

INGENIERIA DE SISTEMAS

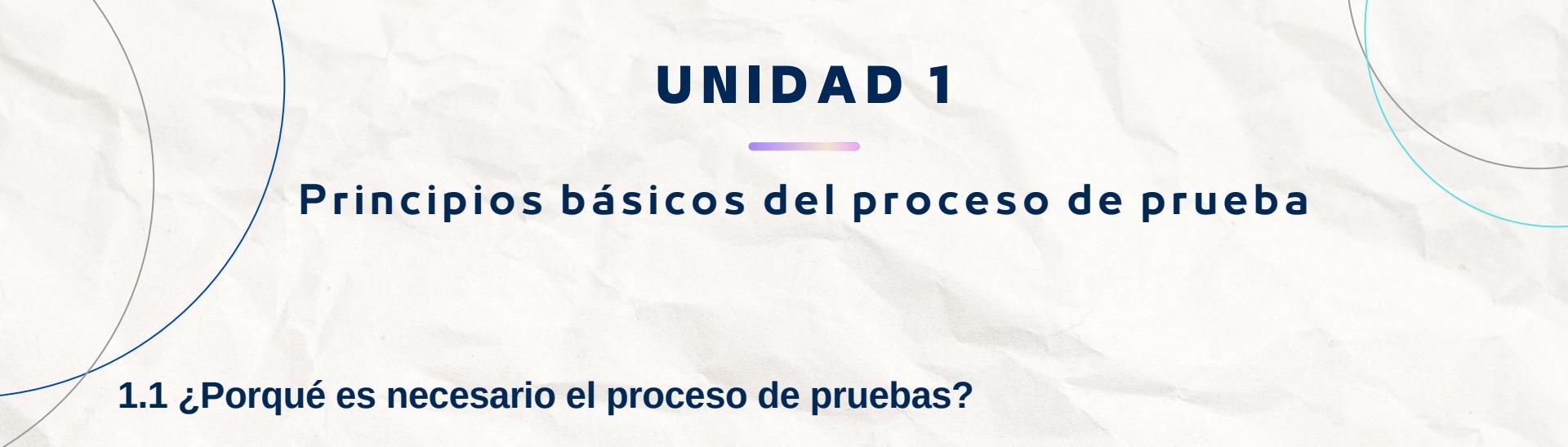
Calidad de Software (P)



Lucas Argañaraz

RESUMEN UNIDAD 1, 2 Y 3

UNIDAD 1



Principios básicos del proceso de prueba

1.1 ¿Porqué es necesario el proceso de pruebas?

1.2 ¿Qué es el testing?

1.3 ¿Qué habilidades necesitarás como Tester?

1.4 ¿En qué consiste el proceso de pruebas?

1.5 Siete principios básicos del proceso de pruebas

1.6 Responsabilidades del Tester o QA en un equipo de desarrollo de software

1.7 Proceso básico de pruebas

1.8 La psicología de las pruebas

TEMA 1.1

¿Porqué es necesario el proceso de pruebas?

1.1.1 Contexto de los sistemas de software

1.1.2 Causas de los defectos de software

1.1.3 Función del proceso de pruebas en el desarrollo, mantenimiento y operaciones de software

1.1.4 ¿Con cuántas pruebas es suficientes?

1.1.1 Contexto de los sistemas de software

- El software se utiliza en distintos tipos de aplicaciones: comerciales, productos de consumo, bancos, etc.
- A muchos nos ha pasado que un software no funcione según lo previsto.
- No todos los sistemas de software llevan el mismo nivel de riesgo.
- La cantidad de pruebas realizadas depende de los riesgos involucrados.

Algunos problemas pueden ser bastante triviales, pero otros pueden ser costosos y perjudiciales.

1.1.2 Causas de los defectos de software

- **Error /Mistake:** Acción humana que produce un resultado incorrecto
- **Defecto /Bug:** Desperfecto en un componente o sistema que puede causar que el componente o sistema falle en desempeñar las funciones requeridas.
- **Fallo /Failure:** Manifestación física o funcional de un defecto.

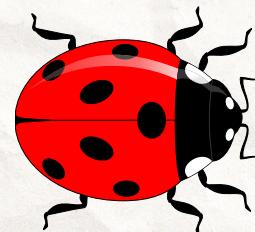
“Un error introduce un defecto, un defecto causa un fallo”

CAUSAS DE LOS DEFECTOS

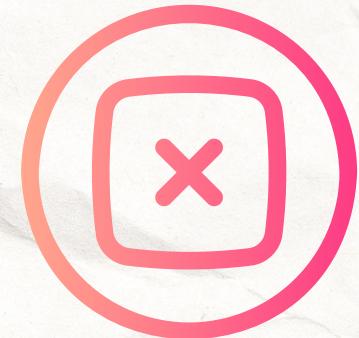


Un desarrollador comete un error (mistake)...

...que inyecta una falla (bug)
en el software



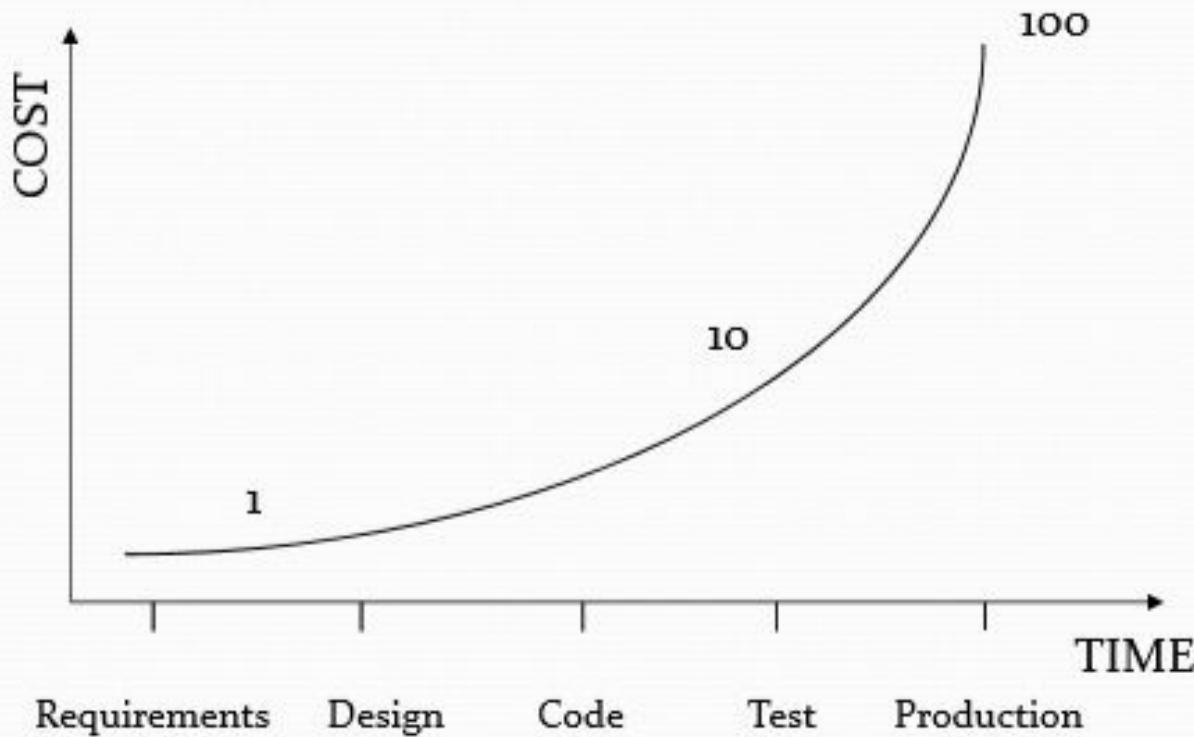
...que hace que un
componente o sistema falle



“No todos los defectos dan lugar a fallas, algunos pueden permanecer en el software y es posible que nunca nos fijemos en ellos”

COSTO DE LOS DEFECTOS

El costo de encontrar y corregir defectos aumenta considerablemente a lo largo del ciclo de vida



1.1.4 ¿Con cuántas pruebas es suficientes?

Niveles de riesgo:

- Riesgos técnicos
- De seguridad
- Comerciales

Limitaciones del proyecto: Tiempo y presupuesto

“Cuando esté seguro de que el sistema funciona correctamente”

TEMA 1.2

¿Qué es el testing?

Es toda una disciplina en la ingeniería de software que permite tener procesos, métodos de trabajo y herramientas para identificar defectos en el software alcanzando un proceso de estabilidad del mismo. El Testing no es una actividad que se piensa al final del desarrollo del software, va paralelo a este. Permite que lo que se está construyendo, se realice de manera correcta de acuerdo a lo que necesita un usuario final.

La definición del ISTQB (International Software Testing Qualification Board :

- Es un proceso
- Está presente en toda las actividades del ciclo de vida
- Es estático y dinámico
- Relacionado con la planificación, preparación y evaluación de los productos de software
- Para determinar si cumplen los requisitos especificados
- Para demostrar que son aptos para el propósito
- Para detectar defectos

TEMA 1.3

¿Qué habilidades necesitarás como Tester?

- **Pensamiento lógico:** Debe saber como desglosar un sistema en unidades mas pequeñas para poder crear casos de pruebas.
- **Muy buena comunicación:** Excelente comunicación verbal y escrita para comunicar los errores y documentarlos.
- **Atención a los detalles:** Se requiere ser curioso y tener un pensamiento crítico con atención a los detalles desde el punto de vista del usuario final .
- **Ser organizado y metódico:** Esto es clave para poder ejecutar los casos de prueba en un orden y poder encontrar la mayor cantidad de errores.
- **Apasionado por la tecnología:** Todo Tester tiene que tener un gusto por la información, ser curioso y creativo.
- **Paciencia y persistencia:** El desarrollo de software es un proceso que requiere flexibilidad y muchas paciencia.

RESUMEN



- Los fallos de software pueden causar importantes prejuicios.
- La calidad del software es la suma de los atributos que se refieren a la capacidad del software de satisfacer un conjunto de requisitos dados.
- El aseguramiento de la calidad constructivo se ocupa de la prevención de defectos.
- El aseguramiento de la calidad analítico se ocupa de detectar y corregir defectos.
- Los atributos de la calidad funcionales y no funcionales definen la calidad total del sistema.
- Cada prueba debe contar con un criterio para la finalización de la prueba (criterio de salida).
- Al alcanzar el criterio de salida de pruebas concluyen las actividades del proceso de pruebas.
- **Los testers buscan fallos** en el sistema e informan sobre los mismo (proceso de prueba – “testing”).
- **Los desarrolladores buscan defectos** y los corrigen (depuración – “debugging”).

TEMA 1.5

Siete principios básicos del proceso de pruebas

1.5.1 Las pruebas demuestran la presencia de defectos

PRINCIPIO # 1.

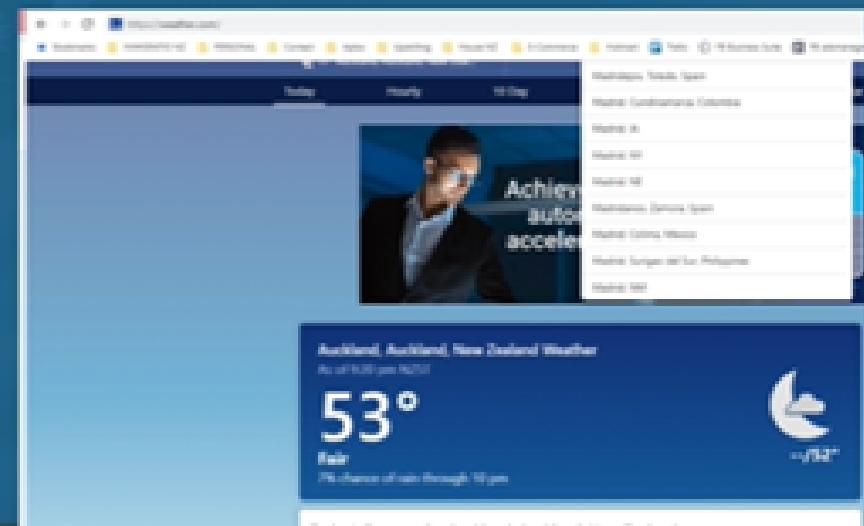
Ejecutar pruebas nos muestra la presencia de defectos... pero no pueden probar que NO los hay.



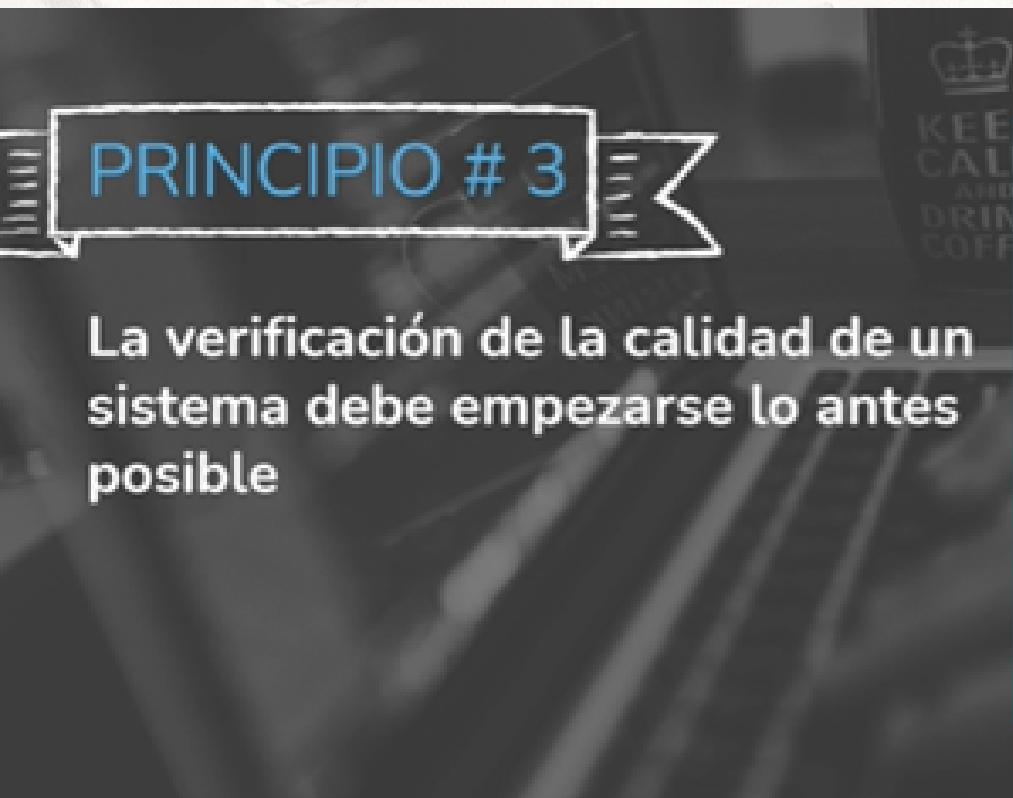
1.5.2 Las pruebas exhaustivas no existen

PRINCIPIO # 2

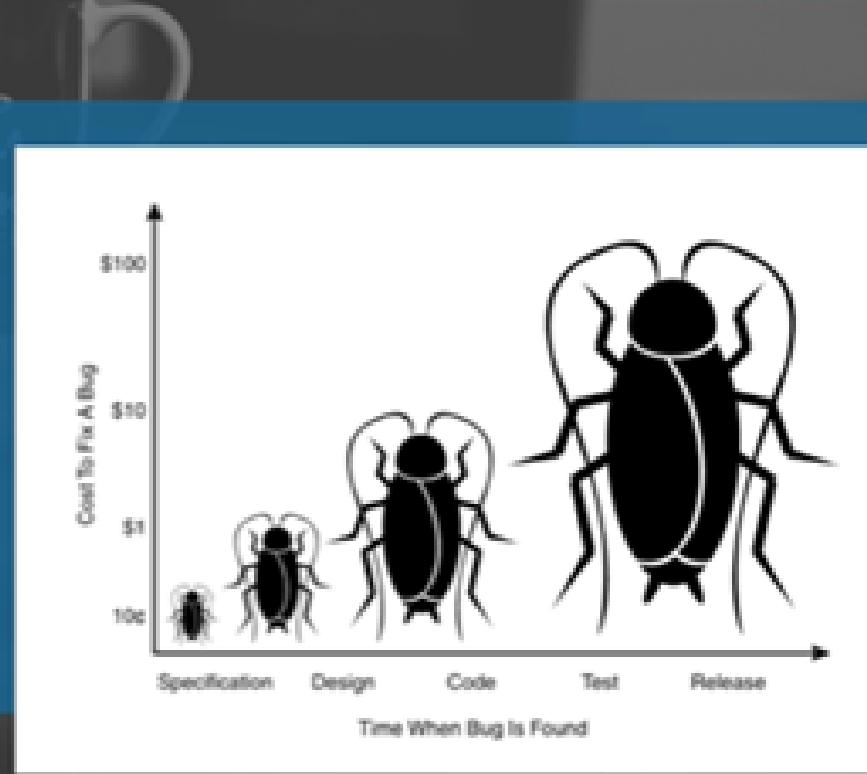
El testing exhaustivo es imposible



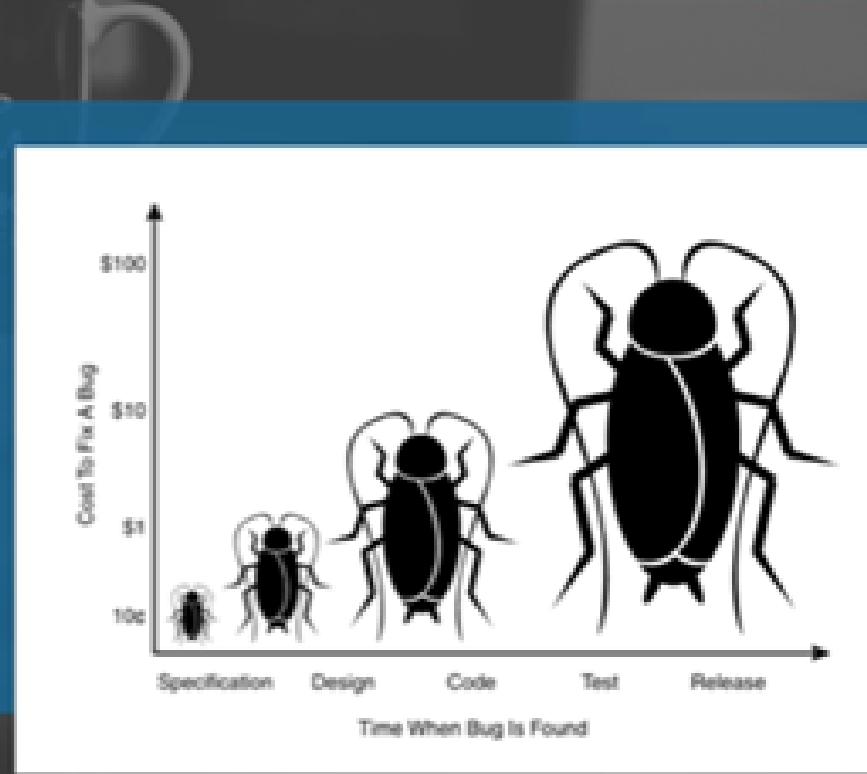
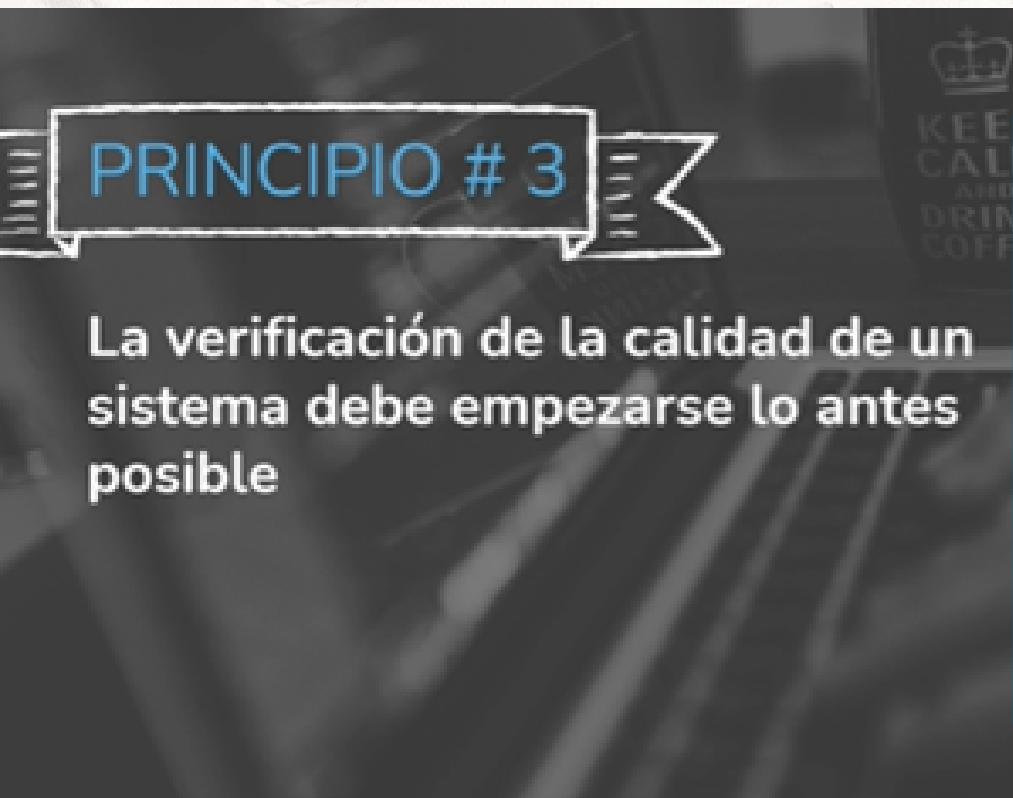
1.5.3 Pruebas tempranas



La verificación de la calidad de un sistema debe empezarse lo antes posible



1.5.3 Pruebas tempranas



1.5.4 Agrupación de defectos

PRINCIPIO # 4

La mayoría de defectos relevantes suelen concentrarse en un grupo muy determinado de módulos de nuestro producto.



1.5.5 Paradoja del pesticida

PRINCIPIO # 5

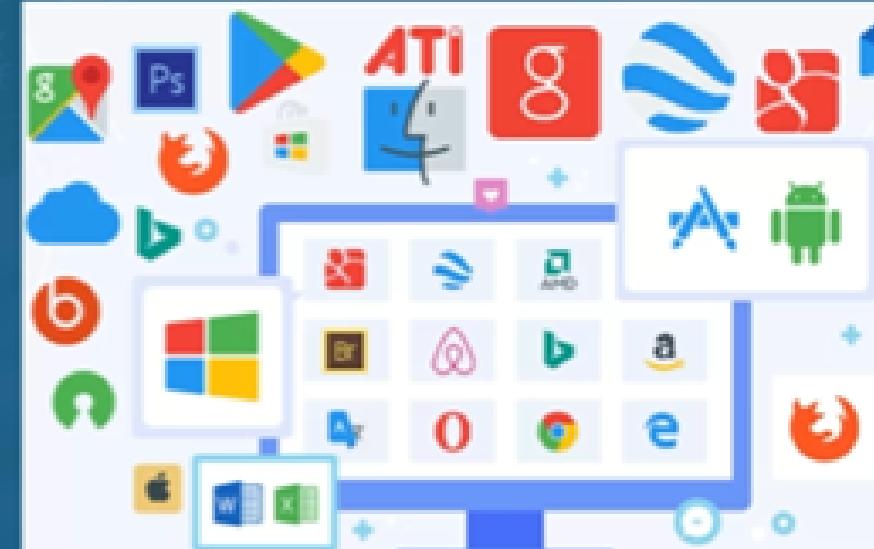
La paradoja del pesticida



1.5.6 Las pruebas dependen del contexto

PRINCIPIO # 6

El testing es totalmente dependiente del contexto



1.5.7 Falacia de ausencia de errores

PRINCIPIO # 7

La falacia de ausencia de errores



TEMA 1.5

Siete principios básicos del proceso de pruebas

1. Las pruebas demuestran la presencia de defectos
2. Las pruebas exhaustivas no existen
3. Pruebas tempranas
4. Agrupación de defectos
5. Paradoja del pesticida
6. Las pruebas dependen del contexto
7. Falacia de ausencia de errores

RESUMEN



- Las pruebas pueden ayudar a detectar defectos en el software, sin embargo, las mismas no pueden demostrar la ausencia de defectos.
- Salvo en casos triviales las pruebas exhaustivas son imposibles, las pruebas de muestras son necesarias.
- Las pruebas tempranas ayudan a reducir costos dado que los defectos descubiertos en fases tempranas del proceso de software son corregidos con menor esfuerzo.
- Los defectos se presentan agrupados. El encontrar un defecto en una ubicación determinada significa que probablemente se encontrará otro defecto a su alrededor.
- Repetir pruebas idénticas no genera nueva información.
- Cada entorno particular determina la forma en la cual se ejecutarán/desarrollarán las pruebas.
- Un software libre de errores no implica que sea adecuado para el uso.

TEMA 1.6

Responsabilidades del Tester o QA en un equipo de desarrollo de software

- Diseñar un plan de pruebas
- Definir los casos de prueba con base a los requisitos
- Gestionar el ambiente y los datos de prueba
- Ejecutar los casos de prueba
- Documentar las pruebas realizadas
- Reportar los errores encontrados y realizar seguimiento para su corrección y revalidación
- Participar en las reuniones de seguimiento diarias y todas las de scrum
- Realizar informes de calidad del producto
- Ayudar a resolver dudas a los Analistas de requisitos o POs
- Ayudar a los programadores a replicar los errores y a investigar su solución
- Implementar prácticas de aseguramiento de calidad para prevenir errores en el código

TEMA 1.7

Proceso básico de pruebas

1.7.1 Planificación y control de pruebas

1.7.2 Análisis y diseño de pruebas

1.7.3 Implementación y ejecución de pruebas

1.7.4 Evaluación de los criterios de salida e informes

1.7.5 Actividades de cierre de pruebas

1.7.1 Planificación y control de pruebas

- La planificación de las pruebas es la actividad de verificar que entendemos las metas y objetivos de los clientes, las partes interesadas, el proyecto y los riesgos que se pretenden abordar.
- El control de las pruebas es la actividad continua de comparar el progreso real con el plan e informar sobre el estado actual de las pruebas incluidas las desviaciones del plan.

1.7.2 Análisis y diseño de pruebas

Tareas principales

- Revisar la base de pruebas
- Evaluar cuan testeable es la base de prueba y los objetos de prueba
- Identificar y priorizar las condiciones de prueba en base al análisis
- Diseñar y priorizar los casos de prueba de alto nivel
- Identificar los datos de prueba
- Diseñar la configuración del entorno de pruebas e identificar cualquier infraestructura y herramienta necesaria

1.7.3 Implementación y ejecución de pruebas

Tareas principales

- Finalizar, implementar y priorizar los casos de prueba
- Desarrollar y priorizar procedimientos de prueba
- Crear juegos de pruebas a partir de los procedimientos de prueba para lograr una ejecución de pruebas eficientes
- Verificar que el entorno de pruebas ha sido correctamente configurado
- Verificar y actualizar una trazabilidad bidireccional entre la base de pruebas y los casos de pruebas
- Ejecutar los procedimientos de prueba manualmente o recurriendo a herramientas de ejecución de pruebas
- Registrar los resultados de la ejecución de las pruebas, las identidades y las versiones del software probado
- Comparar los resultados reales con los resultados esperados
- Reportar las discrepancias en forma de incidencias y analizarlas con vistas a establecer sus causas
- Repetir las actividades de pruebas como resultado de una medida adoptada para cada discrepancia

1.7.4 Evaluación de los criterios de salida e informes

Tareas principales

- Comprobar los registros de pruebas con los criterios de salida previstos en la planificación
- Evaluar si se requieren más pruebas o si deberían modificarse los criterios de salida
- Elaborar un resumen de las pruebas para las partes interesadas

1.7.5 Actividades de cierre de pruebas

Tareas principales

- Comprobar cuáles de los productos entregables previstos han sido efectivamente entregados
- Cerrar los informes de incidencias o aportar modificaciones a aquellos que siguen abiertos
- Documentar la aceptación del sistema
- Finalizar y archivar los productos de soporte de prueba, el entorno de pruebas y la infraestructura de pruebas para su posterior uso
- Entregar los productos de soporte de prueba a la organización de mantenimiento
- Analizar las lecciones aprendidas para determinar los cambios necesarios en futuras versiones y proyectos
- Utilizar la información recopilada para mejorar la madurez de las pruebas

RESUMEN

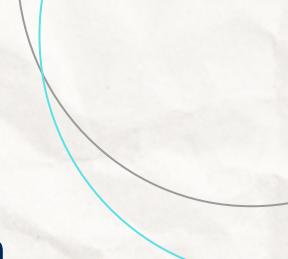


- **Planificación de pruebas** “Test Planning” abarca actividades como la definición de la estrategia de pruebas para todas las fases, así como la planificación de los recursos (tiempo, personal, máquinas).
- **Diseño de pruebas** (Especificación) abarca el diseño de casos de prueba y sus resultados esperados.
- **Ejecución de pruebas** abarca la definición de los datos de prueba, la ejecución de las pruebas y la comparación de resultados.
- **Evaluación de pruebas** y generación de informes abarca la evaluación del criterio de salida y el registro de los resultados de pruebas en forma escrita.
- **Control de pruebas** consiste en el control de las actividades que cubren todas las fases del proceso de pruebas.

UNIDAD 2



Pruebas durante todo el ciclo de vida del software



2.1 Modelo de desarrollo de software

2.2 Niveles de pruebas

2.3 Tipos de pruebas

2.4 Mantenimiento del software

TEMA 2.1

Modelo de desarrollo de software

2.1.1 Principios de todos los modelos

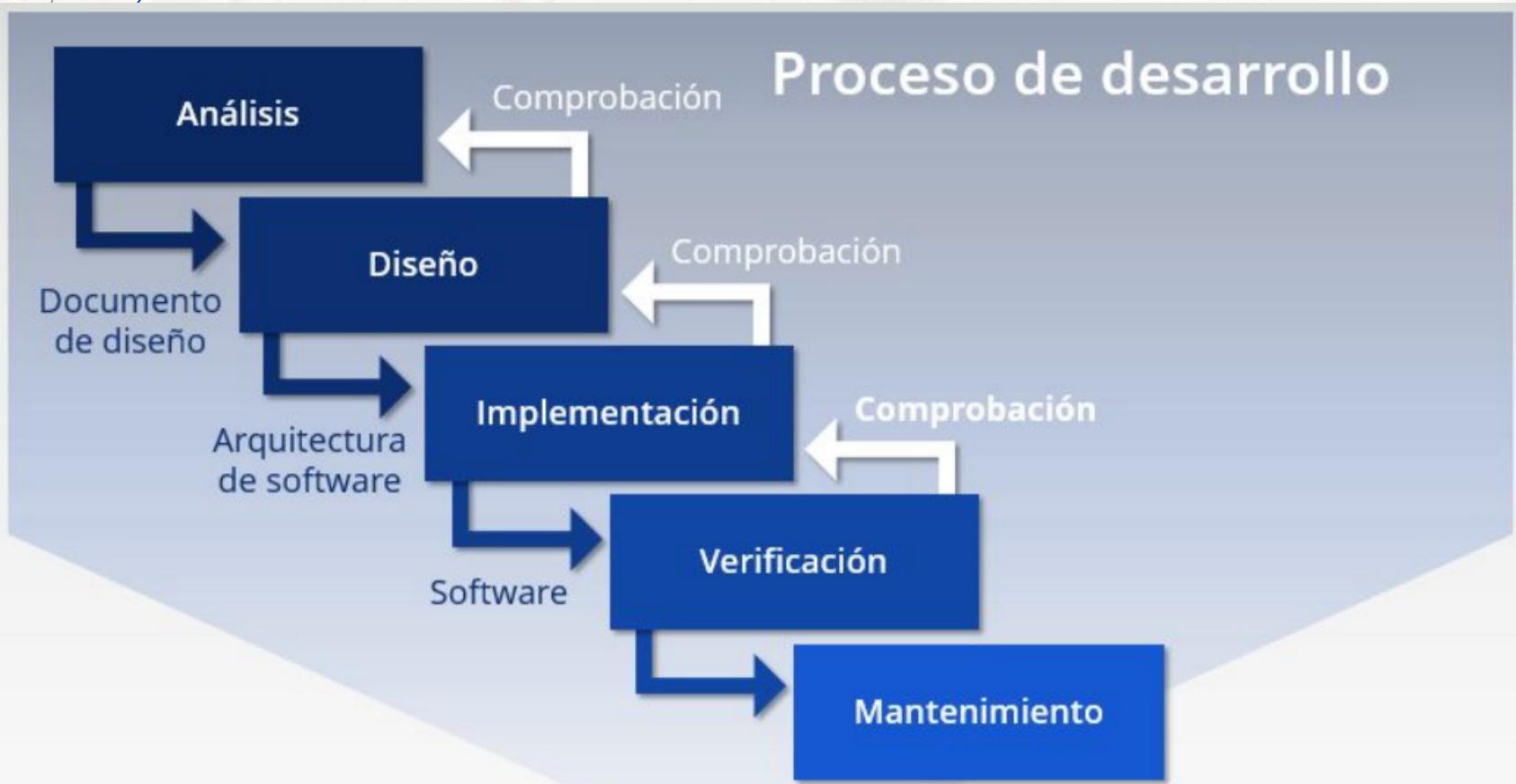
2.1.2 Modelos de desarrollo iterativo-incremental

2.1.3 Pruebas en un Modelo de Ciclo de Vida

2.1.1 Principios de todos los modelos

- Cada actividad de desarrollo debe ser probada
- Cada nivel de prueba debería ser probado de forma específica
- El proceso de pruebas comienza con mucha antelación a la ejecución de pruebas
- Tan pronto como el desarrollo comienza puede comenzar la preparación de las pruebas correspondientes

Modelo de Desarrollo secuencial o en cascada

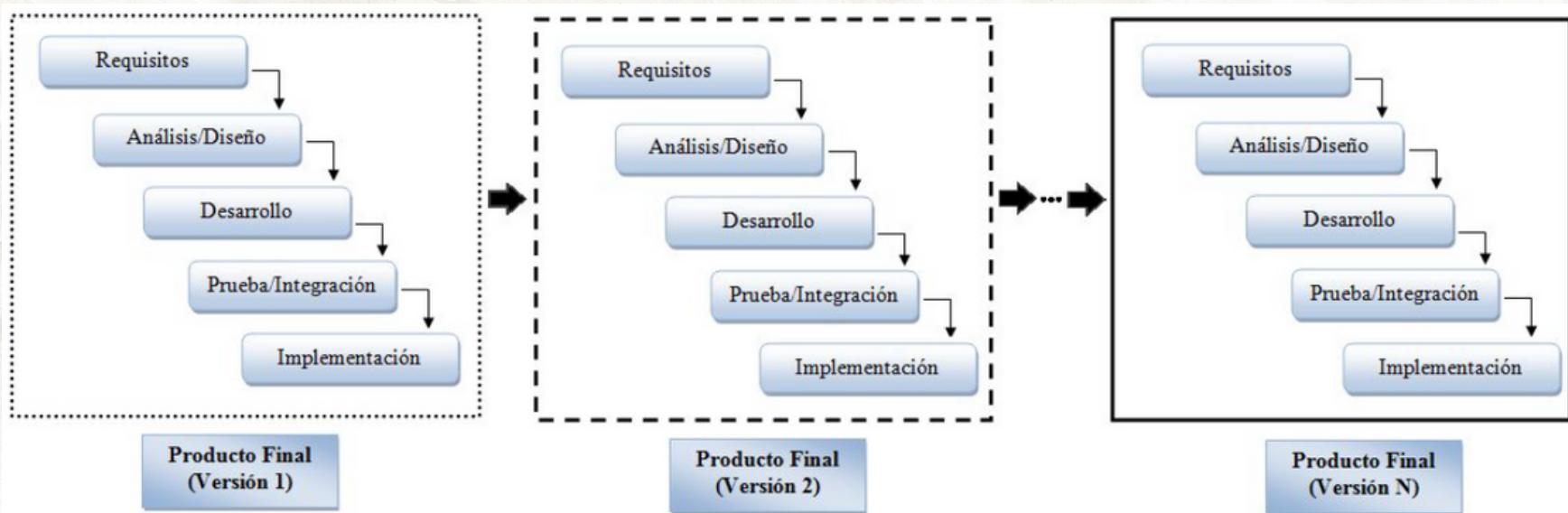


Características de los modelos iterativos

- Cada iteración contribuye con un característica adicional del sistema a desarrollar
- Cada iteración puede ser probada por separado
- Las pruebas de regresión y la automatización de pruebas son elementos de relevancia
- En cada iteración, la verificación y la validación se pueden efectuar por separado

2.1.2 Modelos de desarrollo iterativo-incremental

- Las entregas se dividen en incrementos
- Cada incremento añade una nueva funcionalidad
- El incremento producido por una iteración puede ser probado en varios niveles como parte de su desarrollo

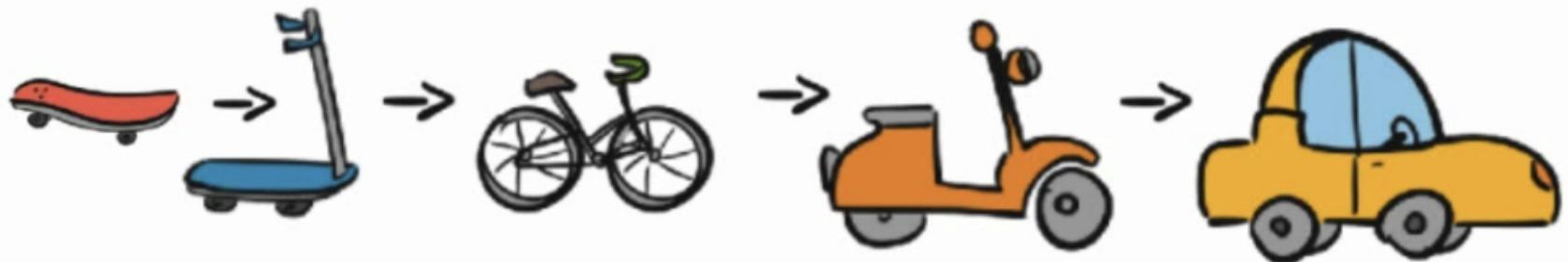


Iterativo vs Incremental

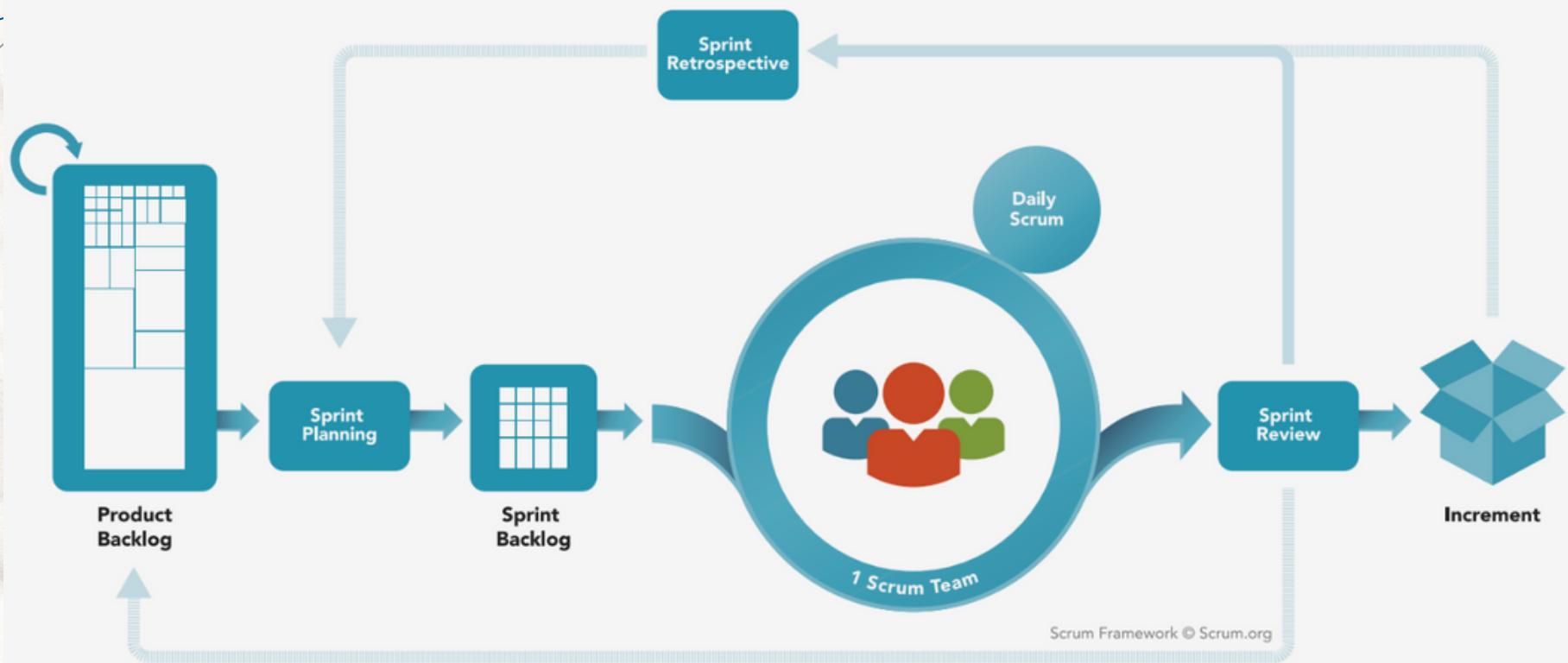


MÓDULO ITERATIVO

MÓDULO ITERATIVO E INCREMENTAL



Desarrollos Agile – Scrum Framework



2.1.3 Pruebas en un Modelo de Ciclo de Vida

- Para cada actividad de desarrollo existe una actividad de prueba
- Cada nivel de prueba tiene objetivos de prueba específico
- Los procesos de análisis y diseño de las pruebas para un nivel de prueba deben inicializarse durante la actividad de desarrollo correspondiente
- Los testers deben iniciar su participación en la revisión de documentos en cuanto haya borradores disponibles

TEMA 2.2

Niveles de prueba

2.2.1 Pruebas de unitarias o de componentes

2.2.2 Pruebas de integración

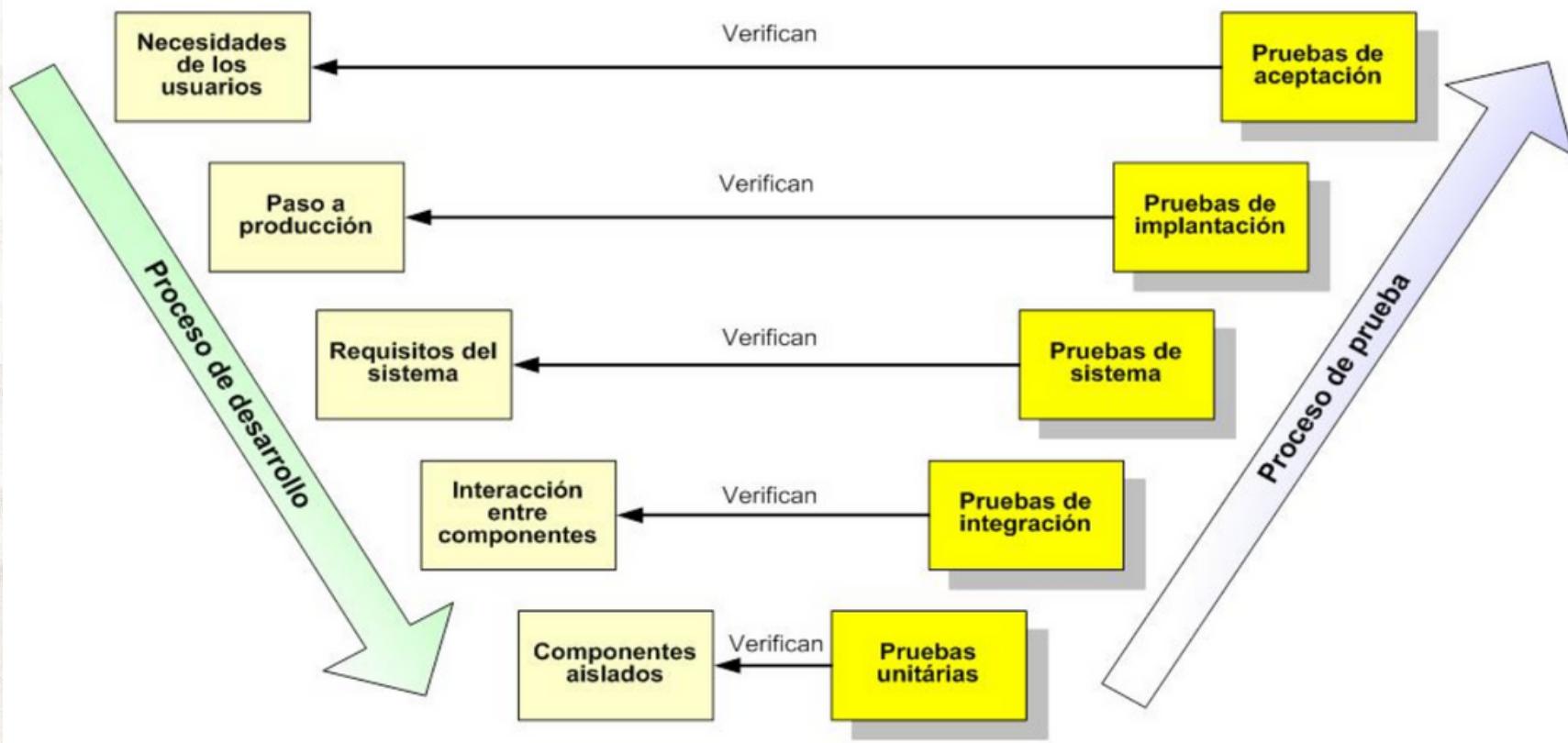
2.2.3 Pruebas de sistema

2.2.4 Pruebas de implantación o de producción

2.2.5 Pruebas de aceptación

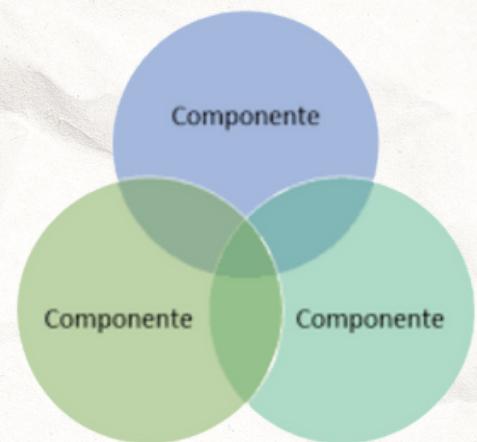
Cada nivel de desarrollo tiene su correspondiente nivel de pruebas

Niveles de prueba



2.2.1 Pruebas unitarias o de componente

Las pruebas unitarias son de muy bajo nivel y se realizan cerca de la fuente de la aplicación. Consisten en probar métodos y funciones individuales de las clases, componentes o módulos que usa tu software. En general, las pruebas unitarias son bastante baratas de automatizar y se pueden ejecutar rápidamente mediante un servidor de integración continua.



Base de pruebas:

- Especificación del componente
- Diseño de detalle
- Código

Objetos de prueba típicos:

- Componentes
- Programas
- Conversión de datos/programas de migración

2.2.2 Pruebas de integración

- Las pruebas de integración verifican que los distintos módulos o servicios utilizados por tu aplicación funcionan bien en conjunto. Por ejemplo, se puede probar la interacción con la base de datos o asegurarse de que los microservicios funcionan bien en conjunto y según lo esperado. Estos tipos de pruebas son más costosos de ejecutar, ya que requieren que varias partes de la aplicación estén en marcha.



Base de pruebas:

- Diseño de software y sistema
- Arquitectura
- Flujo de trabajo
- Casos de usos

Objetos de prueba típicos:

- Implementación de BD de subsistemas
- Infraestructura
- Interfaces

Configuración del sistema:

- Datos de configuración

2.2.3 Pruebas funcionales o de sistema



- Las pruebas funcionales se centran en los requisitos empresariales de una aplicación. Solo verifican el resultado de una acción y no comprueban los estados intermedios del sistema al realizar dicha acción. A veces, se confunden las pruebas de integración con las funcionales, ya que ambas requieren que varios componentes interactúen entre sí. La diferencia es que una prueba de integración puede simplemente verificar que puedes hacer consultas en la base de datos, mientras que una prueba funcional esperaría obtener un valor específico desde la base de datos, según dicten los requisitos del producto.

Base de pruebas:

- Especificación de requisitos del sistema y software
- Casos de Uso
- Especificaciones funcionales
- Informes de análisis de riesgos

Objetos de prueba típicos:

- Manuales de sistema, usuario y funcionamiento
- Configuración del sistema

2.2.4 Pruebas de implantación o de producción

El objetivo de las pruebas de implantación es comprobar el funcionamiento correcto del sistema integrado de hardware y software en el entorno de operación, y permitir al usuario que, desde el punto de vista de operación, realice la aceptación del sistema una vez instalado en su entorno real y en base al cumplimiento de los requisitos no funcionales especificados.



2.2.5 Pruebas de aceptación

Las pruebas de aceptación son pruebas formales que verifican si un sistema satisface los requisitos del cliente. Requieren que se esté ejecutando toda la aplicación durante las pruebas y se centran en replicar las conductas de los usuarios. Sin embargo, también pueden ir más allá y medir el rendimiento del sistema y rechazar cambios si no se han cumplido determinados objetivos.



Base de pruebas:

- Requisitos del usuario
- Requisitos del sistema
- Casos de uso
- Procesos de negocio
- Informes de análisis de riesgos

Objetos de prueba típicos:

- Procesos de negocio en sistema integrado
- Procesos operativos y de mantenimiento
- Procedimientos de usuario
- Formularios
- Informes

TEMA 2.3

Tipos de prueba

2.3.1 Pruebas funcionales

2.3.2 Pruebas no funcionales

2.3.1 Pruebas funcionales

Objetivo:

- Su objetivo se basa en comprobar si las funcionalidades del desarrollo se realizaron siguiendo las especificaciones y requisitos del cliente, así como sus necesidades iniciales. De esta manera se detectan las posibles fallas de las fases anteriores.



2.3.1 Pruebas funcionales

Ejemplos:

- Exploratorias
- De Regresión
- Pruebas de compatibilidad de entorno
- Libres o Free Testing
- De Humo o Smoke Tests
- Pruebas de Mono
- De Sanidad

- **Exploratorias:**

Consiste en ejecutar las pruebas a medida que se piensa en ellas, sin gastar demasiado tiempo en prepararlas o explicarlas, confiando en los instintos.

- **De Regresión:**

Evita que al introducir nuevos cambios en un software se obtengan comportamientos no deseados o defectos en otros módulos no modificados.

- **Pruebas de compatibilidad de entorno:**

Son pruebas en las que se ejecuta el mismo producto en diferentes entornos, para chequear que funcionalmente se comportan igual.

- **Libres o Free Testing:**

Son las pruebas que se ejecutan sin un Test Plan determinado. Comparte la misma filosofía que el testing exploratorio.

- **De Humo o Smoke Tests:**

Es una revisión rápida inicial de la versión de software entregada por desarrollo donde se verificará de forma general sin entrar en detalle las principales funcionalidades del mismo y se asegurará que no tiene defectos que interrumpa el funcionamiento básico del mismo, para que el equipo de testing pueda seguir probando entrando más en detalle.

- **Pruebas de Mono:**

Son las pruebas que se hacen sin atender mucho al funcionamiento teórico del producto, simplemente consiste en navegar por los distintos caminos del software sin un orden determinado e intentando ejecutar todas las opciones posibles.

- **De Sanidad:**

Es un conjunto de pruebas que se ejecutan para comprobar que todo funciona correctamente después de alguna intervención o modificación. Es similar a las pruebas de regresión, pero suelen ser menos exhaustivas.

2.3.2 Pruebas no funcionales

Objetivo:

Evalúa la forma en la que un desarrollo y/o aplicativo opera, haciendo foco en sus atributos y calidad. Asegurando así una buena experiencia del usuario, más allá de las funcionalidades que debería tener.



2.3.2 Pruebas no funcionales

Ejemplos:

- De Carga o Rendimiento
- Recuperación o vuelta atrás
- Referentes a la Instalación
- Pruebas Estructurales
- Pruebas de configuración

- **De Carga o Rendimiento:**

Verifica el comportamiento del sistema frente a un crecimiento de carga de consultas, accesos, etc.

- **Recuperación o vuelta atrás:**

Verifica los procedimientos de recuperación del sistema ante fallos.

- **Referentes a la Instalación:**

Se verifica todo lo relacionado con el despliegue del producto: documentación, instalación de software, configuración pos-instalación, etc.

- **Pruebas Estructurales:**

Se realizan para ejecutar la estructura interna del producto intentando ejecutar todos los caminos posibles del código. Se refieren a las pruebas de caja blanca.

- **Pruebas de configuración:**

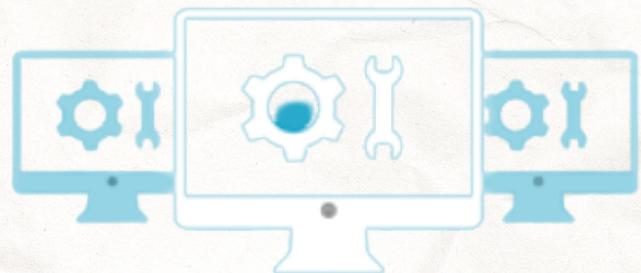
Consiste en la realización de pruebas sobre el producto frente a las diferentes configuraciones posibles.

TEMA 2.4

Mantenimiento del software

El mantenimiento de software cubre dos campos:

- **Mantenimiento:** corrección de errores que han sido parte de la versión inicial del software.
- **Extensión/Ampliación:** adaptaciones como resultado de un cambio del entorno o nuevos requisitos del cliente.



UNIDAD 3

Técnicas estáticas y dinámicas

3.1 Técnicas estáticas y técnicas dinámicas

3.2 Proceso de revisión

3.3 Análisis estático con herramientas

TEMA 3.1

Técnicas estáticas

Las pruebas estáticas:

El objetivo es la revisión de productos de trabajo como documentos de requerimientos, casos de prueba, planes de prueba, código, guías de usuario.

- No requieren que el software este en ejecucion para realizar las pruebas.
- Se enfocan en la prevención de defectos y en la detección temprana de los mismos.
- Permite mejoras en la productividad
- Reduce los tiempos de desarrollo
- Ahorra tiempo en las pruebas y en el dinero invertido de esas pruebas
- Mejora la comunicación

Ejemplo de productos de trabajo que pueden recibir pruebas estáticas:

- Especificaciones de requisitos
- Especificaciones de diseño
- El código
- Planes de prueba
- Especificaciones de prueba
- Casos de Prueba
- Guías de usuario
- Páginas Web

Defectos típicos fáciles de localizar en revisiones:

- Desviaciones de los estándares
- Defectos de requisito
- Defectos de diseño
- Mantenibilidad insuficiente
- Especificaciones de interfaz incorrectas

TEMA 3.1

Técnicas dinámicas

Las pruebas dinámicas:

Tienen como objetivo asegurar que el software se comporte de acuerdo con los requerimientos del negocio mediante la realización de pruebas funcionales y no funcionales.

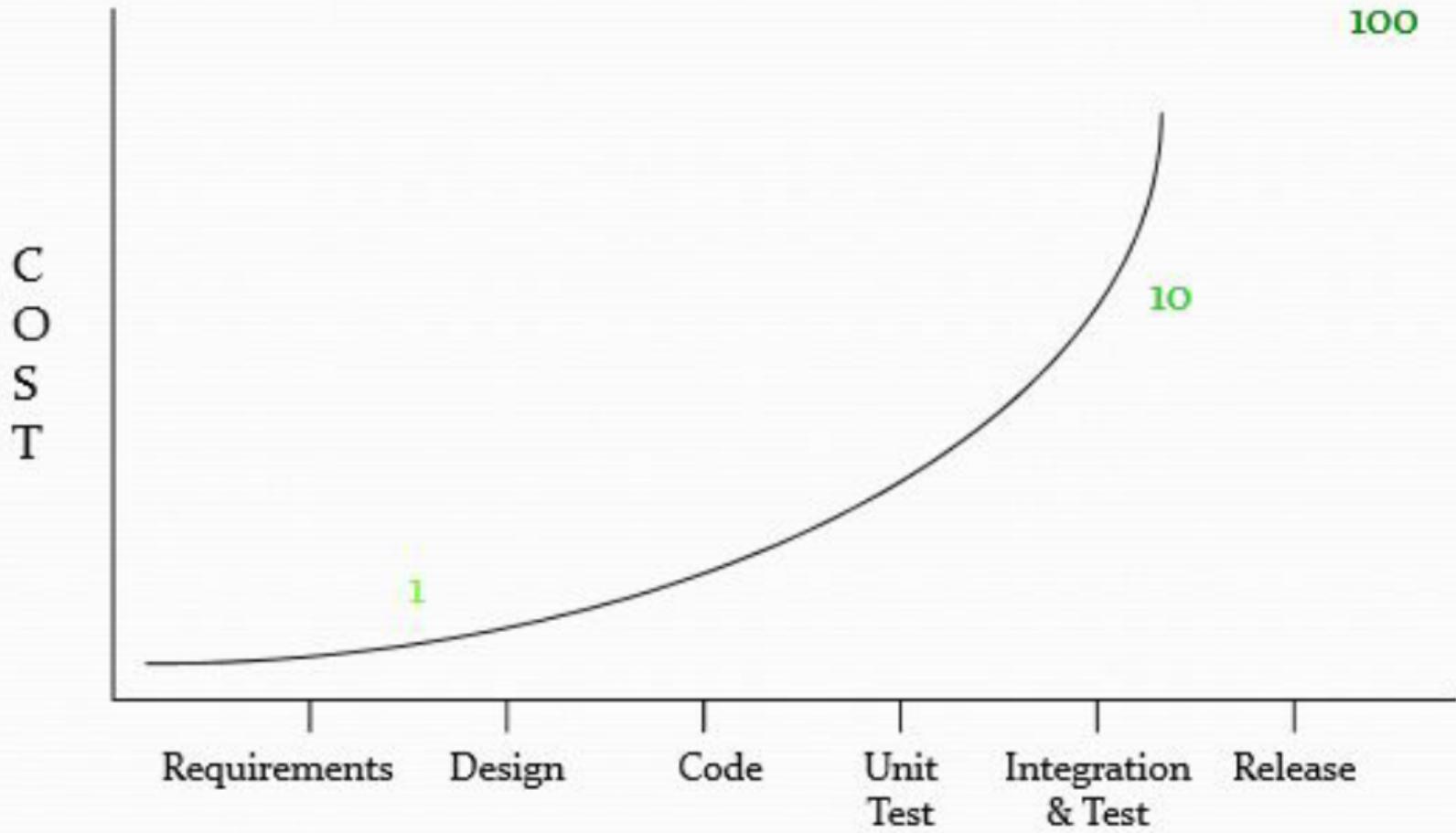
- Son aquellas que se realizan mientras el código está en ejecución
- Estas pruebas se enfocan en la detección y confirmación de la corrección de defectos en el software
- Se realizan en una etapa más tardía que las pruebas estáticas, por lo cual, los defectos encontrados en estas son más costosos

RESUMEN

Pruebas estáticas vs pruebas dinámicas

No ejecución	En ejecución
Las pruebas se realizan a productos de trabajo como documentos de requerimientos, casos de prueba, planes de prueba y código.	Las pruebas se realizan mientras el código está en ejecución.
Prevención	Detección
Estas pruebas se enfocan en la prevención de defectos.	Estas pruebas se enfocan en la detección de defectos.
Pruebas tempranas	Pruebas tardías
Estas pruebas pueden ser realizadas desde las primeras etapas del ciclo de vida del software.	Estas pruebas se realizan cuando el código es desplegado a un ambiente de pruebas.
Técnicas	Técnicas
Se utilizan técnicas como: <ul style="list-style-type: none">• Revisión técnica• Inspección• Revisión de código	Se utilizan tipos de pruebas: <ul style="list-style-type: none">• Funcionales• No funcionales

Relación de Costo de arreglar un Falla



TEMA 3.2

Proceso de revisión

3.2.1 Fases de una revisión

3.2.2 Funciones y responsabilidades

3.2.3 Tipos de revisiones (IEEE 1028)

3.2.1 Fases de una revisión

- Fase de planificación (“Planning”)
- Preparación de la organización e inicio (“Kick-off”)
- Preparación individual (“Preparation”)
- Reunión de revisión (“Review meeting”)
- Reconstrucción – Repetición del trabajo (“Rework”)
- Seguimiento (“Follow up”)

Actividades de una revisión formal

1. Planificar (“Planning”)

- Definir los criterios de revisión
- Seleccionar al personal
- Asignar funciones

2. Definir los criterios de entrada y salida para más tipos de revisión formal

- Seleccionar qué partes de los documentos deben revisarse

Actividades de una revisión formal

3. Inicio (“Kick-off”)

- Repartir los documentos
- Explicar los objetivos, procesos y documentación a los participantes

4. Comprobar los criterios de entrada

Actividades de una revisión formal

5. Preparación individual (“Preparation”)

- Prepararse para la reunión de revisión repasando los documentos

6. Prestar especial atención a posibles defectos, preguntas y comentarios

7. Examen/evaluación/registro de los resultados

- Debatir o registrar, mediante los documentados o actas
- Prestar especial atención a los defectos, hacer recomendaciones sobre cómo manejar los defectos,
- Tomar decisiones al respecto

Actividades de una revisión formal

8. Examinar/evaluar y registrar durante reuniones físicas o de seguimiento de los grupos comunicaciones electrónicas (“Review meeting”)
9. Adaptar
10. Corregir los defectos detectados (“Rework”)
 - Registrar el estado actualizado de los defectos
11. Hacer un seguimiento (“Follow up”)
 - Comprobar que los defectos han sido tratados
 - Recopilar métricas
12. Comprobar los criterios de salida

3.2.2 Funciones y responsabilidades

- Director (“Manager”): inicia la revisión, decide respecto a los participantes y asigna recursos
- Moderador (“Moderator”): dirige la reunión, hace de mediador, concluye resultados
- Autor (“Author”): expone su trabajo a la crítica, lleva a cabo los cambios recomendados
- Revisores (“Reviewer”): detecta defectos, desviaciones, áreas problemáticas
- Registrador (“Recorder/Writer”): documenta todos los puntos identificados

3.2.3 Tipos de revisiones (IEEE 1028)

- Revisión de gestión
- Revisión técnica (“Technical review”)
- Inspección (“Inspection”)
- Revisión guiada (“Walkthrough”)
- Auditoria

Revisión de gestión

Es una evaluación sistemática de un producto o proceso de software realizado por la administración que monitorea el progreso.

Objetivos principales:

- Determina el estado de los planes y horarios
- Monitorea el progreso
- Confirma los requisitos y su sistema de asignación
- Identifican la coherencia y las desviaciones de los planes
- El examen no necesita abordar todos los aspectos del producto o proceso de software.

Revisión técnica

Es una evaluación sistemática de un producto de software por un equipo de personal calificado para determinar su idoneidad para el uso previsto e identificar discrepancias con las especificaciones y estándares.

Objetivos principales:

- Permite debatir, tomar decisiones, evaluar alternativas, encontrar defectos y resolver problemas técnicos.
- Proporciona a la dirección pruebas para confirmar el estado técnico del proyecto.
- El examen no necesita abordar todos los aspectos del producto.

Inspección

Es un examen visual de un producto de software para detectar e identificar anomalías de software, incluidos errores y desviaciones de las normas y especificaciones.

Objetivos principales:

- Detectar e identificar anomalías de productos de software, incluidos errores y desviaciones de las normas y especificaciones.
- Es un examen sistemático de pares
- Las inspecciones constan de dos a seis participantes
- Una inspección es dirigida por un facilitador imparcial capacitado en técnicas de inspección.
- Incluye una recopilación de métricas
- La recopilación de datos con el fin de analizar y mejorar los procedimientos de ingeniería de software es un elemento obligatorio de las inspecciones de software.
- Se entrega un informe de inspección en el que se incluye una lista de las conclusiones

Revisión guiada

Es una técnica de análisis estático en la que un diseñador o programador dirige a los miembros del equipo de desarrollo y a otras partes interesadas a través de un producto de software, luego los participantes hacen preguntas y comentarios sobre posibles anomalías

Objetivos principales:

- Encontrar anomalías.
- Mejorar el producto de software.
- Considerar implementaciones alternativas.
- Evaluar la conformidad con las normas y especificaciones.
- Evaluar la usabilidad y accesibilidad del producto de software.
- Intercambio de técnicas, variaciones de estilo y entrenamiento de los participantes

Auditoría

Es un examen independiente de un producto o proceso de software, realizado por un tercero para evaluar el cumplimiento de especificaciones, estándares, acuerdos contractuales u otros criterio.

Objetivo principal:

- Proporcionar una evaluación independiente de la conformidad de los productos y procesos de software con regulaciones, normas, directrices, planes, especificaciones y procedimientos aplicables.

TEMA 3.3

Análisis estático

Es aquella actividad que consiste en el análisis de un objeto de prueba (código fuente, script, requisito) llevado a cabo sin ejecutar el producto software. El análisis se realiza en alguna versión del código fuente y en otros casos se realiza en el código objeto.

Posibles aspectos a ser comprobados con análisis estático:

- ❖ Reglas y estándares de programación
- ❖ Diseño de un programa (análisis de flujo de control)
- ❖ Uso de datos (análisis de flujo de datos)
- ❖ Complejidad de la estructura de un programa
- ❖ Métricas

Aspectos generales del análisis estático con herramientas:

- ❖ Todos los objetos de prueba deben tener una estructura formal.
- ❖ El análisis estático de un programa mediante el uso de herramientas se desarrolla con un esfuerzo menor que el necesario a una inspección.
- ❖ El análisis estático se ejecuta de forma previa a que tenga lugar una revisión.
- ❖ Con el uso de herramientas para la realización de análisis estático el código del programa puede ser objeto de inspección sin ser ejecutado.
- ❖ En el transcurso de las pruebas estáticas no se ejecuta el objeto de prueba
- ❖ Herramienta a utilizar: Compiladores

Compiladores

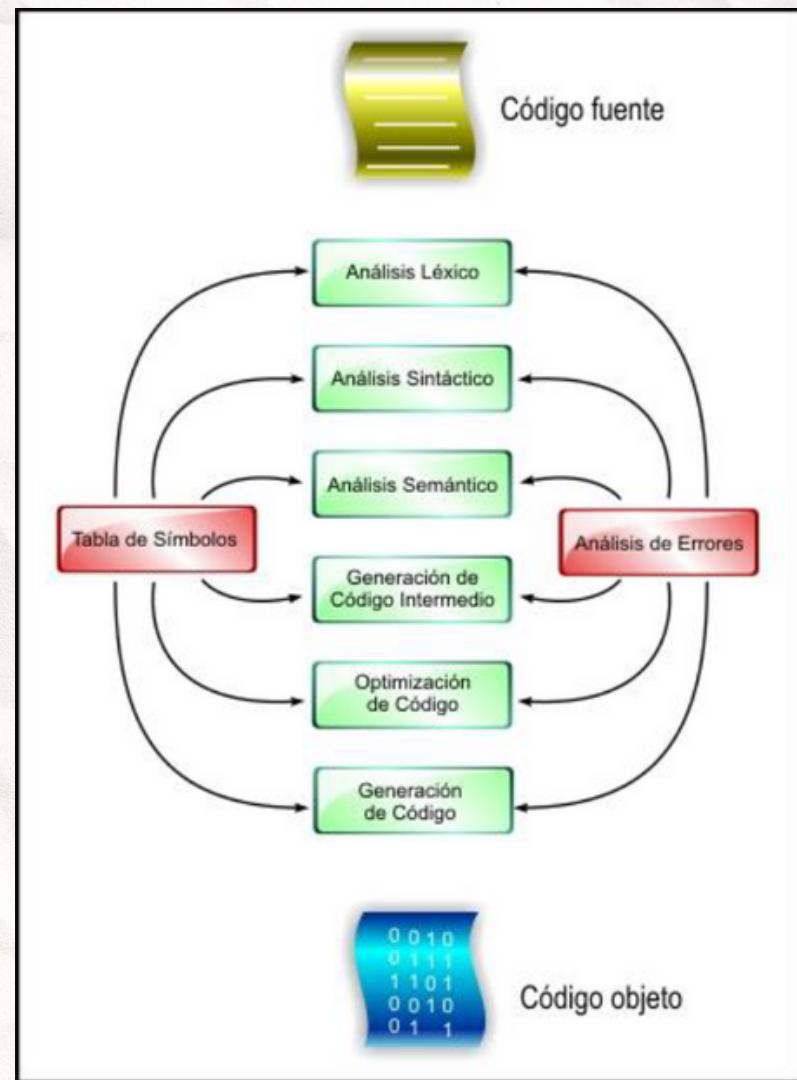
Son programas informáticos, que se encargan de traducir el lenguaje de programación de alto nivel a un lenguaje de máquina , el cual es conocido como de bajo nivel.

Se encargan de:

- ❖ Detectar errores sintácticos en el código fuente de un programa
- ❖ Crear datos de referencia del programa (lista de referencia cruzada, llamada jerárquica, tabla de símbolos)
- ❖ Comprobar la consistencia entre los tipos de variables
- ❖ Detectar variables no declaradas y código inaccessible

Compiladores

Los compiladores trabajan en fases, las cuales transforman el programa fuente de una representación en otra. Se usa con el fin de analizar las secciones, administrar la tabla de símbolos y manejar los errores, para esto utiliza:



Tipos de defectos detectados por las herramientas de análisis estático:

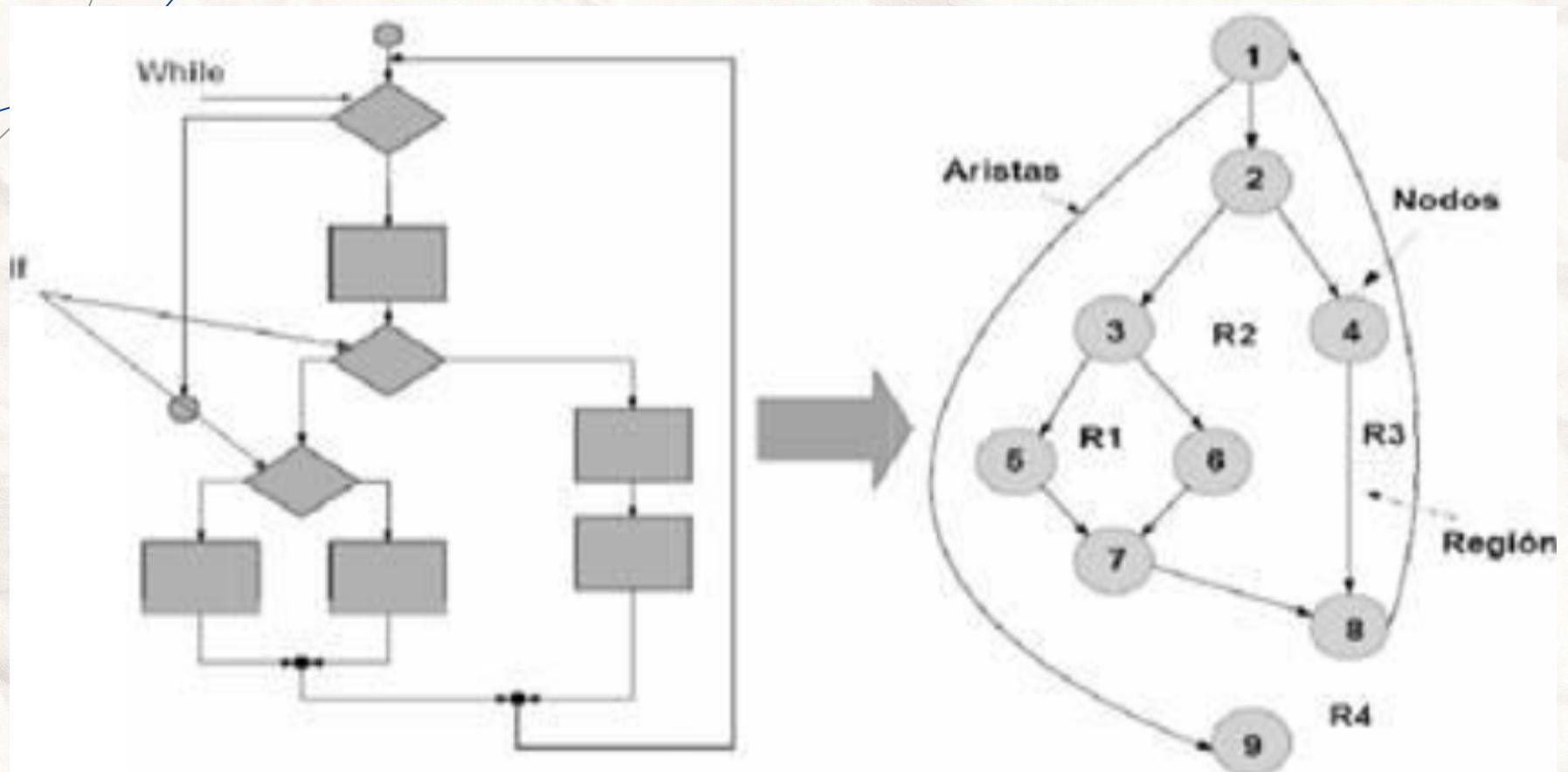
- Variables que no se utilizan
- Variables que se declaran de forma incorrecta
- Código inaccessible (muerto)
- Ausencia de lógica o lógica errónea (bucles indefinidos)
- Construcciones demasiado complicadas
- Infracciones de los estándares de programación
- Vulnerabilidad de seguridad
- Infracciones de sintaxis de código y modelos de software

Análisis del flujo de control

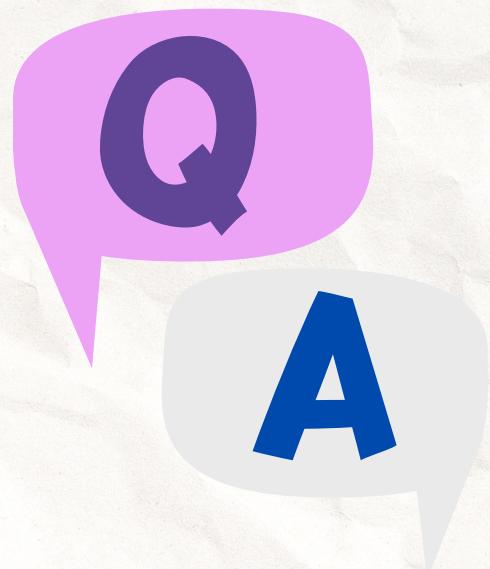
El diagrama de flujo de control presenta el flujo del programa y permite la detección de “Ramas muertas” y código inalcanzable

- Las métricas pueden ser utilizadas para evaluar la complejidad estructural conduciendo a una estimación del esfuerzo en pruebas a esperar.
- Con el diagrama de flujo de control las anomalías pueden ser fácilmente detectadas, los defectos se hacen evidentes.
- Las anomalías en los datos se detectan utilizando el análisis de flujo de datos o Grafo dirigido

Análisis del flujo de control



PREGUNTAS





GRACIAS

The word "GRACIAS" is written in a bold, sans-serif font. The letters are colored in a gradient: G (yellow), R (purple), A (pink), C (yellow), I (cyan), A (pink), S (purple). A thin horizontal line with a purple-to-pink gradient is positioned above the letter "R". A thick, solid pink horizontal bar is positioned below the letter "I".