APIs y servicios web.

**Beneficios**: Aumenta la eficiencia, permite la repetición de pruebas y ahorra tiempo en pruebas de regresión.

# Estrategias de Pruebas en Proyectos Ágiles

**Integración Continua (CI)**: Pruebas se realizan automáticamente cada vez que hay un cambio en el código, asegurando que las integraciones se mantienen estables.

**Iteración Rápida**: Probar en ciclos cortos permite feedback inmediato y ajustes rápidos en el desarrollo.

**Ejemplo**: Un equipo de desarrollo que implementa CI con Jenkins y prueba automáticamente el código cada vez que un desarrollador hace un 'commit'.

## Documentación de Pruebas

**Importancia de la Documentación**: Mantiene un registro detallado de todos los casos de prueba, resultados y defectos identificados.

**Elementos de la Documentación**:

- Casos de prueba
- Resultados de pruebas
- Informes de defectos

**Ejemplo de documentación efectiva**: Un sistema de gestión de pruebas que almacena todos los casos de prueba y resultados, facilitando la revisión y el análisis posterior.

# Gestión de Defectos (Bugs)

- **Proceso de Gestión de Defectos**: Identificación, clasificación, asignación y seguimiento de defectos desde su descubrimiento hasta su resolución.
- **Herramientas Comunes**: JIRA, Azure Devops, Mantis.
- **Ejemplo práctico**: Uso de JIRA para crear un ticket para un defecto encontrado en producción, asignándolo al desarrollador correspondiente y estableciendo una fecha de resolución.

## Métricas de Pruebas

**Definición**: Indicadores que permiten evaluar la eficacia y eficiencia del proceso de pruebas.

**Ejemplos de métricas**:

- Tasa de defectos: número de defectos encontrados por cada 1000 líneas de código.
- Cobertura de pruebas: porcentaje de código cubierto por las pruebas automatizadas.
- Tiempo medio de resolución: tiempo promedio que se tarda en solucionar un defecto reportado.

**Importancia**: Ayuda a identificar áreas de mejora en el proceso de pruebas y en la calidad del software.

## Pruebas de Seguridad

**Definición**: Pruebas diseñadas para identificar vulnerabilidades en un sistema y asegurar que los datos estén protegidos contra amenazas.

**Técnicas comunes**:

- **Pruebas de penetración**: Simulación de ataques para identificar debilidades en el sistema.
- **Análisis de vulnerabilidades**: Uso de herramientas para escanear el software en busca de fallas de seguridad.

**Ejemplo**: Un equipo realiza pruebas de penetración en una aplicación de banca en línea para asegurar que no se pueden acceder a los datos del cliente de manera no autorizada.

## Pruebas de Usabilidad

- **Definición**: Evaluación del software desde la perspectiva del usuario final, asegurando que sea fácil de usar y entender.

- **Métodos de evaluación**:
  - Pruebas con usuarios reales que interactúan con el sistema mientras son observados.
  - Cuestionarios y encuestas post-prueba para obtener feedback del usuario.

- **Ejemplo**: Realizar sesiones de prueba con usuarios de una nueva aplicación móvil, observando cómo navegan y qué problemas encuentran.

## Criterios de Éxito en las Pruebas

**Definición**: Parámetros que determinan si una prueba se considera exitosa o no.

**Ejemplos de criterios de éxito**:

- Todos los casos de prueba críticos pasan.
- No se identifican defectos críticos durante las pruebas de aceptación.
- El software cumple con todos los criterios de aceptación definidos.

**Importancia**: Establece un estándar claro que el software debe cumplir antes de su liberación.

## Pruebas de Carga y Rendimiento

**Definición**: Pruebas realizadas para evaluar cómo un sistema se comporta bajo una carga específica (número de usuarios concurrentes, volumen de datos).

**Objetivo**: Asegurarse de que el sistema pueda manejar el tráfico y las solicitudes esperadas sin fallos ni degradación del rendimiento.

**Ejemplo**: Realizar una prueba de carga en un sitio web de ventas en línea durante un evento de ventas para evaluar su capacidad de manejar múltiples usuarios realizando transacciones simultáneamente.

# Introducción al Desarrollo Guiado por Pruebas (TDD)

# Introducción al Desarrollo Guiado por Pruebas (TDD)

**Definición**: Una técnica de desarrollo de software donde las pruebas son escritas antes de implementar el código que las satisface.

**Ciclo de TDD**:

- **Red**: Escribir una prueba que falle porque la funcionalidad aún no se ha implementado.
- **Verde**: Escribir el mínimo código necesario para que la prueba pase.
- **Refactorizar**: Mejorar el código sin cambiar su funcionalidad, asegurando que las pruebas sigan pasando.

**Ejemplo**: Al desarrollar una nueva función para calcular el interés en un sistema bancario, primero se escribe la prueba que valida el cálculo correcto antes de escribir el código que realiza el cálculo.

## Beneficios del TDD

**Mayor calidad del código**: La escritura de pruebas primero obliga a los desarrolladores a pensar en la funcionalidad antes de escribir el código.

**Menor número de defectos**: Los errores se detectan de inmediato durante la fase de desarrollo, lo que reduce la necesidad de pruebas extensivas más adelante.

**Documentación efectiva**: Las pruebas sirven como documentación viva del comportamiento del sistema.

**Ejemplo de éxito**: Un equipo que adoptó TDD logró reducir el número de defectos encontrados en producción en un 40%.

## Desafíos del TDD

**Curva de aprendizaje**: Puede ser un cambio significativo en la mentalidad de desarrollo para aquellos que no están familiarizados con las pruebas.

**Tiempo adicional inicial**: Al principio, el TDD puede parecer que consume más tiempo, pero a largo plazo ahorra tiempo en pruebas y correcciones.

**Ejemplo práctico**: Un desarrollador experimenta dificultad al adaptarse a escribir pruebas antes de la implementación, pero con el tiempo se convierte en una segunda naturaleza.

## Criterios de Éxito en TDD

**Cobertura de pruebas**: El objetivo es que la mayor parte del código sea probado.

**Número de pruebas fallidas**: Idealmente, después de una fase de desarrollo, no deberían existir pruebas fallidas.

**Evolución del código**: El código debe poder ser modificado y expandido sin romper las pruebas existentes.

# Integración de TDD en Proyectos Ágiles

**Metodologías Ágiles y TDD**: La integración de TDD se complementa perfectamente con enfoques ágiles, donde la iteración rápida y el feedback constante son clave.

**Planificación de Iteraciones**: Las pruebas son parte del trabajo a realizar en cada iteración, garantizando que cada funcionalidad cumpla con los requisitos antes de avanzar.

**Ejemplo de uso**: Un equipo Scrum incorpora TDD en su backlog, asegurando que cada historia de usuario incluya pruebas antes de ser considerada "hecha".

## Herramientas Comunes para TDD

- **JUnit**: Para pruebas en aplicaciones Java.
- **NUnit**: Framework de pruebas para .NET.
- **Jest**: Para pruebas en aplicaciones JavaScript.

**Ejemplo de implementación**: Un equipo que utiliza JUnit para escribir pruebas unitarias en un proyecto Java, siguiendo el ciclo de TDD.

## Impacto del TDD en la Cultura de Desarrollo

**Colaboración**: Fomenta la colaboración entre desarrolladores y testers, promoviendo una mayor comunicación sobre los requisitos y expectativas.

**Mentalidad proactiva**: Los desarrolladores se vuelven proactivos en la identificación de problemas antes de que se conviertan en defectos en el software.

**Ejemplo de cultura positiva**: Un equipo que ha adoptado TDD tiene una tasa de satisfacción más alta, ya que los miembros sienten que su trabajo es de mayor calidad y cumplen mejor con las expectativas del cliente.

## Conclusión del TDD

**Beneficio a largo plazo**: Aunque la implementación inicial puede ser desafiante, los beneficios a largo plazo en términos de calidad y eficiencia hacen que valga la pena.

**Incorporación gradual**: Los equipos pueden comenzar a adoptar TDD en nuevas funcionalidades y luego extender su uso a todo el proyecto.

**Ejemplo final**: Una empresa de software que implementa TDD en un nuevo proyecto reporta una reducción significativa en el tiempo de prueba y defectos en producción.

# Para ampliar:

Testing - Fundamentos

**Certificación ISTQB**