

# PRUEBAS DE SOFTWARE

## Las pruebas de software brindan grandes beneficios:

- Ayuda a los negocios a mantenerse presentes en el mercado
- Desarrollo de aplicaciones o productos de software eficaces
- Asegura la confiabilidad y satisfacción del cliente
- Detecta defectos en etapas tempranas del desarrollo
- Reduce los costos del desarrollo y/o mantenimiento
- Asegura la calidad del producto

## Contexto en las pruebas del software:

- Todos cometemos errores
- Las pruebas dependen del contexto
- Se deben tomar en cuenta los riesgos

## Definición de error, defecto y falla con base en la IEEE 610



**Error:** es la acción humana que produce un resultado incorrecto

**Defecto:** es la imperfección o desperfecto en el código que puede ocasionar una desviación del componente o sistema en el desempeño de la función para la cual fue creado

**Falla:** es una desviación del componente o sistema en relación al resultado esperado

## Costo de los beneficios:

cuanto más rápido se identifique un error durante el ciclo de vida del desarrollo del software, menos será el costo para solucionarlo

## Causas de los defectos del software:

- Errores en las especificaciones, el diseño o la implementación
- Errores durante el uso del sistema
- Condiciones del ambiente inadecuadas
- Daño intencional
- Consecuencias potenciales de los errores tempranos, defectos y fallas

FASE	COSTO RELATIVO POR CORREGIR
DEFINICIÓN	\$1
DISEÑO DE ALTO NIVEL	\$2
DISEÑO DE BAJO NIVEL	\$5
CÓDIGO	\$10
PRUEBAS UNITARIAS	\$15
PRUEBAS DE INTEGRACIÓN	\$22
PRUEBAS DEL SISTEMA	\$50
POST-LIBERACIÓN	\$100+

Como se puede observar, detectar errores en las fases más tempranas en el ciclo de vida de desarrollo de software, reduce significativamente los costos, de ahí la importancia de la implementación de pruebas desde el inicio del proyecto

## Algunas definiciones:

- La norma ISO/IEC 9126 define la calidad como “grado mediante el cual un componente, sistema o proceso reúne requerimientos específicos sobre las necesidades o expectativas del cliente o usuario final”
- La norma IEEE 610 define software como “programa de computadora, procedimientos, documentación y datos a la operación del sistema de computadora”

- La norma ISO/IEC 9126 define la calidad de software como “totalidad de la funcionalidad y prestaciones de un producto software relacionados con sus capacidades de satisfacer las necesidades explícitas o implícitas”

### Tipos de aseguramiento de calidad:

- **Aseguramiento de calidad constructivo:** su objetivo principal es la prevención de defectos empleando métodos basados en la ingeniería del software
- **Aseguramiento de calidad analítico:** su objetivo es la detección de defectos, mediante técnicas de identificación de estos para incrementar la calidad del software

### Verificación y validación

Verificación:	Validación:
<ul style="list-style-type: none"> <li>• ¿Se construyo correctamente el producto?</li> <li>• ¿cumplió la especificación de requerimiento?</li> <li>• ¿El producto refleja adecuadamente los requerimientos especificados?</li> </ul>	<ul style="list-style-type: none"> <li>• ¿Se construyo el producto correcto?</li> <li>• ¿cumplió las necesidades operacionales?</li> <li>• ¿cumplió con el uso para el cual estaba pensado?</li> </ul>

### Principios:

	<b>Primer principio:</b> Las pruebas de software muestran la presencia de defectos, pero no puede mostrar la no presencia de defectos. Las pruebas reducen la probabilidad de defectos no descubiertos en el software, esto no es garantía de perfección. El objetivo de las pruebas es encontrar defectos
	<b>Segundo principio:</b> Realizar pruebas de todas sus combinaciones de entradas y precondiciones no es factible, con excepción de casos triviales. En vez de pruebas exhaustivas, es mejor realizar un análisis de riesgo y de las prioridades para enfocar sus esfuerzos en ellas. Las pruebas exhaustivas no son prácticas, además de ser muy costosas
	<b>Tercer principio:</b> El equipo de pruebas de software se debe involucrar en etapas tempranas del ciclo de vida del software y debe basarse en objetivos definidos. Al equipo de pruebas se le involucra casi al final de las fases, de modo que el trabajo que se realiza es correctivo y no preventivo
	<b>Cuarto principio:</b> Un pequeño número de módulos, generalmente los mas complejos, contienen una gran parte de los defectos que son responsable de la mayoría de las fallas, sobre todo si son críticos o se encuentran en fases complejas de la aplicación
	<b>Quinto principio:</b> Si durante un ciclo de vida de software, las mismas pruebas se repitieran una y otra vez, en algún momento estas dejarían de encontrar nuevos defectos. Por tal motivo, es recomendable generar casos de pruebas más robustos. Este principio explica por qué no siempre la automatización de pruebas es lo más optimo y por qué no debería regir el proceso
	<b>Sexto principio:</b> Las pruebas se realizan de manera diferente dependiendo del contexto del sistema y todo lo que este conlleve



### Séptimo principio:

Encontrar y corregir defectos no ayuda si el sistema que se ha desarrollado no cubre las necesidades ni las expectativas de los clientes expresados en la especificación de requerimiento

### Ciclo de vida del software:



### Proceso de pruebas:

Permite definir las etapas desde la planeación, el diseño, ejecución, evaluación y cierre. La decisión sobre el nivel de formalidad del proceso depende del sistema, del contexto del software y del nivel de riesgo asociado

- Planeación y control de pruebas
- Análisis y diseño de pruebas
- Implementación y ejecución de las pruebas
- Evaluación del criterio de salida y reporte
- Actividades de cierre de pruebas



### Primera etapa: Planeación y control de pruebas

- Tareas principales de la planeación:
  - ✓ Determinar el alcance y riesgo de las pruebas
  - ✓ Determinar el enfoque de las pruebas
  - ✓ Implementar la política y estrategia de las pruebas
  - ✓ Determinar los recursos requeridos
  - ✓ Definir el criterio de salida
- Actividad continua que monitorea y compara el progreso actual con el planeado
- Las tareas principales del control de pruebas son:
  - ✓ Medición y análisis de resultados (pruebas pasadas vs. fallidas)
  - ✓ Cobertura de pruebas y criterio de salida
  - ✓ Tomar acciones correctivas cuando sea requerido
  - ✓ Tomar decisiones con base en las mediciones e información reunida durante las pruebas

### Segunda etapa: Análisis y diseño de pruebas

- Evaluar si el requerimiento se presta para ser probado, es decir, ¿es posible igualar las condiciones de ambiente para evaluar el sistema y sus configuraciones?
- Empleo de técnicas para diseñar y priorizar pruebas

- Identificar y priorizar las condiciones de prueba, con base en análisis, especificación, comportamiento y estructura
- Identificar los datos empleados para la ejecución de las pruebas, dependiendo de si son pruebas de funcionalidad de desempeño, etc.

### **Tercera etapa: Implementación y ejecución de las pruebas**

- Desarrollar y priorizar los casos de prueba
- Generar los datos de entrada correspondientes
- Crear una suite de pruebas para una ejecución eficiente del módulo o componente
- Montar y verificar el ambiente de pruebas
- Ejecutar la suite de pruebas con base en el procedimiento de pruebas
- Generar registros de las pruebas ejecutadas
- Comparar el resultado actual con el respaldo
- Reportar las discrepancias encontradas como incidencias
- Realizar pruebas de re ejecución o confirmación, así como pruebas de regresión

### **Cuarta etapa: Evaluación del criterio de salida y reporte**

- Evaluar los resultados de la ejecución de las pruebas contra los objetivos definidos
- Verificar los logs contra el criterio de salida especificado en el plan de pruebas
- Evaluar si se necesitan más pruebas
- Redactar un reporte con el resumen de las pruebas para los implicados, indicando que pruebas se ejecutaron y cuáles fueron los resultados

### **Quinta etapa: Actividades de cierre de pruebas**

- Verificar que se han entregado los artefactos especificados en el plan, como casos de pruebas, scripts de pruebas y reporte de incidencias
- Comprobar que las incidencias reportadas han sido resueltas
- Analizar las lecciones aprendidas para futuros proyectos

### **Independencia de las pruebas**

- Las pruebas las realiza otro desarrollador del mismo equipo de desarrollo
- Las pruebas las realiza un tester perteneciente a un equipo de pruebas de la empresa
- Las pruebas las realiza un tester perteneciente a una organización ajena
- Reconocer las ventajas
- Reconocer las desventajas

### **Psicología de las pruebas**

La psicología de las pruebas involucra los siguientes aspectos:

- La comparación de la mentalidad del probador y el desarrollador
- El grado de independencia evita el sesgo de autor y es, a menudo, más eficaz en la búsqueda de defectos y fracasos
- El equilibrio entre la auto-prueba y las pruebas independientes
- La comunicación y retroalimentación claro y cortés de los defectos entre probador y desarrollador



### **Código de ética**

El equipo de pruebas debe tener en todo momento una actitud ética. Es responsabilidad del líder de pruebas promover el siguiente código de ética:

- Publico: los probadores deben actuar con base en los intereses del cliente, particularmente donde la seguridad y confiabilidad de los sistemas es crítica
- No utilizar la información confidencial para fines personales



- Asegurar que los productos de software cumplan con los más altos estándares de calidad
- Los probadores mantendrán su integridad e independencia en su juicio profesional
- Como líder de pruebas, promover un enfoque completamente ético
- Promover la integración y reputación

### El modelo de cascada (SLCD)

El modelo en cascada requiere que cada una de las etapas previas haya sido concluida antes de iniciar la siguiente:

- Facilita la planeación y estimación de tiempos
- La dificultad para documentar las solicitudes de cambio

Los costos de los defectos en el ciclo de vida de desarrollo de software



### Modelo V

Surge como una respuesta a las deficiencias de su predecesor, el modelo en cascada. Ilustra como las actividades de pruebas, tales como la validación y la verificación, se pueden integrar en cada fase del ciclo de vida. Existen diversas variantes del modelo V, pero hay cuatro etapas en común:



**Pruebas de componente:** búsqueda de defectos y verificación en los componentes de software

**Pruebas de integración:** pruebas a interfaces e interacción entre los distintos componentes

**Pruebas de sistema:** referente al comportamiento del sistema en su totalidad, verifican el sistema contra los requerimientos

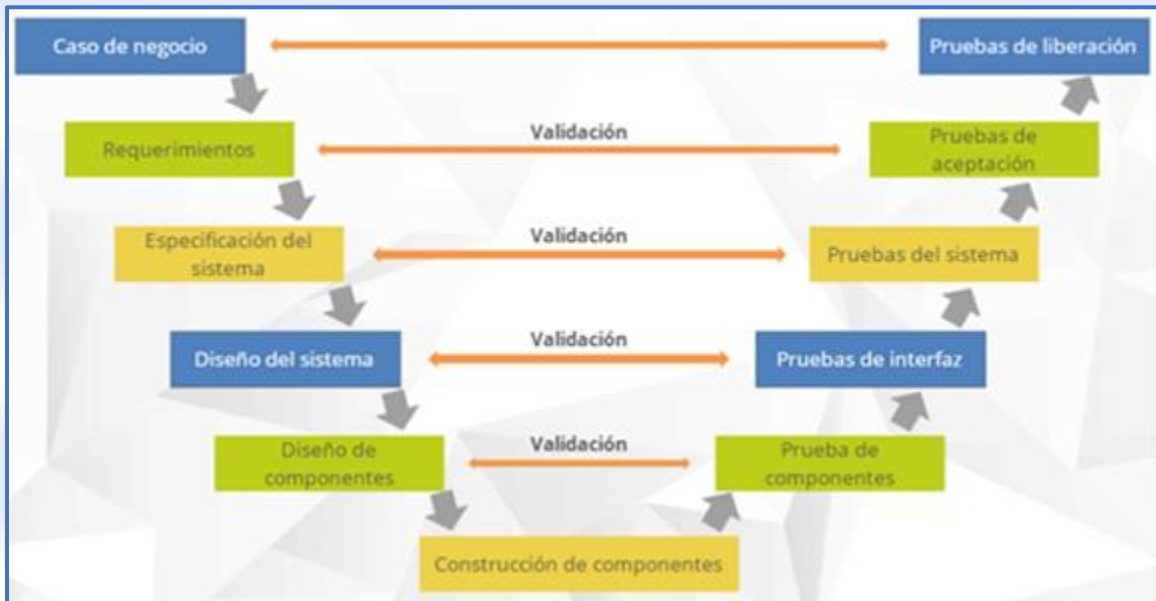
**Pruebas de aceptación:** pruebas de validación con respecto a las necesidades del cliente, requerimientos y proceso de negocio para determinar la aceptación del sistema por el usuario final

### Ciclos de vida iterativos e incrementales

Los ciclos de vida iterativos e incrementales consisten en la interacción continua de varias fases en un mismo desarrollo o proyecto. Existen diferentes variantes de este tipo de modelos. Las pruebas de regresión se incrementan de forma significativa para este tipo de desarrollos. Algunos ejemplos de los modelos de desarrollo iterativo e incrementales:

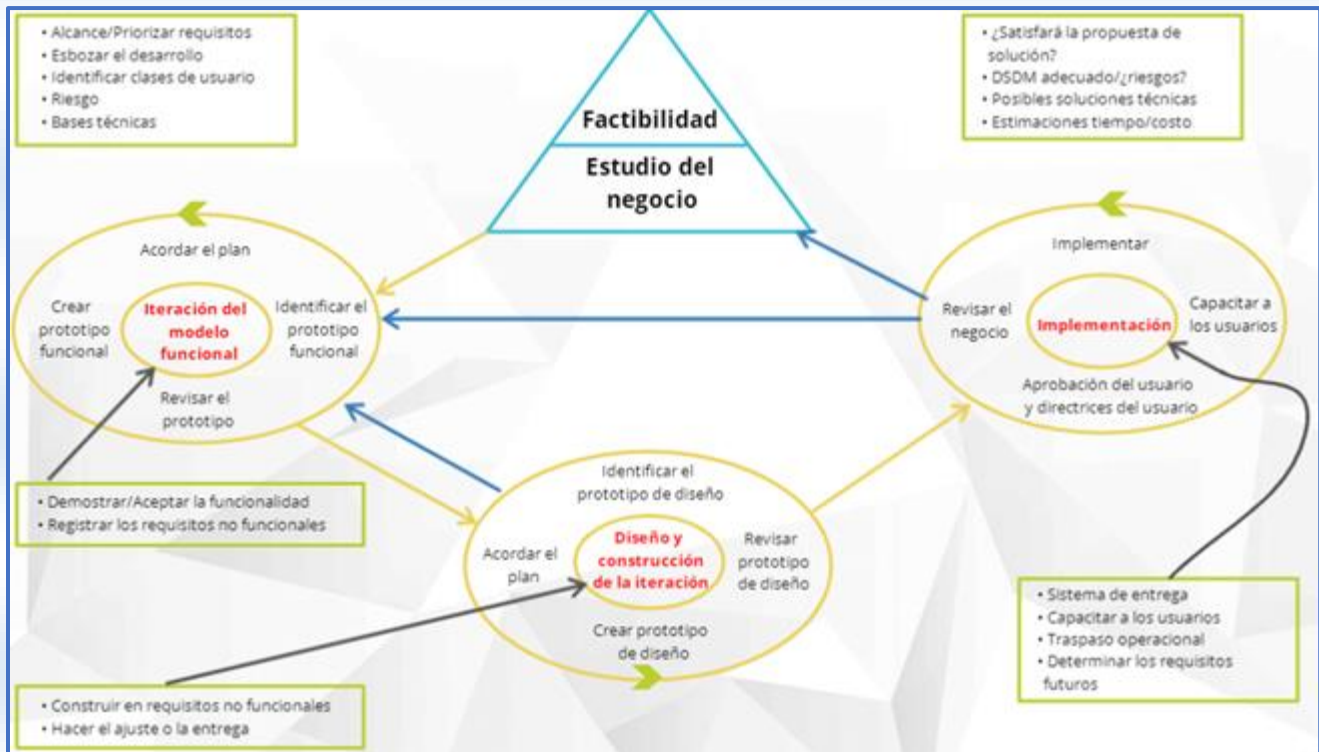


- Prototyping
- RAD
- Scrum
- RUP
- Extreme Programming
- Dynamic System Development Methodology



### DSDM (Dynamics System Development methodology)

El modelo DSDM es similar a RAD, pero refinado, es decir, permite un mayor control al detener y evaluar los procesos cuando estos se han salido del control; asimismo, requiere una estricta gestión de la configuración para la implementación de cambios rápidos.



### Extreme Programming

El extreme programming es una metodología ágil que promueve la generación de historias de usuario.

Promueve la compartición de código, sostiene que los casos de prueba deben ser escritos antes que el código y sugiere que se implemente una solución simple para dar solución al cliente



## RAD (Rapid Application Development Model)



El model RAD consiste en el desarrollo en paralelo de componentes como mini proyectos, esto permite al usuario tener contacto con la aplicación y así proveer la retroalimentación. Implica una evaluación temprana del riesgo

tecnológico y una respuesta rápida a cambios durante las etapas de desarrollo

## Tipos de pruebas

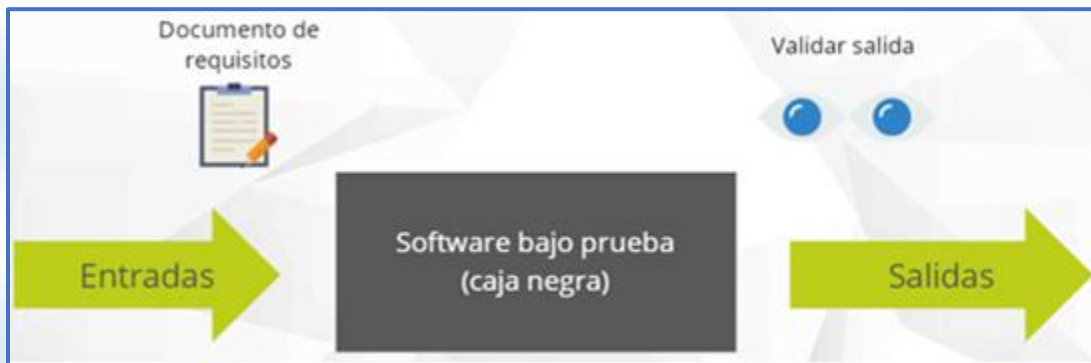
NIVEL	TIPOS DE PRUEBAS QUE SE PUEDEN APLICAR
Pruebas de componentes	Pruebas estructurales
Pruebas de integración	Pruebas funcionales
Pruebas de sistema	Pruebas funcionales y no funcionales (volumetría, cargas, estrés, etc.)
Pruebas de aceptación	Pruebas funcionales, confirmación y regresión

## Pruebas funcionales

- Son pruebas que están descritas generalmente en un requerimiento o caso de uso
- Se pueden llevar a cabo en todos los niveles de pruebas
- También se conocen como pruebas de caja negra, aunque esto no es completamente cierto, pues las pruebas de caja negra también incluyen algunas pruebas no funcionales
- Las pruebas funcionales se basan en la norma ISO 9126, que se enfoca en adecuación, interoperabilidad, seguridad, precisión y cumplimiento
- Las técnicas de pruebas funcionales generalmente se basan en especificación, sin embargo, las pruebas basadas en experiencia pueden ser bastantes útiles
- Las condiciones y los casos de prueba se derivan de la funcionalidad del componente o sistema

## Pruebas de caja negra

Es una técnica de pruebas de software en la cual la funcionalidad se verifica sin tomar en cuenta la estructura interna de código, detalles de implementación o escenarios de ejecución en el software



### Pruebas no funcionales

- Se llevan a cabo en todos los niveles de pruebas
- Se enfocan en atributos de calidad o no funcionales
- Incluyen desempeño, carga, estrés, usabilidad, mantenibilidad, confiabilidad, portabilidad, etc.
- Este tipo de pruebas responden a la pregunta ¿qué tan bien trabaja el sistema?



### Pruebas estructurales

- Son conocidas como pruebas de caja blanca
- Son utilizadas para conocer la cobertura de las pruebas
- Ocurren a cualquier nivel de pruebas, sin embargo, son mas aplicadas en pruebas de componentes e integración para conocer el porcentaje de pruebas
- Para estas pruebas se utiliza herramientas especializadas
- Este tipo de pruebas se utilizan cuando se conoce la codificación del sistema, se busca código ambiguo, código muerto, parte de código no utilizada durante la ejecución del sistema

### Pruebas de caja blanca

se centran en los detalles procedimentales del software, por lo que su diseño está ligado al código fuente. El ingeniero de pruebas escoge distintos valores de entrada para examinar cada uno de los flujos de ejecución del programa y cerciorarse de que se devuelven los valores de salida adecuados.



### Pruebas de mantenimiento

Existen sistemas que llevan años o décadas en servicio. Durante ese tiempo, el sistema ha sido modificado, corregido o extendido. Las pruebas que han sido ejecutadas durante estas etapas se conocen como pruebas de mantenimiento. Las pruebas de mantenimiento involucran dos partes:

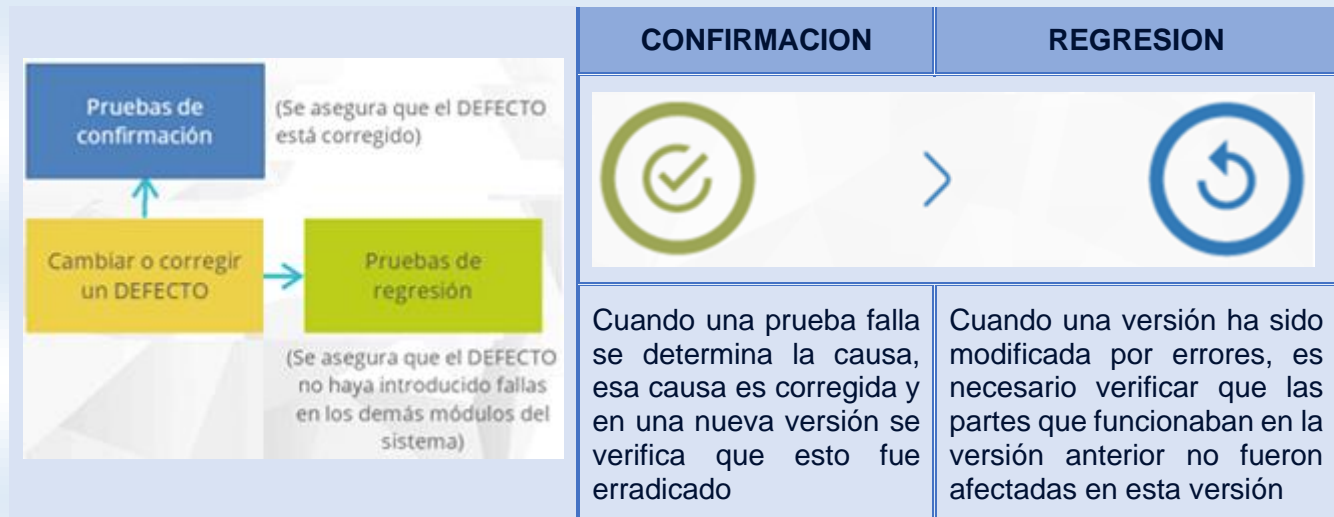
Probar los cambios	Pruebas de regresión para detectar si el sistema fue afectado
--------------------	---

Se requiere un análisis de impacto en conjunto con los implicados para determinar que partes del sistema pueden ser afectadas de manera no intencional y requerirán pruebas de regresión.



Un análisis de riesgos es útil en esta etapa

## Pruebas de confirmación y regresión



## Niveles de prueba

- Tienen objetivos diferentes
- Requiere condiciones particulares para realizar pruebas
- Involucra diferentes áreas
- Necesita diferentes habilidades o expertise
- Permite delimitar alcances

## ISTQB (International Software Testing Qualifications Board)



## Pruebas de componentes

Las pruebas de componentes son también conocidas como unitarias, modulares o de programa. En algunos casos, estas pruebas requieren acceso al código (pruebas unitarias). En este nivel de pruebas, los defectos son corregidos tan pronto como son identificados

Stub	Driver
Un <b>stub</b> es invocado por un componente de software para ser probado.	Un <b>driver</b> invoca a un componente de software que está siendo probado.

Frecuentemente se requiere Stubs y Drivers para simular una interfaz que aun no ha sido desarrollada

Este tipo de pruebas puede incluir pruebas de funcionalidad, así como pruebas no funcionales, tales como: desempeño, robustez, carga, estrés, entre otras.

Un enfoque de pruebas de componentes en Extreme Programming (XP) es TDD (desarrollo dirigido a pruebas). Este enfoque es altamente iterativo y se basa en ciclos que inician con la definición de casos de pruebas, construcción, integración de pequeños módulos y ejecución de los componentes de pruebas



**Pruebas unitarias:** son la parte mas pequeña del software. En el caso de programas como java y C++, se conocen como clases: en C se conoce como función; en lenguajes menos estructurados, como Basic y Cobol, esa unidad puede ser representada por el programa completo

### Pruebas de integración

Las pruebas de integración se utilizan para verificar la funcionalidad o interacción entre dos o mas componentes o interfaces de un sistema. Existen mas de un nivel de pruebas de integración:

- **Pruebas de integración de componente:** entre elementos de un mismo sistema
- **Pruebas de integración de sistemas:** donde se prueba la integración entre diferentes sistemas, en este tipo de pruebas, se debe modificar solamente uno de los sistemas

La mejor elección es iniciar la integración con aquellos módulos o componentes que pueden generar mayor problema o se estima involucren un mayor riesgo. Idealmente, los testers deben entender la arquitectura e influir en la planeación para la integración de los componentes

Estas pruebas se llevan a cabo por desarrolladores o testers expertos en integración. Los testers se concentran únicamente en las pruebas de integración, por ejemplo, si se esta probando la integración del componente A y el componente B. están interesados en la comunicación entre ambos componentes y no en la funcionalidad de cada uno de ellos.

### Técnicas de integración

- **Técnicas de integración Big Bang:** todos los componentes del sistema son integrados de manera simultánea, después, se realizan pruebas de integración de sistema en su totalidad

VENTAJAS	DESVENTAJAS
El desarrollo está concluido antes de iniciar las pruebas de integración. No se requiere simular componentes que aun no han sido contruidos (Stubs/drivers)	Se requiere de mucho tiempo y es complicado identificar las causas de una falla, sobre todo en integraciones tardías

- **Técnicas de integración incremental:** los componentes se van agregando uno a uno y las pruebas se llevan a cabo después de cada incremento.

VENTAJAS	DESVENTAJAS
Los defectos son identificados en etapas tempranas en pequeñas integraciones y es relativamente sencillo identificar las causas	Se invierte tiempo en el desarrollo de drivers y Stubs para que sean utilizados en las pruebas

Dependiendo de la arquitectura, existen tres tipos de integraciones incrementales:

- **Top-Down:** parte de arriba hacia abajo, siguiendo la estructura arquitectónica, por ejemplo, partiendo de la GUI. Los componentes son sustituidos por Stubs

- **Bottom-Up:** a diferencia de la integración Top-Down, tiene lugar partiendo de un bajo nivel y siguiendo el ejemplo anterior hacia la GUI
- **Funcional Incremental:** las pruebas tienen lugar con base en la funcionalidad, es decir, basadas en el documento o especificación funcional

### Pruebas de sistemas:

Las pruebas de sistemas se centran en el comportamiento del sistema en su totalidad, como se definió



en el alcance del desarrollo del proyecto. Pueden incluir pruebas basadas en el riesgo, especificación de requerimiento, proceso de negocio, caso de uso o cualquier otro documento de alto nivel del comportamiento del sistema. Son frecuentes en etapas finales del desarrollo para verificar que el sistema a liberar cumple con la especificación y su propósito es encontrar tantos defectos como sea posible.

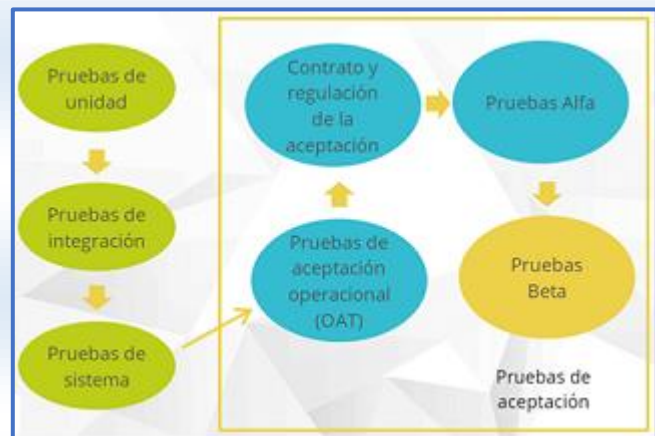
Pueden requerir tanto pruebas funcionales como pruebas no funcionales. Las pruebas no funcionales típicas requeridas son desempeño y confiabilidad. Para las pruebas funcionales, se requiere utilizar la técnica de pruebas de caja negra más adecuada. Este tipo de pruebas requieren un ambiente controlado como control de versiones, sets de pruebas, datos de entrada, esto para minimizar el riesgo.

### Pruebas de aceptación de usuario

Una vez que la organización ha ejecutado pruebas de sistema y ha corregido la mayor parte de los defectos, se está listo para liberar el sistema al usuario para pruebas de aceptación. Este tipo de pruebas debe responder preguntas tales como:

- ¿puede el sistema ser liberado a producción?
- ¿Cuáles son los riesgos más relevantes?
- ¿el desarrollo ha reunido los requerimientos?

Las pruebas de aceptación son responsabilidad del cliente o usuario final y requieren un ambiente similar al productivo. Su objetivo es crear confianza en el sistema, determinar si el sistema es acorde al propósito. Encontrar defectos no es el propósito principal de estas pruebas. En algunas organizaciones se requieren pruebas OAT (aceptación operacional), como respaldo/restauración, recuperación de desastres, tareas de mantenimiento y vulnerabilidad de la seguridad



Otro tipo de pruebas de aceptación que existen son:




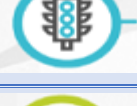


- **Pruebas de aceptación de cumplimiento o regulación:** son pruebas ejecutadas contra regulaciones, generalmente gubernamentales o regulaciones legales
- **Pruebas de aceptación de contrato:** son pruebas ejecutadas contra lo especificado en el contrato

Es frecuente que el software comercial (COTS) se someta a dos etapas de pruebas de aceptación:





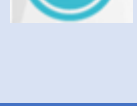
- **Pruebas alfa:** pruebas de aceptación realizadas en el lugar o instalaciones donde fue desarrollado el software
- **Pruebas beta:** pruebas de aceptación realizadas en el lugar o instalaciones donde se utilizará la aplicación bajo condiciones reales de trabajo. El cliente enviara registro de las incidencias que pudieran encontrarse para su solución

## Resumen

### Modelos de desarrollo de software

	<b>Modelo cascada (SLCD)</b> requiere que cada una de las etapas previas haya sido concluida antes de iniciar la siguiente
	<b>Modelo V</b> ilustra como las actividades de pruebas tales como la validación y la verificación se pueden integrar en cada fase del ciclo de vida
	<b>Ciclo de vida iterativos e incrementales</b> Consiste en la interacción continua de varias fases en un mismo desarrollo o proyecto
	<b>RAD (Rapid Application Development Model)</b> Desarrollo en paralelo de componentes como mini proyectos. Permite al usuario tener contacto con la aplicación y así proveer retroalimentación
	<b>DSDM (Dynamics System Development methodology)</b> Permite un mayor control al detener y evaluar los procesos cuando estos se han salido de control
	<b>Extreme programming</b> <ul style="list-style-type: none"><li>• Promueve la generación de historias de usuario</li><li>• Promueve la compartición de código</li><li>• Sostiene que los casos de prueba deber ser escritos antes que el código</li><li>• Sugiere que se implemente una solución simple al cliente</li></ul>

### Tipos de pruebas

	<b>Pruebas funcionales</b> Requiere que cada una de las etapas previas haya sido concluida antes de iniciar la siguiente
	<b>Pruebas no funcionales</b> Se llevan a cabo en todos los niveles de pruebas. Incluyen desempeño, carga, estrés, usabilidad, mantenibilidad, confiabilidad, portabilidad, etc.
	<b>Pruebas estructurales</b> Se conocen como pruebas de caja blanca y son utilizadas para conocer la cobertura de las pruebas
	<b>Pruebas de confirmación y regresión</b> <b>Confirmación:</b> cuando un aprueba falla se determina la causa, esa causa es corregida y en una nueva versión se verifica que esto fue erradicado <b>Regresión:</b> cuando una versión ha sido modificada por errores, es necesario verificar que las partes que funcionaban en la versión anterior no fueron afectadas en esta versión
	<b>Pruebas de mantenimiento</b> Son pruebas que se han realizado a través del tiempo cuando el sistema se ha modificado

### Niveles de pruebas

- **Pruebas de componentes:** pueden incluir pruebas de funcionalidad, así como pruebas no funcionales, tales como: desempeño, robustez, carga, estrés, entre otras



- **Pruebas de integración:** son pruebas para verificar la funcionalidad o interacción entre dos o mas componentes o interfaces de un sistema
- **Pruebas de sistema:** se centran en el comportamiento del sistema en su totalidad, como se definió en el alcance del desarrollo del proyecto
- **Pruebas de aceptación de usuario:** las pruebas de aceptación son responsabilidad del cliente o usuario final, su objetivo es crear confianza en el sistema y determinar en el sistema es acorde al propósito

## Técnicas estáticas

- Las pruebas estáticas inician en etapas tempranas del ciclo de vida del desarrollo de software, su objetivo es la detención de defectos en etapas temprana, reduciendo el re trabajo
- El re-trabajo es reducido de manera sustancial y la productividad en el desarrollo se incrementa.
- Ofrece un beneficio adicional, que es un intercambio entre los participantes.
- Contribuye a la identificación de defectos

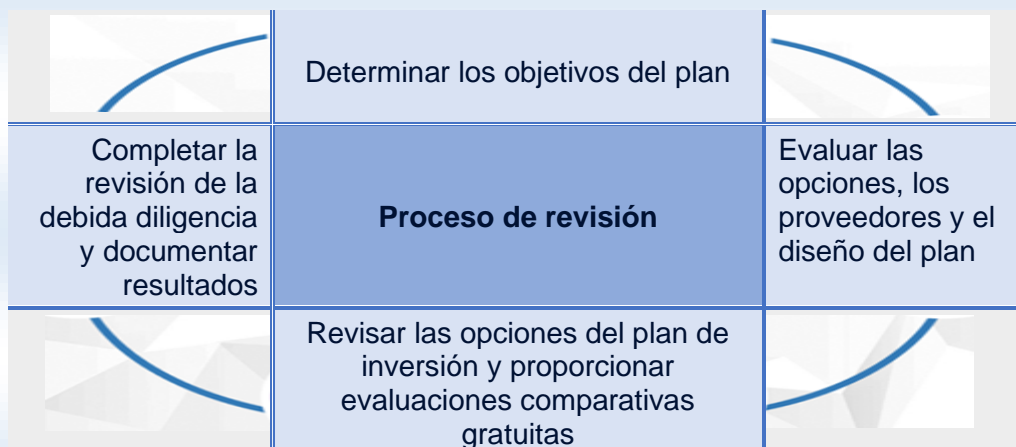


## Roles

	<b>El revisor:</b> identifica y describe las anomalías en los productos o proyectos. Pueden ser elegidos para representar los diferentes puntos de vista y roles
	<b>El moderador:</b> persona principal responsable para inspección u otro proceso de revisión
	<b>El escribano:</b> registra cada defecto mencionado y las sugerencias de mejora. Debe garantizar que el formulario de registro sea legible y comprensible

## El proceso de revisión

- Las revisiones van desde revisiones informales hasta revisiones muy formales
- Las revisiones son quizá el proceso mejor documentado, pero no el único
- Las revisiones informales se aplican en repetidas ocasiones en etapas tempranas del ciclo de vida del proyecto



## Tipos de revisiones



- **Inspección:** tipo de revisión por pares, se basa en la inspección visual de documentos para detectar defectos. Es la técnica mas formal y se basa en procedimientos documentados
- **Revisión técnica:** actividad de debate de grupo que se enfoca en lograr un consenso sobre el método técnico a utilizar
- **Revisión guiada (tutorial):** actividad de debate que se enfoca en lograr un consenso sobre el método técnico a utilizar
- **Revisión informal:** revisión que no se basa en ningún procedimiento formal (documentado)

## ¿Qué tipos de defectos estamos buscando?

- **Consistencia:** cuando el producto bajo revisión (work product) no cumple con las especificaciones
- **Omisión:** cuando el producto bajo revisión no incluye algún elemento especificado
- **Reincidencia:** cuando el producto bajo revisión tiene un defecto que se encontró previamente durante la revisión o en otra revisión
- **Derivado:** cuando no existía el defecto, pero se deriva de una corrección a otro defecto previamente detectado
- **Mejoras:** cuando se trata de una sugerencia de mejora para el producto bajo revisión con el fin de añadir mejoras relacionadas con las buenas prácticas técnicas



## Proceso de revisión



## Planeación

- La planeación inicia con una solicitud de revisión por parte del autor o moderador.
- Se realiza un chequeo inicial del documento para asegurar que los participantes no malgastaran el tiempo en un documento que no esta listo para ser revisado.
- El moderador se encarga de definir fechas, tiempos, lugar y notificar a los participantes.

Algunos criterios de entrada son:

- Una verificación del documento por parte del moderador
- Documentos con números de línea
- El autor debe estar consiente de las criticas que recibirá, así como sentir confianza con la calidad del documento
- Usar alguna herramienta de corrección ortográfica

Enfoque para la revisión de un documento

- **Enfoque de alto nivel:** el diseño cumple con el requerimiento
- **Enfoque en documentos relacionados al mismo nivel:** interfaces entre funciones de software
- **Enfoque estándar:** consistencia interna, claridad, convención de nombres y template
- **Enfoque en uso:** capacidad de ser probado o mantenibilidad

## Inicio

- En el inicio, el objetivo es poner a los participantes en contexto respecto al documento a revisar y el tiempo destinado para esto.
- En esta fase, los participantes reciben una pequeña introducción de los objetivos y documentos a revisar y la relación con otros documentos.
- Asignación de roles, porcentajes de revisión, paginas a revisar, etc.

## Preparación

- En la preparación, los participantes trabajan de manera individual en la revisión del documento, todos los errores encontrados se documentan.
- Los errores ortográficos son documentados, pero no se mencionan en la junta de revisión.
- Se puede utilizar una lista de chequeo para hacer la revisión mas eficiente.
- Un factor crítico es el número de páginas revisadas por hora, conocido como tasa de revisión.

## Reuniones de revisión

<b>Registro de incidencias</b>	<ul style="list-style-type: none"> <li>• Se documentan por página y participante, ya sea el autor o un escribano.</li> <li>• Las incidencias que requieren discusión se documentan y se presentan en la fase de discusión.</li> <li>• Se clasifican por severidad: Critico, mayor y menor</li> <li>• El participante que identificó el error propone la severidad.</li> </ul>
<b>Fase de discusión</b>	<ul style="list-style-type: none"> <li>• En las revisiones formales se hace la distinción entre la etapa de registro y discusión.</li> <li>• Los participantes participan en la discusión aportando argumentos.</li> <li>• Los participantes que no sean requeridos en la revisión, se pueden retirar.</li> <li>• El moderador se asegura de que todos los issues sean revisados</li> </ul>
<b>Fase de decisión</b>	<ul style="list-style-type: none"> <li>• Si el número de errores excede un límite, el documento deberá revisarse nuevamente</li> </ul>

## Re trabajo

- En la fase de re trabajo, con base en los errores encontrados, el autor mejorara el documento. No todos los defectos se corregirán, es responsabilidad del autor juzgar si el defecto se corrige
- Los cambios realizados en el documento deben ser fáciles de identificar durante la etapa de seguimiento.

## Seguimiento

- En el seguimiento, el moderador es responsable de garantizar que las acciones tomadas son satisfactorias.
- Verifica que el autor ha tomado acciones en todos los defectos conocidos
- Si se decidió que todos los participantes verificaran el documento actualizado, el moderador convocara a la reunión
- En reuniones muy formales, el moderador verifica el cumplimiento contra el criterio de salida

El análisis estático puede detectar enlaces rotos, por ejemplo:

- Referencias a variables con valores no definidos

- Lógica faltante o incorrecta, mal uso de operadores (igualdad en vez de asignación)
- Interfaces inconsistentes entre módulos y/o componentes
- Complejidad excesiva
- Variables que nunca son utilizadas
- Violación o estándares de programación
- Código inalcanzable
- Vulnerabilidad de la seguridad

## Técnicas de diseño pruebas

Técnicas de pruebas estáticas	Prueban cualquier forma de documento, incluyendo el código fuente
Técnicas de pruebas basadas en especificación, caja negra	Se basan en las especificaciones del sistema, pero no en su estructura interna
Técnicas de pruebas basadas en estructura, caja blanca	Se basan en el análisis de la estructura interna de un sistema
Técnicas de pruebas basadas en experiencia	Las pruebas se basan en el conocimiento e intuición del probador

La **especificación de diseño de pruebas** es un documento que detalla las condiciones de pruebas (cobertura de elementos) para un elemento de pruebas, el detalle del enfoque de pruebas y los casos de pruebas de alto nivel asociado.

IEEE 829: plantilla de especificación de diseño de pruebas:

Id del documento	Id de cada prueba
Características a ser probadas	Criterio pasado/fallido
Enfoque detallado	

La **especificación de casos de pruebas** es un documento que detalla el conjunto de casos de pruebas (objetivos, entradas, salidas, acciones, resultados y precondiciones) para un elemento de pruebas.

IEEE 829: plantilla de especificación de casos de pruebas:

Id del documento	Necesidades de ambiente
Elemento bajo prueba	Procedimientos especiales
Especificación de entrada	Dependencias existentes
Especificación de salida	

La **especificación de procedimiento de pruebas** es un documento que detalla una secuencia de acciones para la ejecución de una prueba. También conocido como script de pruebas manual o automático.

IEEE 829: plantilla de especificación de procedimiento de pruebas:

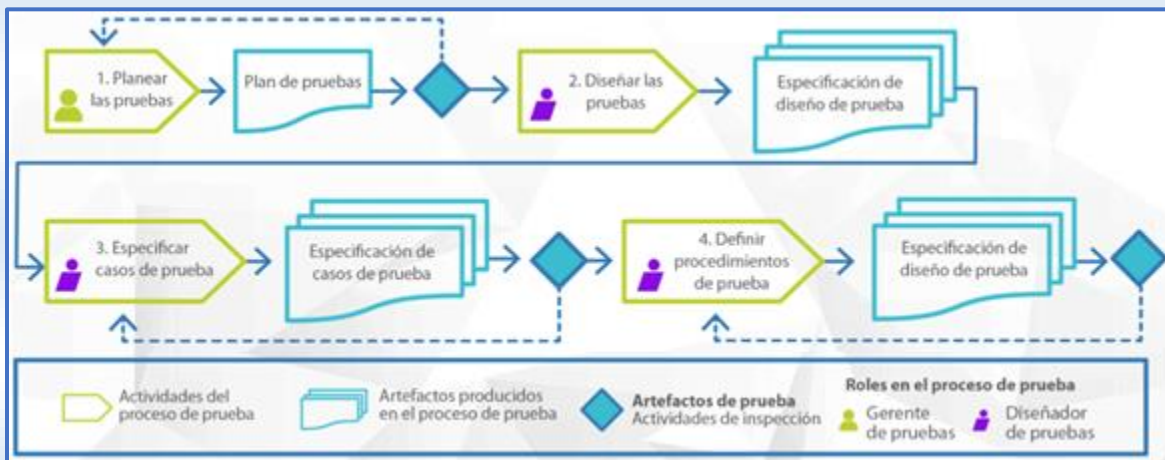
Id del documento	Requerimientos especiales
Propósito	Pasos de procedimiento

La **trazabilidad** es la habilidad para identificar items relacionados entre la documentación y el software, tales como requerimientos y casos de prueba, la trazabilidad, de acuerdo con Garzías, es bidireccional, es decir, se dirige desde los requisitos al código y viceversa. Los beneficios que ofrece la trazabilidad, de acuerdo con Alonso y otros autores, son:





- Facilita la detección de clases, componentes y casos de prueba que se requiere actualizar cuando se cambian los requisitos de un caso de uso
- Hace más fácil mantener la integridad del sistema y conservarlo actualizado



## Pruebas de caja negra

Las técnicas de caja negra o funcionales, obtienen casos a partir de los requisitos funcionales del programa a probar



- Partición equivalente
- Análisis de valor límite
- Tabla de decisiones
- Pruebas de transición de estados
- Pruebas de casos de uso

## Partición equivalente

Las particiones equivalentes, también conocidas como clases equivalentes, se pueden aplicar a cualquier nivel de pruebas. La idea detrás de esta técnica es dividir el conjunto de pruebas en grupos o conjunto que se consideren iguales

### Ejemplo

una cuenta de ahorro de un banco ofrece diferentes tasas de interés dependiendo del promedio mensual en la cuenta. Se requiere identificar los rangos. Si el promedio va de \$0 a \$100 la tasa de interés es de 3%. Si el promedio es mayor de \$100 y hasta \$1000 la tasa de interés es de 5%. si el balance es mayor a \$1000 la tasa de interés es de 7%



## Análisis de valor límite

El análisis de valor límite se basa en pruebas donde los límites dividen las particiones. Los límites son un buen lugar para encontrar los defectos, ya que estos tienden a aproximarse cerca de los límites.

### Ejemplo

una impresora tiene una opción para especificar el numero de copias que puede sacar, este va desde 1



hasta 100. Para aplicar el análisis del valor límite, tomaremos los límites mínimo y máximo (1 y 100), así como el primero y último valor de las particiones inválidas (0 y 101)

Los límites abiertos son más complicados de probar, por lo tanto, se recomienda:

1. Identificar en el requerimiento si se estableció este límite
2. Investigar en otras áreas relacionadas si está definido

De no ser así, será recomendable emplear la intuición y la experiencia para identificar y validar los límites

## Partición equivalente y análisis del valor límite

### Ejemplo:

Si se toma el tren antes de las 9:30 am de la mañana o por la tarde después de las 4:00 pm hasta las 7:30 pm (“hora pico”), se deberá pagar la tarifa completa. Un

<div> <div>Normal</div> <div>Descuento</div> <div>Normal</div> <div>Descuento</div> </div> <div> <div>9:30 am</div> <div>4:00 pm</div> <div>7:30 pm</div> </div>				
Condiciones	Particiones válidas	Límites válidos	Particiones inválidas	Límites inválidos
Hora de toma del tren para obtener descuento	De 9:31 am a 3:59 pm y a partir de 7:32 pm en adelante Ejemplos: 10:00 am, 1:00 pm, 8:00 pm	9:30 am 4:00 pm 7:31 pm	Antes de las 9:28 am y de entre las 4:02 pm a 7:29 pm Ejemplos: 9:00 am, 5:30 pm, 7:25 pm	9:29 am 4:01 pm 7:30 pm

descuento esta disponible para los trenes entre las 9:30 am y las 4:00 pm, y después de las 7:30 pm. ¿Cuáles son las particiones y los valores límite para poner a prueba los horarios de los trenes para los tipos de entrada? ¿cuáles son las particiones válidas y cuáles son particiones no válidas? ¿Cuáles son los valores límite?

### Tabla de decisiones

La tabla de decisiones muestra la combinación de entradas y salidas, las cuales son útiles para el diseño de casos de pruebas. Son empleadas para capturar reglas de negocio complejas que un sistema puede implementar. Pueden servir como guías para crear casos de pruebas, además, son útiles cuando se tiene una combinación de decisiones

Reglas →	
Talón de Condiciones	Entrada de Condición
Talón de Acciones	Entrada de Acción
Estructura de la Tabla de Decisiones	

### Ejemplo:

Si eres un nuevo consumidor abriendo una cuenta de banco, obtendrá el 15% de descuento en todas sus compras el primer día. Si ya eres cliente del banco y cuentas con una tarjeta de lealtad, obtendrá el 10% de descuento. Si tiene un cupón obtendrá el 20% de descuento, pero este cupón no puede ser utilizado junto con el descuento de nuevo cliente

Condiciones	Regla 1	Regla 2	Regla 3	Regla 4	Regla 5	Regla 6	Regla 7	Regla 8
Nuevo cliente (15%)	T	T	T	T	F	F	F	F
Tarjeta de lealtad (10%)	T	T	F	F	T	T	F	F
Cupón (20%)	T	F	T	F	T	F	T	F
Acciones								
Descuentos (%)	X	X	20	15	30	10	20	0

### Transición de estados

La transición de estados permite describir algunos aspectos del sistema conocidos como “máquinas de estado finito” y se refiere a las aplicaciones que tienen un número definido o finito de estados y las transiciones de un estado a otro definidas por reglas. Un modelo de transición de estados tiene 4 partes básicas.

El estado que ocupa el software  
La transición de un estado a otro

El evento que desencadena la transición  
La acción que resulta de la transición

### Ejemplo:

Presentemos en el ingreso de un ID para acceder a la cuenta del banco a través de un ATM:

1. Insertar tarjeta
2. Esperar solicitud de NIP
3. Ingresar NIP -primer intento erróneo
4. Ingresar NIP -segundo intento erróneo
5. Ingresar NIP -tercer intento erróneo
6. Tragarse la tarjeta – mensaje de error
7. Regresar a la pantalla de inicio



### Pruebas de caso de uso



Las pruebas de caso de uso son una técnica que nos permite identificar casos de prueba que pueden el sistema completamente basados en transacciones de principio a fin. Los casos de uso describen la interacción entre el sistema y el actor, describen el flujo de proceso con base en el uso mas frecuente. Asimismo, especifican precondiciones necesarias para que le caso de uso funcione y postcondiciones de resultados visibles al final y/o descripciones del estado final del sistema después de la ejecución del caso de uso

### Ejemplo:

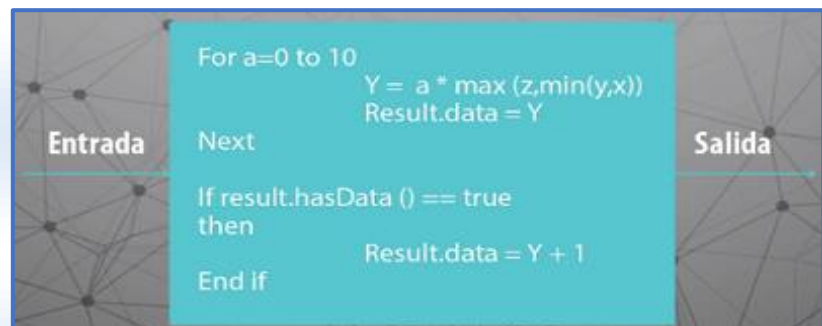
Escenario Principal	Paso	Descripción
	1	A: Insertar Tarjeta
	2	S: Valida tarjeta y solicita ID
	3	A: Ingresa ID
	4	S: Valida ID
	5	S: Permite el acceso a la cuenta

Extensión	2a	Tarjeta no válida S: Muestra mensaje y rechaza tarjeta
	3a	ID no válido S: Muestra mensaje y solicita re-ingresar ID
	4b	ID inválido 3 veces S: Se traga la tarjeta y concluye interacción con el cliente

### Pruebas de caja blanca

Las pruebas de caja blanca tienen como objetivo medir la cobertura de código y generar casos de prueba que adicionalmente cubran otros aspectos no considerados

- Pruebas de cobertura
- Pruebas de cobertura de decisiones
- Pruebas de cobertura de sentencias
- Complejidad ciclomática





## Pruebas de cobertura

las pruebas de cobertura son el grado expresado en porcentaje de elementos ejercidos de un conjunto de pruebas o suite de pruebas, su fórmula es:

$$\text{Cobertura} = \left( \frac{\text{Número de elementos ejercidos}}{\text{Número total de elementos}} \right) \times 100\%$$

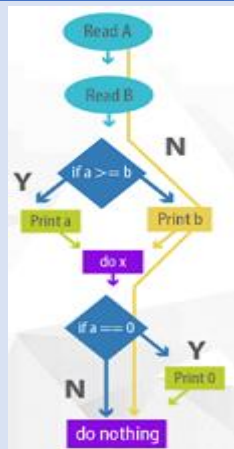
Por su parte las pruebas ad hoc son ejecutadas de manera informal, sin planeación, sin emplear una técnica de pruebas, sin definir resultados esperados y ejecutando flujos de manera arbitraria

## Pruebas de cobertura de sentencias

Las pruebas de cobertura de sentencias se refieren al porcentaje de sentencias ejercibles que han sido ejecutadas de una suite de pruebas



### Pruebas de cobertura de decisiones



Las pruebas de cobertura de decisiones miden la cobertura de estructuras condicionales, tales como sentencias IF, CASE, DO, WHILE, REPEAT UNTIL, etc. Para alcanzar una cobertura completa de decisiones es necesario que se evalúe la condición como verdadera y falsa.

La cobertura de decisión es mas fuerte que la cobertura de sentencias, porque el 100% de cobertura de decisiones garantiza el 100% de cobertura de sentencias

### Complejidad ciclomática

Una vez calculada la complejidad ciclomática de un fragmento de código, se puede determinar el riesgo que supone utilizando los rasgos definidos en la siguiente tabla:

Complejidad Ciclomática	Evaluación de Riesgo
1 - 10	Programa simple, sin mucho riesgo
11 - 20	Más complejo, riesgo moderado
21 - 50	Complejo, Programa de alto riesgo
50	Programa no testeable, Muy alto riesgo

### Complejidad ciclomática

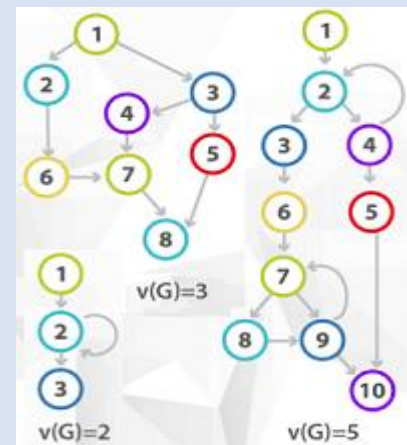
La complejidad ciclomática se basa en el numero de nodos y líneas de un diagrama y permite determinar el 100% de cobertura, su fórmula es:

$$CC = \text{Líneas} - \text{Nodos} + 2P$$

Donde  $P$ : es el número de componentes conectados

Una versión simplificada para el calculo de la complejidad ciclomática es la siguiente

$$CC = \text{Numero de condiciones} + \text{Numero de retornos o salidas}$$



## Pruebas basadas en la experiencia

Se basan en la pericia y experiencia de los testers para llevarse a cabo

Técnicas de diseño de pruebas basadas en experiencia:

- **Pruebas de suposición de errores:** es la técnica de diseño de pruebas donde la experiencia del tester es usada para anticipar que defectos se presentaran en un componente o sistema bajo prueba, lo que permite el diseño de pruebas específicas para evidenciar los errores



- **Pruebas exploratorias:** son pruebas donde la planeación es mínima y la ejecución de pruebas es al máximo. El diseño y la ejecución se llevan a cabo de manera paralela. No hay documentación formal de condiciones de prueba, casos de prueba o script de pruebas. Consiste en ir explorando en el software conforme se va probando y aprendiendo como funciona
- **Pruebas basadas en listados**
- **Aplicación de la mejor técnica**
- **Ataque de fallas:** es un intento directo y enfocado para evaluar la calidad, particularmente la confiabilidad de un objeto a prueba forzándolo a fallar



## Conclusiones

Las pruebas de caja negra cubren entre el 60% y un 75% de sentencias	Las pruebas de caja blanca cubren entre el 40% y un 60% de decisiones
Las pruebas ad hoc cubren un 30% de código existente en la aplicación, dejando un 70% sin ser ejecutado	Las pruebas ad hoc cubren aproximadamente un 20%, dejando un 80% sin ser ejecutado

## Organización de pruebas

Las personas que forman parte del equipo de pruebas son:



**Test manager o líder de pruebas:** es la persona responsable de la gestión, actividades y recursos del proyecto. Es el individuo que dirige controla, administra, planea y regula la evaluación del objeto bajo pruebas

**Tester:** es el profesional con habilidades que esta involucrado en las pruebas de un componente o sistema. Es conocido como ingeniero de pruebas

## Actividades del tester:

- Revisar y contribuir en la elaboración de los planes de pruebas
- Identificación de las condiciones de prueba
- Evaluación de requerimientos y diseño de pruebas
- Reportar incidencias detectadas
- Ejecutar casos de prueba
- Dar seguimiento

## Actividades del líder de pruebas:

- El líder de pruebas se ve involucrado en la planeación, monitoreo y control de las actividades de pruebas
- En colaboración con otros implicados, definen los objetivos de las pruebas, así como la política
- Estima las pruebas a ejecutar y negocia con los directivos la adquisición de los recursos necesarios
- Guía el monitoreo, análisis, diseño, implementación y ejecución de los casos, procedimientos y suites de pruebas

## El plan de pruebas

El plan de pruebas, es el plan del proyecto para las pruebas que se realizaran. No es una especificación de diseño de pruebas, una colección de casos de prueba o procedimiento de pruebas. ¿Por qué escribir un plan de pruebas?

- Permite ordenar las ideas de lo que se va a hacer

- Nos confronta con los retos a realizar y resalta los temas prioritarios
- Permite ubicar a todos los involucrados en el mismo contexto

El estándar IEEE 829 define los elementos mínimos para el plan de pruebas:

ID del documento	introducción	Elementos a probar	Que se aprobara
Que no se probará	enfoque	Criterio pasado/fallado	Criterios de suspensión y reanudación de pruebas
entregables	tareas	Ambientes necesarios	responsables
Necesidades de recursos y capacitación	calendario	Riesgos y contingencias	Quien aprueba



Escribir un buen plan de pruebas es tan sencillo como escribir una novela, pero ambas tareas requieren un enfoque organizado, así como ideas claras. Se requiere dividir el trabajo en diferentes niveles, así como definir una estrategia

Para los criterios de entrada y salida se deben considerar:

- La adquisición y suministro de recursos
- El estado de los elementos a probar
- La solución a los defectos encontrados
- La cobertura de las pruebas
- Calidad del sistema
- Presupuesto para realizar más pruebas
- El riesgo latente en relación al estado que guarda el sistema
- No hay que olvidar que el criterio de salida guarda una fuerte relación con la calidad, el presupuesto y el tiempo

## Estimación de pruebas

**La granularidad de las tareas:** estimación, monitoreo y control

**Técnicas de estimación:** generalmente se requiere negociar las estimaciones dependiendo de cuestiones como los recursos involucrados, tiempo, etc. existen dos técnicas de estimación de pruebas:

- **Basadas en personas:** parte de consultar a las personas que realizarán las pruebas e ir detallando las actividades, así como a gente con cierta experiencia. La idea es generar un consenso para definir la estimación
- **Basadas en métricas:** se basa en proyectos anteriores, considerando cuestiones como complejidad, tamaño, duración, etc.



## Factores que afectan el esfuerzo de las pruebas

Existen factores que se deben tener en cuenta y que pueden jugar en contra. Un ejemplo son las pruebas no funcionales, como usabilidad, confiabilidad, seguridad y desempeño, que requieren un fuerte presupuesto y gran consumo de tiempo. Algunos factores son:

- El uso de tecnologías innovadoras
- La necesidad de diversas configuraciones
- Demasiadas reglas de negocio o de seguridad

- La distribución del equipo, especialmente cuando las diferencias horarias son significativas
- La documentación bien detallada requiere grandes esfuerzos, así como tiempo para ser generada
- La presión de tiempo se debe tener en cuenta, pues puede implicar asumir grandes riesgos

### Métodos de estimación de esfuerzo de pruebas

QA Symphony: <http://www.qasymphony.com/media/2012/01/Test-Case-Point-Analysis1.pdf>

Cognizant: [https://www.cmcrossroads.com/sites/default/files/article/file/2013/XUS373692file1\\_0.pdf](https://www.cmcrossroads.com/sites/default/files/article/file/2013/XUS373692file1_0.pdf)

Priya Chaudhary y C.S. Yadav: <http://ijarcet.org/wp-content/uploads/IJARCET-VOL-1-ISSUE-1-35-40.pdf>

### Estrategias de pruebas

- **Analítica:** parte del análisis de riesgos empleando documentos e implicados para posteriormente planear, estimar, diseñar y priorizar las pruebas con base en el riesgo
- **Basada en modelos:** parte de modelos matemáticos bajo los cuales se espera que sea el comportamiento del sistema bajo pruebas
- **Metódico:** se apoya de checklist que parten de normas, por ejemplo, ISO 9126 para el diseño, implementación, ejecución, etc., de las pruebas
- **Cumplimiento de estándares o procesos:** se basan en una norma, por ejemplo, la IEEE 829, para su proceso de pruebas o adoptan una metodología ágil
- **Dinámica:** es similar a las pruebas exploratorias que tienen como objetivo encontrar tantos defectos como sea posible durante la ejecución
- **Directa:** los usuarios o desarrolladores indican que parte de las pruebas requiere mayor atención
- **Regresión:** son un conjunto de procedimientos de pruebas generalmente automatizados que permiten identificar errores introducidos en partes donde la funcionalidad ya era adecuada



### Monitoreo y control de pruebas

Para cuantificar las fallas y los defectos se utilizan: tasa de fallas de pruebas y densidad de defectos

- **Control de pruebas:** el control de pruebas se refiere a las acciones correctivas para tratar de alcanzar el mejor resultado para el proyecto
- **Monitoreo de pruebas:** el monitoreo de pruebas es el indicador que dispara este evento en caso de que las pruebas se salgan de los límites establecidos
- **Reporte del estado de las pruebas:** es el profesional con habilidades que esta involucrado en las pruebas de un componente o sistema

El monitoreo de pruebas sirve a varios propósitos durante el proyecto:

- Da retroalimentación al área de pruebas, permitiendo guiar y mejorar el proceso
- Proporciona visibilidad sobre los resultados
- Indica la cobertura contra el criterio de salida
- Genera información para pruebas futuras

La norma IEEE 829 especifica lo que debe contener el template de log de pruebas:

- |   |   |
|---|---|
| • ID de documento   | • Descripción (elementos probados y ambiente) |
| • Eventos (ejecución, procedimiento, ambiente, anomalías e incidencias) |   |

Para cuantificar las fallas y los defectos, se utilizan las siguientes medidas:

- **Tasa de fallas:** numero de fallas de una categoría dad en una unidad de medida. Ejemplo: fallas por unidad de tiempo y fallas por número de transacciones
- **Densidad de defectos:** numero de defectos identificados en un componente o sistema dividido entre el tamaño del componente o sistema. Ejemplo: líneas de código, numero de clases y puntos por función



## Reporte del estado de las pruebas

El reporte del estado de las pruebas se refiere a la información generada de las pruebas y la comunicación hacia los implicados respecto al estatus de las pruebas

La norma IEEE 829 especifica los lineamientos para la generación de un reporte de las pruebas:

ID del documento	Resumen de resultados
Resumen	Evaluación
Variaciones	Resumen de actividades
Evaluación integral	Quien aprueba

## Gestión de configuración

- Determina que elementos componen el sistema
- Garantiza que estos elementos sean manejados de manera cuidadosa
- Tienen un número importante de implicaciones dentro de las pruebas
- Permite saber que versión se está probando
- Al liberar un sistema para pruebas, se acompaña de un release

Que elementos son los que componen el sistema:

Código	hardware
Scripts de pruebas	datos
software de tercera parte	documentación

La norma IEEE 829 especifica los lineamientos que debe contener este documento:

ID del documento	Estatus
Ubicación	Quién aprobó
Elementos transmitidos	



## El riesgo y las pruebas

El riesgo se define como el factor que puede resultar en consecuencias negativas, generalmente expresado en probabilidades, generalmente expresados en probabilidad e impacto. El riesgo se clasifica en riesgo del producto y riesgo del proyecto

- **Riesgo del producto:** es la posibilidad de que el sistema o software falle al satisfacer las expectativas del usuario, cliente o implicado. Los problemas pueden causar daño financiero u otro tipo de daño relacionado con alguna característica de calidad
- **Riesgo del proyecto:** es el riesgo que este asociado directamente con las liberaciones tardías y todo lo que esto involucra. Existen algunos riesgos indirectos, tales como excesivos retardos en la reparación de defectos encontrados durante la fase de pruebas

## Administración de riesgos

- **Mitigación:** tomar las acciones necesarias para reducir la probabilidad de riesgo
- **Contingencia:** tener un plan de acción para reducir el impacto que tenga el riesgo una vez que se ha presentado
- **Transferencia:** convencer a los miembros o implicados en el proyecto de reducir la probabilidad o responsabilizarse del daño que este pueda tener
- **Ignorarlo:** no hacer nada acerca del riesgo, lo cual es una opción inteligente solo cuando hay muy poco que hacer o cuando la probabilidad e impacto son muy bajos



Algunos conceptos importantes en la gestión de incidencias son:

	<b>Registro de incidencia</b> Consiste en registrar los detalles de cualquier incidencia ocurrida
	<b>Reporte de incidencias</b> Documento donde se reporta cualquier evento que se haya presentado
	<b>Reporte de defectos</b> Documento donde se reportan fallas del componente o sistema
	<b>Porcentaje de detección de defectos</b> Numero de defectos encontrados en una fase de pruebas, dividido entre el numero de defectos encontrados en esa fase y las posteriores
	<b>Prioridad</b> Nivel de importancia asignado a un elemento
	<b>Severidad</b> Grado de impacto que un defecto tiene en el desarrollo u operación del componente o sistema






El estándar IEEE 829 especifica los elementos que requiere un reporte de incidencias:

- ID del documento
- Resumen
- Impacto

Descripción de la incidencia (entrada, resultado esperado, resultado actual, anomalías, fecha y hora, pasos, ambiente e intentos para replicarlos)

## Organización de pruebas

	<b>Líder de pruebas</b> Responsable de la gestión, actividades y recursos del proyecto. Dirige, controla, administra, planea y regula la evaluación del objeto bajo pruebas
---	--

	<b>Tester</b> Es el profesional con habilidades que está involucrado en las pruebas de un componente o sistema
	<b>Plan de pruebas</b> Es el plan del proyecto para las pruebas que se realizarán

### El proceso del riesgo



### Evaluación dinámica del riesgo



### Herramientas de software de pruebas de software

- **Script de pruebas:** son datos e instrucciones en una sintaxis formal para ser utilizados en una herramienta de automatización. Se pueden automatizar uno o más casos de pruebas, navegación, inicialización u operaciones de configuración de entorno
- **Suite de pruebas:** es uno o mas conjuntos de pruebas reunidos para satisfacer un objetivo de prueba. Las suites constituyen un conjunto de scripts y el orden de ejecución de los mismo
- **Pruebas de regresión:** las pruebas de regresión tienen como objetivo verificar que no ocurrió una regresión en la calidad del producto luego de un cambio, asegurándose de que los cambios nos introducen un comportamiento no deseado o errores adicionales
- **Pruebas de humo:** son un conjunto de pruebas aplicadas a cada nueva versión, su objetivo es validar que las funcionalidades básicas de la versión se comportan según lo especificado.

### Tipos de herramientas – gestión de pruebas

Las características proporcionadas por las herramientas de gestión de pruebas incluyen las que se enumeran a continuación, algunas herramientas proporcionaran todas estas características, otras pueden proporcionar una o más de las características, sin embargo, este tipo de herramientas todavía serian clasificadas como la gestión de pruebas

### Características de las herramientas

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>• Gestión de pruebas</li> <li>• Calendario de pruebas que serán ejecutadas</li> <li>• Gestión de las actividades de pruebas</li> <li>• Interfaces con otras herramientas</li> <li>• Trazabilidad de las pruebas</li> </ul> | <ul style="list-style-type: none"> <li>• Resultados y defectos</li> <li>• Resultado de los registros</li> <li>• Preparación del progreso de reportes</li> <li>• Basados en métricas</li> </ul> |
|---|--|

### Herramientas de gestión de pruebas

- **Testlink:** Testlink es un Sistema de gestión de pruebas de software basado en la web, es de código abierto (open source) y dispone de una amplia comunidad de foristas y voluntarios que publican guías e información sobre cómo utilizarlo
- **Gemini:** es una solución de automatización de flujo de trabajo, comunicaciones y reportes en una gran variedad de escenarios de gestión de tecnología de información (TI)
- **Hewlett Packard Quality Center (HP QC) o AML:** es un software de gestión de calidad suministrado por la división de software de HP, que ofrece diversas capacidades para el aseguramiento de calidad, gestión de requerimientos, gestión de software testing, entre otros

- **Redmine:** es una aplicación de software para la gerencia de proyectos, puede funcionar en diversas plataformas y bases de datos de distintos proveedores
- **Zephyr/jira:** es un software de gestión de ciclo de pruebas de software, disponible como versión empresarial y como un add-on de Atlassian jira

### Herramientas para gestión de requerimientos

Las herramientas para gestión de requerimientos soportan el registro de requerimientos, atributos y trazabilidad. Características de la herramienta:

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• Almacenamiento de requerimientos</li> <li>• Almacenamiento de información relacionada con atributos</li> </ul> | <ul style="list-style-type: none"> <li>• Trazabilidad entre requerimientos y pruebas</li> <li>• Trazabilidad para cada nivel de pruebas</li> <li>• Cobertura de requerimientos</li> </ul> |
| <ul style="list-style-type: none"> <li>• Identificación de requerimientos indefinidos, extraviados o definidos de manera tardía</li> </ul>              |   |

### Herramientas para gestión de incidentes

Las herramientas para la gestión de incidentes facilitan el registro y rastreo de incidencias de pruebas, además, facilitan el rastreo de correcciones y re-ejecuciones. Características de las herramientas:

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>• Almacenajes de atributos de incidencias</li> <li>• Priorización de incidencias</li> </ul> | <ul style="list-style-type: none"> <li>• Asignación de acciones</li> <li>• Almacenaje de archivos adjuntos</li> </ul> |
| <ul style="list-style-type: none"> <li>• Estatus de incidencias</li> </ul>   |   |

### Herramientas de gestión de configuración

Las herramientas de gestión de la configuración proveen soporte para gestión y control de la configuración, cambio y versionado de liberaciones. Características de la herramienta:

- Almacenamiento de información por versiones
- Gestionar la construcción y liberación
- Trazabilidad entre software y suite de pruebas
- Control de accesos
- Conserva el rastreo entre que versiones pertenecen a que configuración



### Tipos de herramientas – de soporte para análisis estático

#### Herramientas para soporte de revisiones

Las herramientas para soporte de revisiones proveen soporte para la revisión del proceso, incluyendo planeación y rastreo, comunicación y reporte de métricas. Características de las herramientas:

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>• Almacén y ordenamiento de comentarios de revisiones</li> <li>• Permite comunicar los comentarios a gente relevante</li> <li>• Coordinar revisiones en línea</li> <li>• Rastreo y seguimiento a comentarios, incluyendo defectos, además de proveer estadísticas</li> </ul> | <ul style="list-style-type: none"> <li>• Trazabilidad entre comentarios</li> <li>• Repositorio para reglas, procedimientos y listas de chequeo</li> <li>• Monitoreo del estatus de la revisión</li> <li>• Recolección de métricas y reporte</li> </ul> |
|---|--|

#### Herramientas de soporte para análisis estático

- **PMD:** analizador estático de código que utiliza unos conjuntos de reglas para identificar problemas dentro del software. Detecta cosas como código duplicado, código muerto, complejidad de métodos (if necesarios, etc.). trabaja principalmente con lenguajes java, aunque, con menos soporte, también posee conjuntos de reglas para JavaScript, XSL y ECMAScript
- **SONAR:** una herramienta de software libre y gratuita que permite gestionar la calidad del condigo fuente. Al instalarla podremos recopilar, analizar y visualizar métricas del código fuente.

SONAR es básicamente la fusión de las herramientas Checkstyle y PMD, mas otras como Findbugs, Clover y cobertura. También realiza un histórico de todas las métricas del proyecto. Permite visualizar informes con resúmenes de las métricas

- **Google CodePro Analytix:** otra de las herramientas de calidad software, ofrece un entorno para evaluación de código, métricas análisis de dependencias, cobertura de código, generación de test unitarios, etc. Mira las excepciones, refactorizaciones potenciales, convenios de JavaDoc, métricas, etc. Disponible como plugin de Eclipse
- **Simian:** herramienta para detectar código duplicado, si hay código repetido, costara mas dinero mantener el software en desarrollos realizados con los lenguajes java, C, C#, C++, COBOL, Ruby, JSP, ASP, HTML, XML y Visual Basic
- **Checkstyle:** herramienta que se utiliza para comprobar que el código analizado cumple con una serie de reglas de estilo, ejemplo: analiza el código según el estándar “Sun Code Conventions” (mira las cabeceras importaciones de paquetes, javadoc, etc.)

### Herramientas para análisis estático

Las herramientas para el análisis estático proveen soporte para analizar artefactos de software. Son frecuentemente utilizadas por desarrolladores para permitirles entender y evaluar la estructura del código. Estas herramientas son una extensión de los compiladores, de hecho, algunos compiladores ofrecen análisis estático. Características de la herramienta:

- |   |  |
|---|--|
| <ul style="list-style-type: none"> <li>• Cálculo de métricas, tales como la complejidad ciclomática</li> <li>• Identificar anomalías y defectos en el código</li> </ul> | <ul style="list-style-type: none"> <li>• Hacer cumplir los estándares de código</li> <li>• Auxiliar en el entendimiento del código</li> <li>• Análisis de estructura y dependencias</li> </ul> |
|---|--|

### Herramienta para modelado

Ayudan en la validación de modelos de sistemas de software. son empleadas comúnmente por los desarrolladores para evaluar el diseño del software. Estas herramientas, al igual que las de análisis estático, se pueden utilizar antes de realizar pruebas dinámicas, permitiendo identificar defectos en etapas tempranas del desarrollo. Características de la herramienta:

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>• Identificar inconsistencias y defectos en el modelo</li> <li>• Priorizar áreas del modelo que requieren ser probadas</li> </ul> | <ul style="list-style-type: none"> <li>• Predecir respuestas y comportamientos en el sistema bajo situaciones específicas, tales como nivel de carga</li> <li>• Permite entender el funcionamiento del sistema, apoyado de herramientas como UML</li> </ul> |
|--|---|

### Tipos de herramientas – para soporte de especificaciones de pruebas

#### Herramientas para diseño de pruebas

Las herramientas para diseño de pruebas soportan la generación de datos de entrada y construcción de casos de prueba. Permiten generar combinaciones de factores para asegurar que todas las combinaciones de sistemas operativos y exploradores sean probadas. Un ejemplo, son los arreglos ortogonales. Características de la herramienta:

- |   |  |  |
|---|--|--|
| <ul style="list-style-type: none"> <li>• Requerimientos</li> <li>• Modelos</li> </ul> | <ul style="list-style-type: none"> <li>• Condiciones de prueba</li> <li>• Interfaces de usuario</li> </ul> | <ul style="list-style-type: none"> <li>• Código</li> </ul> |
|---|--|--|

#### Herramientas para soporte de especificación de pruebas

- **MOVASS:** modelo y herramienta para el proceso de especificación de pruebas de validación de sistemas software

#### Herramientas para preparación de datos

Las herramientas para preparación de datos permiten la selección de datos de una base de datos para ser utilizados en pruebas cuando la confiabilidad y desempeño de los datos requiere ser similar a los existentes en producción. Características de la herramienta:



- Permite extraer registros de archivos o bases de datos en ambientes productivos
- Protege los datos con mensajes predefinidos para cuidar la identidad de las personas

- Habilita los registros para ser acomodados en diferente orden, genera nuevos registros con datos pseudoaleatorios
- Permite construir un gran número de registros para contar con grandes volúmenes de datos en pruebas

## Tipos de herramientas – para soporte de ejecución y registro de pruebas

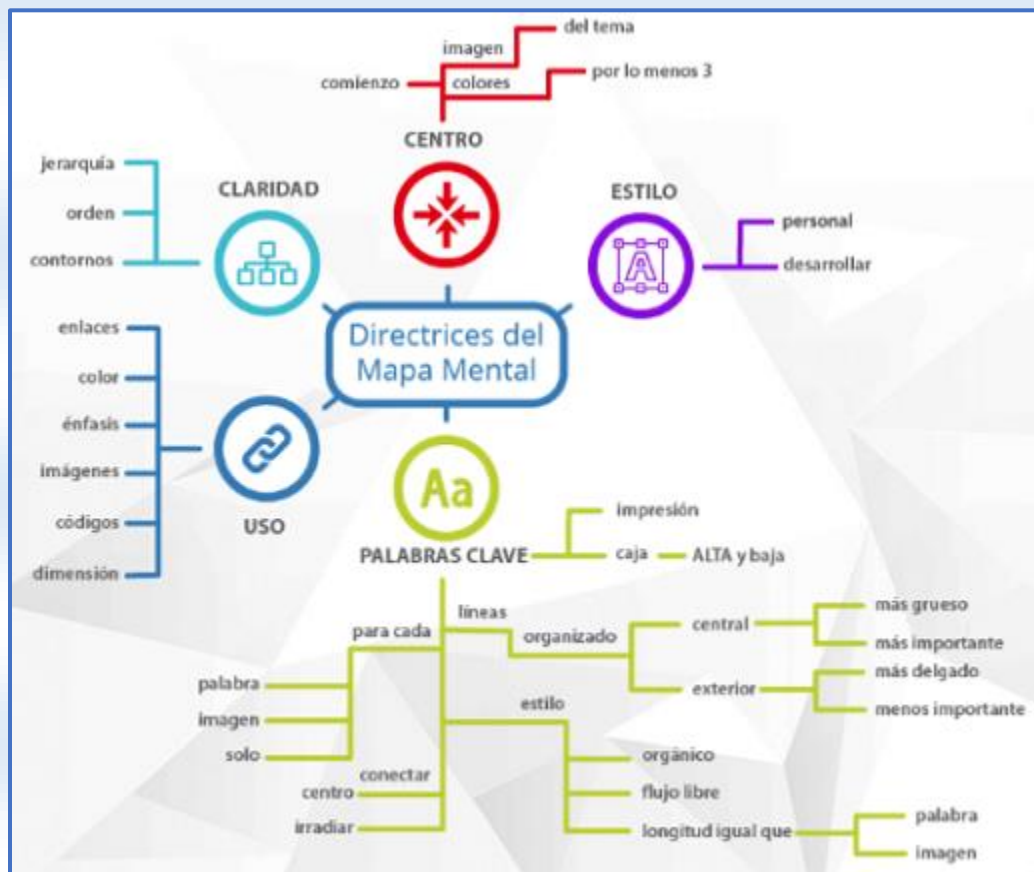
### Herramientas para ejecución de pruebas

Las herramientas de ejecución de pruebas permiten la ejecución del software de manera automatizada a través de scripts. Características de la herramienta:

- Grabación de pruebas de entrada
- Almacenamiento de resultados esperados en pantalla o archivo
- Ejecución de pruebas de scripts almacenados
- Comparación dinámica de pantallas, elementos, vínculos, controles, objetos y valores
- Filtrado para ejecución de subconjuntos de pruebas
- Medición de tiempos para pruebas
- Sincronía de entradas con la aplicación bajo pruebas
- Envío del resumen de resultados a una herramienta de gestión de pruebas

### desventajas

- El script no conoce el resultado esperado hasta que se programa
- Un pequeño cambio en el software invalida cientos de scripts de pruebas
- Eventos inesperados durante la ejecución pueden ser mal interpretados por la herramienta



- **Selenium:** compuesto por dos herramientas: Selenium IDE y Selenium WebDriver. La primera es un plugin de Firefox que genera un entorno de desarrollo y permite crear casos de pruebas

para aplicaciones web. La segunda, Selenium WebDriver, ejecuta las pruebas. Este entorno de automatización de pruebas automáticas opera en los principales navegadores (IE, Mozilla, Chrome y Opera). Además, permite pruebas para dispositivos móviles, para Iphone y Android. Utiliza los siguientes lenguajes: Python, Ruby, Java y C#. la licencia es "Apache 2.0 License"

- **JMeter:** Aplicación de escritorio en java y dentro del proyecto jakarta. Esta herramienta permite realizar pruebas funcionales (y de rendimiento) para aplicaciones web. Trabaja con los siguientes protocolos: HTTP, HTTPS, SOAP, JDBC, LDAP, Mail, POP3(S) E IMAP(S). la licencia es "Apache 2.0 License"
- **Testlink:** permite crear y gestionar casos de prueba, organizarlos en planes de pruebas, realizar un seguimiento de los resultados, establecer trazabilidad con los requisitos, generar informes, etc. Se integra con otros sistemas de seguimiento de bugs y ticketing como Bugzilla, mantis, etc. La licencia es GPL.
- **UFT:** el software HPEUFT automatiza las pruebas a través de una experiencia de usuario intuitiva y visual que relaciona las pruebas manuales, automatizadas y basadas en marcos en un solo IDE. Esta solución de largo alcance reduce significativamente el coste y la complejidad del proceso de pruebas funcional e impulsa continuamente la calidad.

## Herramientas para pruebas unitarias

Las herramientas para pruebas unitarias proveen ambientes para pruebas unitarias o de componentes, donde un componente puede ser probado de manera aislada o en conjunto con Stubs y drivers. También tiene alcance para debugeo. Característica de la herramienta:

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>• Medición de cobertura para nivel de código</li> <li>• Soporte para debugeo</li> <li>• Almacenamiento de pruebas</li> <li>• Registro de resultado pasado/fallado para cada prueba</li> </ul> | <ul style="list-style-type: none"> <li>• Ejecutar un conjunto de pruebas dentro de un marco de referencia</li> <li>• Recibir salida por el software bajo pruebas</li> <li>• Proveer entradas para el software bajo pruebas</li> </ul> |
|--|---|

En java existe el Junit (<http://www.junit.org>) que es un framework para implementar las pruebas unitarias a nuestro código en java. Este framework es distribuido de forma gratuita en la dirección antes mencionada, otras herramientas son: TestNG, CPPUnit, Visual Studio Unit Testing

## Herramientas para comparar resultados de pruebas

Las herramientas para comparar resultados de pruebas como su nombre lo indica, comparan un valor obtenido contra uno dado de manera automática. Característica de la herramienta:

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>• Comparación dinámica de eventos que ocurren durante la ejecución</li> </ul> | <ul style="list-style-type: none"> <li>• Filtrado de subconjuntos de datos, actual contra esperado</li> </ul> |
| <ul style="list-style-type: none"> <li>• Comparación post-ejecución de datos almacenados</li> </ul>                  |   |

- **PrestoSoft:** ExamDiff Pro es un potente pero intuitivo y fácil de usar archivo visual y herramientas de comparación de directorios para Windows. Cuenta con una funcionalidad única que distingue a ExamDiff de otros programas de comparación

## Herramientas para cobertura de código

Las herramientas para cobertura de código proveen medición objetiva de que elementos estructurales han sido ejecutados de una suite de pruebas. Característica de la herramienta:

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• Identificar elementos para la cobertura</li> <li>• Identificación de pruebas de entrada para ejercer elementos aun no cubiertos</li> </ul> | <ul style="list-style-type: none"> <li>• Generación de Stubs y Drivers</li> <li>• Cálculo del porcentaje de elementos ejercidos por una suite de pruebas</li> </ul> |
| <ul style="list-style-type: none"> <li>• Reporte de elementos que han sido ejecutados</li> </ul>  |   |

- **Coverity Code Advisor:** la solución saca a la superficie los defectos de calidad y de seguridad en el flujo de trabajo de desarrolladores, con una guía de corrección precisa y procesable, basada en técnicas patentadas y una década de investigación, desarrollo y análisis de mas de 10 mil millones de líneas de código fuente abierto y propietario.
- **CodeCover:** es una herramienta de cobertura de código fuente abierto extensible con las siguientes características:

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>• Compatible con la cobertura de sentencias</li> <li>• Cobertura de decisiones</li> </ul> | <ul style="list-style-type: none"> <li>• Cobertura de condición estricta</li> <li>• Cobertura de loops</li> </ul> |
|--|---|

## Tipos de herramientas – seguridad

Las herramientas para pruebas de seguridad, como su nombre lo indica, son utilizadas para evaluar la seguridad. Tratan de corromper el sistema para evaluar si esta o no protegido. Característica de la herramienta:

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>• Identificación de virus</li> <li>• Detección de intrusos</li> <li>• Simulación de ataques externos</li> </ul> | <ul style="list-style-type: none"> <li>• Pruebas a puerto abiertos u otros puntos de ataque</li> <li>• Identificación de debilidades en passwords</li> </ul> |
|--|--|

## Herramientas para pruebas de seguridad

- **Wireshark:** se trata de un programa analizador de protocolos de red o sniffer, que permite capturar y navegar de forma interactiva por los contenidos de los paquetes capturados en la red
- **Kali Linux:** es una distribución basada Debian GNU/Linux, diseñada principalmente para la auditoria y seguridad informática en general, Kali ha sido desarrollado a partir de la reescritura de BackTrack, que se podría denominar como la antecesora de Kali Linux
- **NMAP (“Network Mapper”):** es una herramienta gratuita para la exploración de la red, diseñada para analizar rápidamente grandes redes, aunque funciona muy bien contra equipos individuales. NMAP utiliza paquetes IP para determinar que hosts están disponibles en la red, que servicios (nombre de la aplicación y la versión) ofrecen estos equipos, que sistemas operativos (y versiones de sistemas operativos) se están ejecutando, que tipo de filtros de paquetes o cortafuegos están en uso, y docenas de otras características. NMAP se ejecuta en la mayoría de los ordenadores y la consola y versiones graficas disponibles. NMAP es libre y de código abierto
- **Anubis:** es una aplicación desarrollada por el equipo Flu Project, diseñada para anexionar gran parte de las herramientas necesarias para los procesos de las auditorias de seguridad y test de intrusión dedicados a la búsqueda de información, denominados Footprinting y Fingerprinting, en una única herramienta. Con esta herramienta se puede descubrir nueva información que de manera manual no se podría descubrir, gracias a las automatizaciones que lleva Anubis incorporadas.

## Tipos de herramientas – para monitoreo y desempeño

### Herramientas para análisis dinámico

Las herramientas para análisis dinámico proveen información en tiempo de ejecución del estado del software. Son útiles para identificar punteros no asignados, uso de la memoria y perdida de la memoria. Característica de la herramienta:

- |  |   |
|--|---|
| <ul style="list-style-type: none"> <li>• Detectar perdidas de memoria</li> </ul>       | <ul style="list-style-type: none"> <li>• Identificar apuntadores nulos</li> </ul> |
| <ul style="list-style-type: none"> <li>• Identificar dependencias de tiempo</li> </ul> |   |

- **Process Monitor:** es una herramienta de supervisión avanzada para Windows que muestra la actividad del sistema de archivos en tiempo real, registro y proceso /hilo. Sus potentes funciones hacen de Process Monitor una utilidad esencial en la solución de problemas del sistema y un kit de herramientas de caza malware

- **Webinspect:** una herramienta de pruebas de seguridad dinámica automatizada (DAST) que imita las técnicas de piratería y ataques del mundo real y aporta un análisis dinámico e integral de los servicios y las aplicaciones web complejas
- **ESET SysInspector:** es una herramienta de diagnóstico gratuita de última generación para sistemas Windows. También es un aparte integrada de ESET Smart Security 4 y ESET NOD32 Antivirus 4. Su tarea consiste en observar con atención el sistema operativo del usuario y capturar detalles, como procesos activos, contenido del registro, elementos del inicio y conexiones de red. Una vez que toma esta fotografía instantánea del sistema. ESET SysInspector aplica la heurística para asignarle un nivel de riesgo a cada uno de los objetos registrados. Su interfaz gráfica de usuario intuitiva le permite al usuario seleccionar fácilmente objetos específicos de un color determinado (cada color corresponde a un nivel de riesgo) de entre la gran cantidad de datos usando una herramienta de selección, para poder analizarlos con más detalle. ESET SysInspector es una aplicación conveniente para la caja de herramientas de todo experto en IT que quiera ser el primero en responder

### Herramientas para desempeño, carga y estrés

Las herramientas para desempeño, carga y estrés proveen información en tiempo de ejecución del software en condiciones extremas o poco comunes, simulando una carga excesiva de usuarios accediendo al sistema en el mismo tiempo. Son útiles para identificar como se comportará el uso de los recursos del sistema bajo altas demandas. Característica de la herramienta:

- |   |   |
|---|---|
| <ul style="list-style-type: none"> <li>• Con las herramientas de desempeño, se simulará el diseño de la carga, ejemplo, datos aleatorios similares a los existentes en producción</li> <li>• Incluir retardos para simular entradas más reales</li> </ul> | <ul style="list-style-type: none"> <li>• ¿Qué aspectos exactamente se medirán?</li> <li>• ¿Qué hacer si la ejecución se detiene de forma prematura?</li> <li>• ¿Cómo presentar la información?</li> </ul> |
|---|---|

### Herramientas open source para pruebas de carga y rendimiento

FunkLoad	FWPTT load testing	loadUI	jMeter
----------	--------------------	--------	--------

### Herramientas comerciales para pruebas de carga y rendimiento

HP LoadRunner LoadStorm	NeoLoad Forecast	WebLOAD profesional Load impact	WebServer stress Tool ANTS Advanced.NET testing System
----------------------------	---------------------	------------------------------------	---

### Herramientas para monitoreo

Las herramientas para monitoreo son utilizadas para evaluar, de manera constante, el estado del sistema para identificar los problemas lo antes posible y corregir o mejorar. Característica de la herramienta:

- |  |  |
|--|--|
| <ul style="list-style-type: none"> <li>• Identificar problemas y enviar alertas</li> <li>• Registrar tiempo real y generar históricos</li> </ul> | <ul style="list-style-type: none"> <li>• Encontrar ajustes óptimos</li> <li>• Monitorear el tráfico en la red</li> </ul> |
| <ul style="list-style-type: none"> <li>• Monitorear el número de usuarios en la red</li> </ul>   |  |

### Tipos de herramientas – para aplicaciones específicas

Las herramientas para aplicaciones específicas sirven para procesos específicos, por ejemplo, desempeño de servicios web o sistemas embebidos.

### Resumen

Herramientas de análisis estático	Herramientas de gestión de pruebas
Las herramientas de análisis estático son muy útiles para los desarrolladores, sobre todo durante la compilación	Las herramientas de gestión de pruebas generan gran cantidad de información, sin embargo, esta requiere ser sintetizada



Durante la introducción de la herramienta pueden surgir números problemas, sobre todo con códigos viejos referentes a la estandarización de código

Los reportes generados pueden ser muy útiles en el momento, sin embargo, pueden ser inútiles dos o tres meses después





### Factores de éxito:

- Promover entrenamiento adecuado
- Monitorear el uso de la herramienta
- Implementar un mecanismo en mejora continua




### Tipos de herramientas

	<b>Herramientas de gestión de pruebas</b> <ul style="list-style-type: none"> <li>Gestión de requerimientos</li> <li>Gestión de incidencias</li> <li>Gestión de configuración</li> </ul>
	<b>Herramientas para soporte para análisis estático</b> <ul style="list-style-type: none"> <li>Soporte de revisiones</li> <li>Análisis estático</li> <li>Modelado</li> </ul>
	<b>Herramientas para soporte de especificaciones de pruebas</b> <ul style="list-style-type: none"> <li>Diseño de pruebas</li> <li>Preparación de datos</li> </ul>
	<b>Herramientas para soporte de ejecución y registro de pruebas</b> <ul style="list-style-type: none"> <li>Ejecución de pruebas</li> <li>Pruebas unitarias</li> <li>Comparar resultados de pruebas</li> <li>Cobertura de código</li> </ul>
	<b>Herramientas de seguridad</b> <ul style="list-style-type: none"> <li>Pruebas de seguridad</li> </ul>
	<b>Herramientas de monitoreo y desempeño</b> <ul style="list-style-type: none"> <li>Análisis dinámico</li> <li>Desempeño, carga y estrés</li> <li>Monitoreo</li> </ul>
	<b>Tipos de herramientas para aplicaciones específicas</b>

## Beneficios y riesgos

BENEFICIOS		RIESGOS	
	<ul style="list-style-type: none"> <li>• Reducción de trabajo repetitivo</li> <li>• Reúne consistencia y repetitividad</li> <li>• Evaluación de objetivos</li> <li>• Fácil acceso sobre las pruebas</li> </ul>		<ul style="list-style-type: none"> <li>• Expectativas irreales de la herramienta</li> <li>• Subestimar tiempo, costo y esfuerzo para alcanzar el objetivo</li> <li>• Subestimar el esfuerzo requerido para mantener la herramienta</li> <li>• Sobre confianza en la herramienta</li> </ul>

## Consideraciones especiales

	<b>Herramientas de ejecución de pruebas:</b> Un pequeño cambio en el software invalida cientos de scripts de pruebas
	<b>Herramientas de desempeño:</b> Se simulará el diseño de carga
	<b>Herramientas de análisis estático</b> Son muy útiles para los desarrolladores, sobre todo durante la compilación
	<b>Herramientas de gestión de pruebas</b> Generan gran cantidad de información, sin embargo, esta requiere ser sintetizada

## Introducir una herramienta a la organización

Considerar factores como:

Evaluar la madurez de la organización	Proyecto piloto
Identificar las áreas involucradas	Aprender más de la herramienta
Evaluar las herramientas contra requerimientos y objetivos bien definidos	Realizar una planeación interna para la implementación
Realizar una prueba de concepto para saber si el producto trabajara como se espera	Saber como se adapta la herramienta a los procesos y documentación existentes
Evaluar al vendedor	