



INGENIERIA DE SOFTWARE II

2025

Clase 1

Ingeniería de Software II - Temas



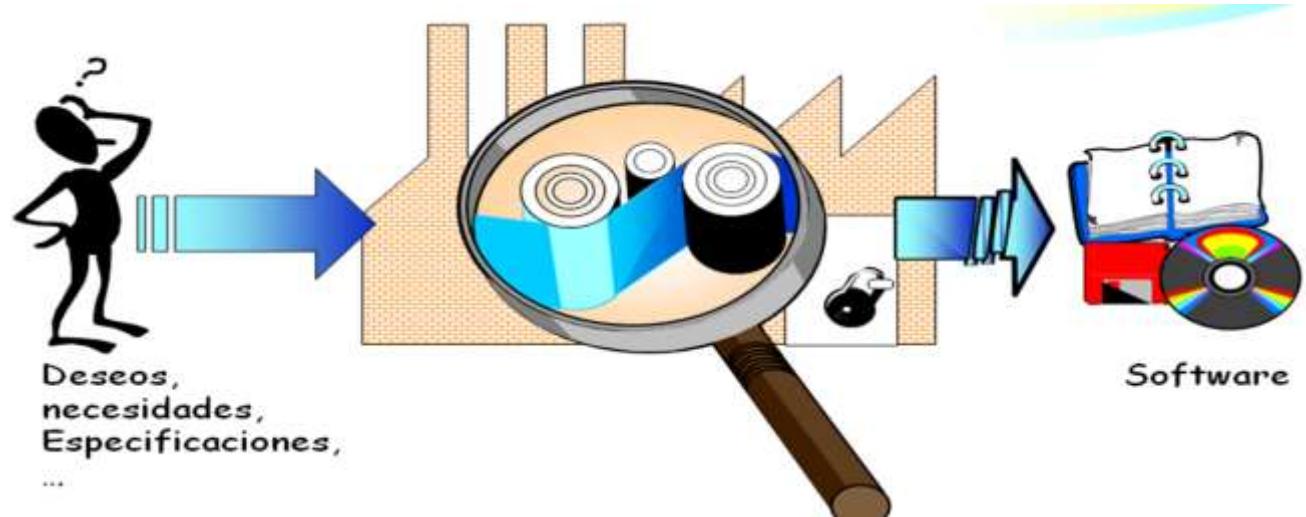
- 1- Gestión o Administración de Proyectos.
- 2- Diseño e Implementación de Software.
- 3- Verificación y Validación.
- 4- Mantenimiento de Software.
- 5- Gestión de Configuración.
- 6- Conceptos de Auditoría y Peritaje.



¿Qué es un proceso de software? - Repaso



- ❖ Es un conjunto de actividades y resultados asociados que producen un producto de software.



¿Modelo ? Es una representación abstracta de un proceso del software

Repaso

Actividades fundamentales de los modelos de Proceso

- Especificación del software
 - *Técnicas de elicitation*
 - *Especificación de requerimientos*
- Desarrollo del software
- Validación del software
- Evolución del software

El problema de la comunicación - Repaso

- ❖ Les proponemos ver el video y analizar los inconvenientes que se producen y qué sería necesario para poder resolverlo:

- ❖ Requerimientos

<https://www.youtube.com/watch?v=93SgXeu-SeY>



El problema de la comunicación y los requisitos



La solicitud del usuario



Lo que entendió el líder del proyecto



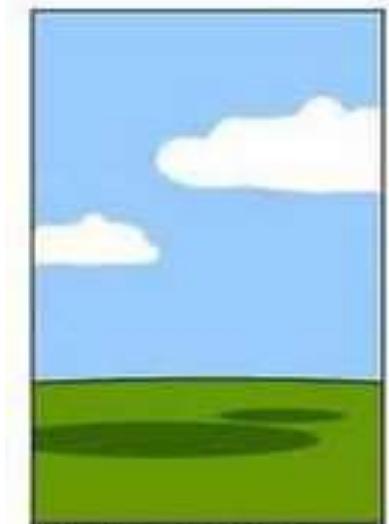
El diseño del analista de sistemas



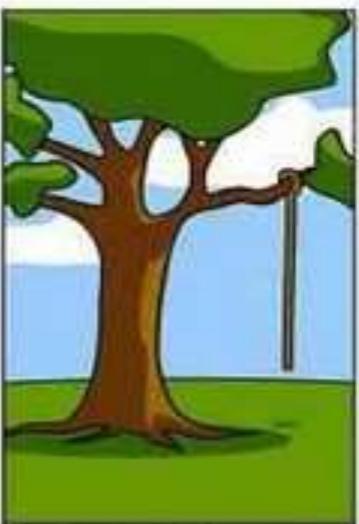
El enfoque del programador



La recomendación del consultor externo



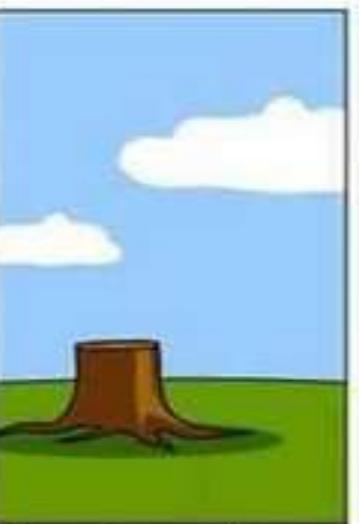
La documentación del proyecto de Software



La implantación en producción



El presupuesto del proyecto



El soporte operativo



Lo que el usuario realmente necesitaba



Comunicación

❖ ¿Cómo comunicar?

La **comunicación** es el proceso más importante de la interacción humana, una necesidad personal y la forma más completa de crear la realidad. Las personas comunicamos constantemente, tanto con las palabras (comunicación verbal) como con nuestra actitud y comportamiento (comunicación no verbal)

Peter Heinemann dice que la comunicación es un proceso de interacción social a través de símbolos y sistemas de mensajes que se producen como parte de la actividad humana.



Elicitación de Requisitos Repaso

Elicitación de Requisitos



- ❖ Es el proceso de adquirir (“eliciting”) [sonsacar] todo el conocimiento relevante necesario para producir un modelo de los requerimientos de un dominio de problema

- ❖ **Objetivos:**
 - Conocer el dominio del problema para poder comunicarse con clientes y usuarios y entender sus necesidades.
 - Conocer el sistema actual (manual o informatizado).
 - Identificar las necesidades, tanto explícitas como implícitas, de clientes y usuarios y sus expectativas sobre el sistema a desarrollar.

Muestreo de la documentación, los formularios y los datos existentes



Recolección de hechos a partir de la documentación existente.

¿Qué tipo de documentos pueden enseñar algo acerca del sistema?

- Organigrama (identificar el propietario, usuarios claves).
- Memos, notas internas, minutas, registros contables.
- Solicitudes de proyectos de sistemas de información anteriores.

Permiten conocer el historial que origina el proyecto

Observación del ambiente de trabajo

El analista se convierte en observador de las personas y actividades con el objeto de aprender acerca del sistema.

Lineamientos de la observación:

- Determinar quién y cuándo será observado.
- Obtener el permiso de la persona y explicar el porqué será observado.
- Mantener bajo perfil.
- Tomar nota de lo observado.
- Revisar las notas con la persona apropiada.
- No interrumpir a la persona en su trabajo.

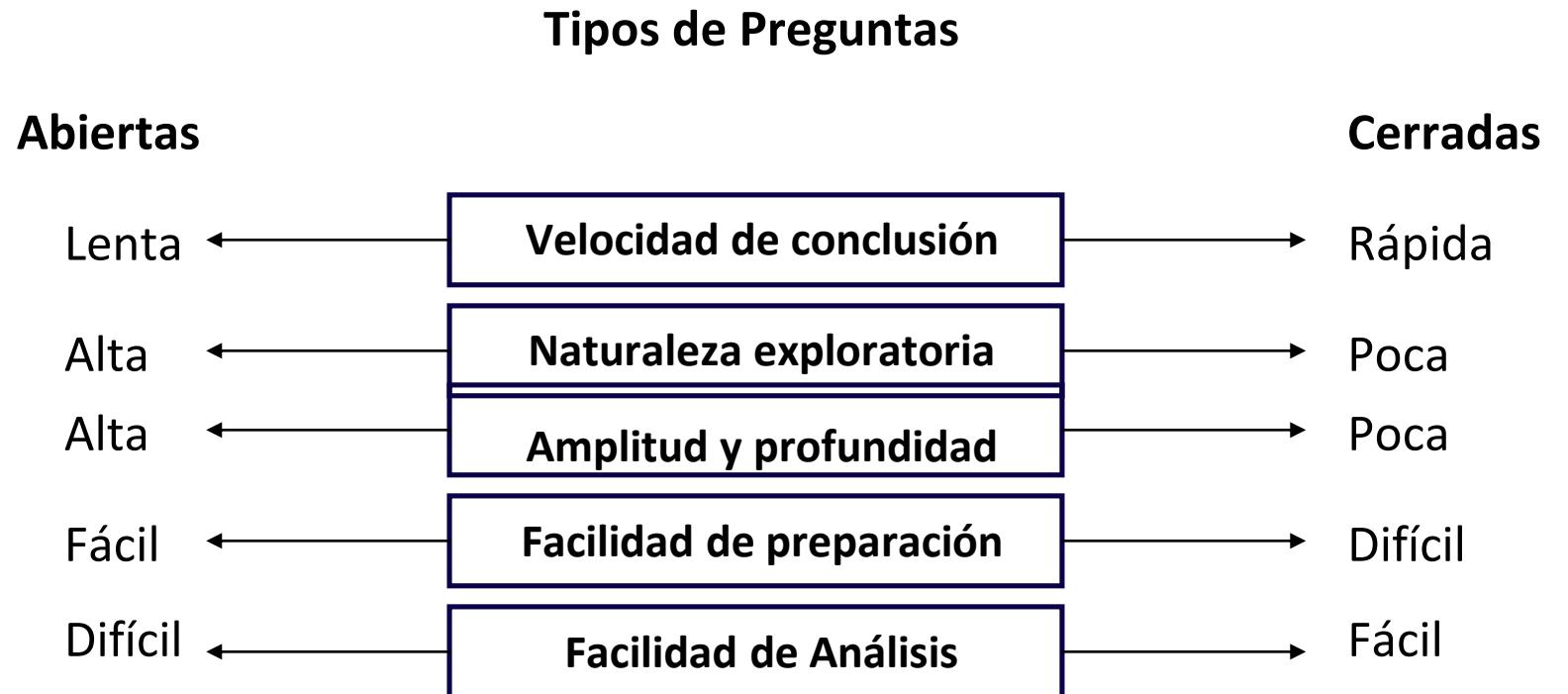


Visitas al sitio

- ✓ Investigar el dominio del problema.
- ✓ Patrones de soluciones (mismo problema en otra organización).
- ✓ Revistas especializadas.
- ✓ Buscar problemas similares en internet.
- ✓ Consultar otras organizaciones.



Cuestionarios



Entrevistas



Proceso de Preparación y Conducción de Entrevistas

Leer los Antecedentes

Revisar la información previa para obtener contexto

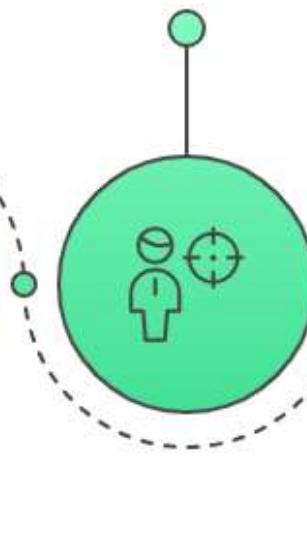


Prestar Atención al Lenguaje

Asegurar un vocabulario compartido con el entrevistado

Establecer Objetivos de Entrevista

Definir metas claras para la entrevista

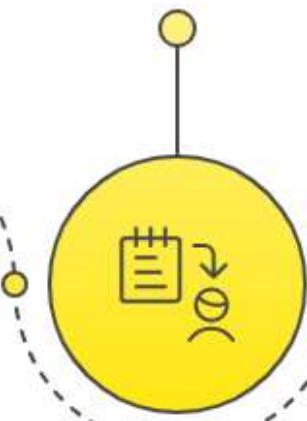


Seleccionar Entrevistados

Elegir participantes adecuados para la entrevista

Planificar Entrevista

Organizar los detalles logísticos de la entrevista

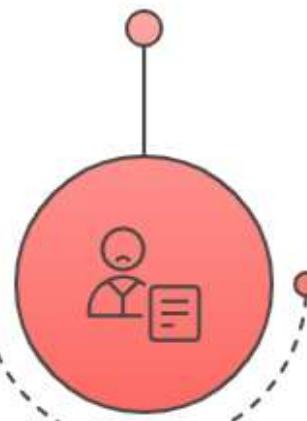


Preparar Preguntas

Diseñar preguntas efectivas y neutrales

Definir Guión de Entrevista

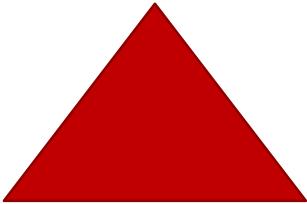
Crear un guión estructurado para la entrevista



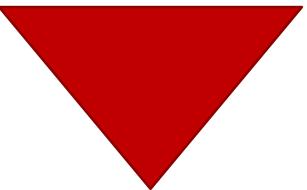
Organización de una entrevista



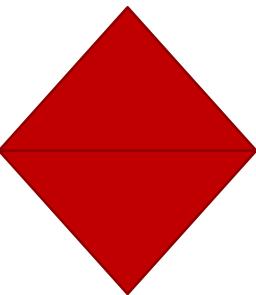
Piramidal (Inductivo)



Embudo (Deductivo)



Diamante (Comb. de las anteriores)



Entrevistas

Ejemplo de guión

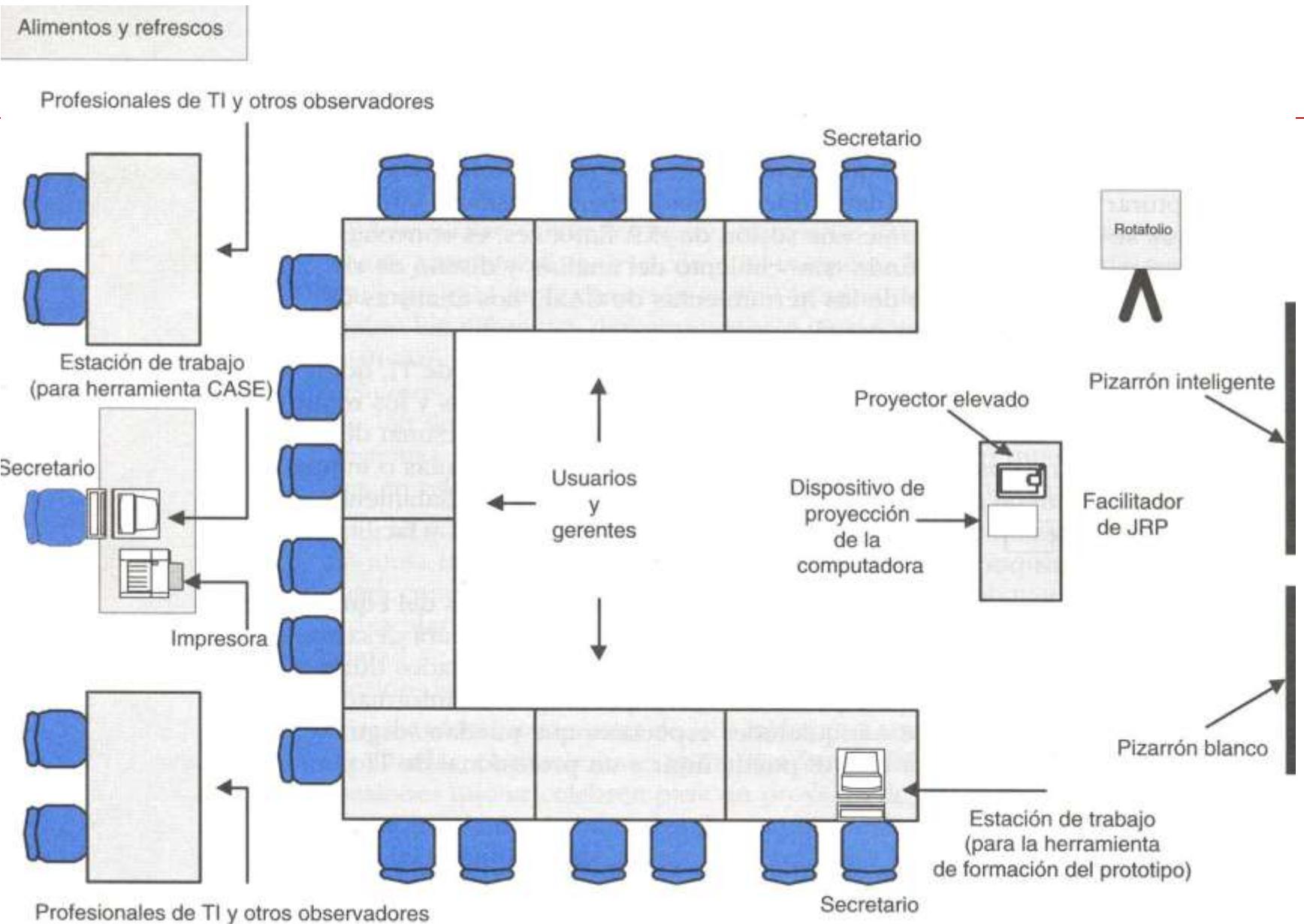
Entrevistado:	Jeff Bentley, Gerente de cuentas por cobrar	
Fecha:	19 de enero de 2003	
Hora:	1:30 p. m.	
Lugar:	Sala 223, Edificio de administración	
Tema:	Política actual de investigación de crédito	
Tiempo asignado	Pregunta u objetivo del administrador	Respuesta del entrevistado
1 a 2 min.	Objetivo Comienza la entrevista: <ul style="list-style-type: none">• Nos presentamos• Gracias Sr. Bentley por su valioso tiempo• Enunciar el propósito de la entrevista: obtener una comprensión de las políticas existentes de investigación de crédito.	
5 min.	Pregunta 1 ¿Qué condiciones determinan si se aprueba una solicitud de crédito del cliente? Seguimiento	
5 min.	Pregunta 2 ¿Cuáles son las posibles decisiones o acciones que podrían tomarse una vez que estas condiciones han sido evaluadas? Seguimiento	
3 min.	Pregunta 3 ¿Cómo se notifica a los clientes cuando no se aprueba su solicitud de crédito? Seguimiento	

Entrevistas

Ejemplo de guión

1 min.	<p>Pregunta 4</p> <p>Después que se aprueba una nueva solicitud de crédito y se coloca en el archivo que contiene las solicitudes que pueden llenarse, un cliente puede pedir que se haga una modificación a la solicitud. ¿Tendría que pasar ésta nuevamente por la aprobación de crédito si el costo total de la nueva solicitud sobrepasa al costo original?</p> <p>Seguimiento</p>
1 min.	<p>Pregunta 5</p> <p>¿Quiénes son las personas que realizan las investigaciones de crédito?</p> <p>Seguimiento</p>
1 a 3 min.	<p>Pregunta 6</p> <p>¿Puedo obtener el permiso para hablar con estas personas para aprender específicamente cómo llevan a cabo el proceso de investigación de crédito?</p> <p>Seguimiento</p> <p>Si así es: ¿Cuál sería el momento apropiado para reunirme con cada uno de ellos?</p>
1 min.	<p>Objetivo</p> <p>Termino de la entrevista:</p> <ul style="list-style-type: none"> • Agradezca al Sr. Bentley por su cooperación y asegúrele que estará recibiendo una copia de lo que se obtuvo durante la entrevista.
21 minutos	Tiempo asignado para preguntas y objetivos
9 minutos	Tiempo asignado para preguntas de seguimiento y redirección
30 minutos	Tiempo asignado para la entrevista (1:30 p.m. a 2:00 p.m.)

Comentarios generales y notas:



Brainstorming

» Técnica para generar ideas al alentar a los participantes para que ofrezcan tantas ideas como sea posible en un corto tiempo sin ningún análisis hasta que se hayan agotado las ideas.

» Se promueve el desarrollo de ideas creativas para obtener soluciones.





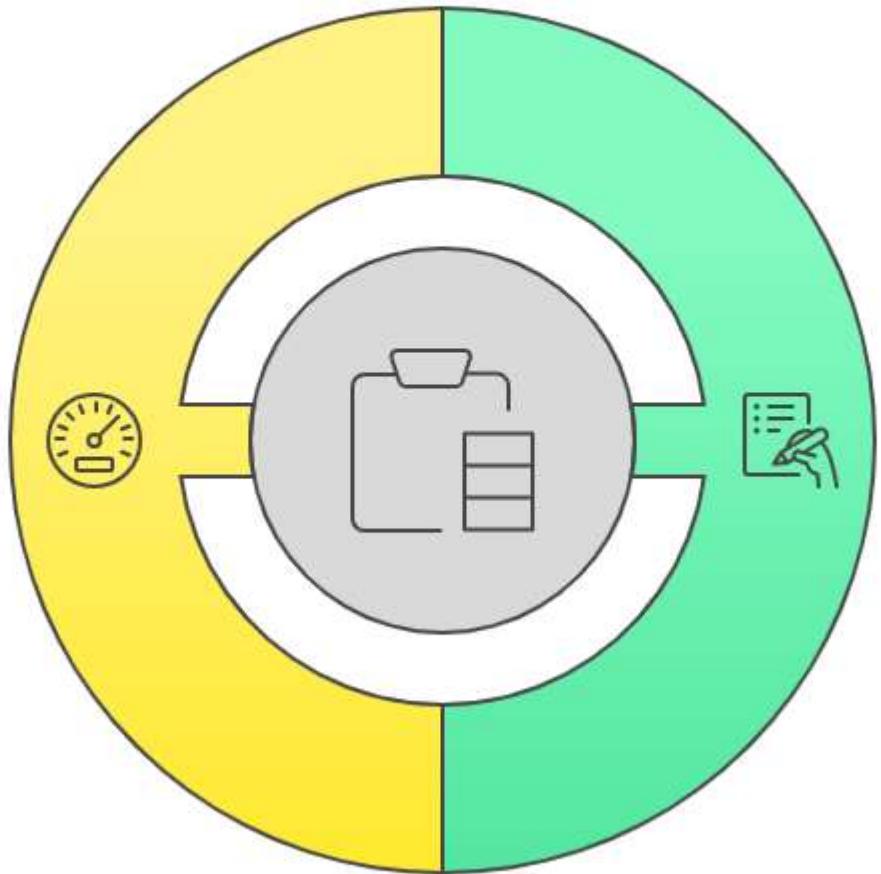
Requerimientos

Repaso

Tipos de requerimientos

Tipos de requerimientos del Sistema

Requerimientos No Funcionales
Describen atributos deseables como el rendimiento y la usabilidad.



Requerimientos Funcionales
Definen el comportamiento y las tareas del sistema.

Especificación del sistema

- ❖ Documento, también denominado *ConOps* y normalizado en el estándar *IEEE Std. 1362-1998*.
- ❖ Documento dirigido a los usuarios, que describe las características de un sistema propuesto, desde el punto de vista del usuario.
- ❖ La Descripción del Sistema es el medio de comunicación que recoge la visión general, cualitativa y cuantitativa de las características del sistema; compartido por la parte cliente y desarrolladora.

ConOps
Std 1362
IEEE

Descripción del sistema - IEEE 1362

- ❖ Ofrece un formato y contenidos para la confección de las descripciones de sistema en los desarrollos y modificaciones de sistemas.
- ❖ El estándar no especifica técnicas exactas, sino que proporciona las líneas generales que deben respetarse. Es una guía de referencia.
- ❖ El estándar identifica los elementos que al menos debe incluir una Descripción del sistema. El usuario puede incorporar otros elementos, agregando cláusulas y sub-cláusulas.

Descripción del Sistema

❖ Requerimientos del Software

Documento, también denominado SRS (ERS) y normalizado en el estándar IEEE Std. 830-1998.

Un documento **SRS** es la especificación de las funciones que realiza un determinado producto de software, programa o conjunto de programas en un determinado entorno.

El documento de especificación de requisitos puede desarrollarlo personal representativo de la parte desarrolladora, o de la parte cliente; si bien es aconsejable la intervención de ambas partes

830
SRS IEEE Std



IEEE Std 830-1998
(Revision of
IEEE Std 830-1993)

IEEE Recommended Practice for Software Requirements Specifications

❖ Alcance

Brindar una colección de buenas prácticas para escribir especificaciones de requerimientos de software (SRS). Se describen los contenidos y las cualidades de una buena especificación de requerimientos.

Características del SRS

❖ Naturaleza del SRS

El SRS es una especificación para un producto de software particular. El SRS es escrito por uno o más representantes del equipo de desarrollo y uno o mas representantes de la parte cliente o ambos.

❖ Ambiente del SRS

El software puede contener toda la funcionalidad del proyecto o puede ser parte de un sistema más grande. En el último caso habrá un SRS que declarará las interfaces entre el sistema y su software desarrollado, y pondrá qué función externa y requerimientos de funcionalidad tiene con el software desarrollado.

Características de un buen SRS



❖ Correcto

Un SRS es correcto si, y sólo si, cada requisito declarado se encuentra en el software.

❖ No ambiguo

Un SRS es inequívoco si, y sólo si, cada requisito declarado tiene sólo una interpretación.

Características de un buen SRS

❖ **Completo**

Un SRS está completo si, y sólo si, se reconoce cualquier requisito externo impuesto por una especificación del sistema.

❖ **Consistente**

La consistencia se refiere a la consistencia interior. Si un SRS no está de acuerdo con algún documento del nivel superior, como una especificación de requerimientos de sistema, entonces no es consistente.

Características de un buen SRS



❖ Priorizado

Un SRS es priorizado por la importancia de sus requerimientos particulares.

❖ Comprobable

Un SRS es comprobable si, y sólo si, cada requisito declarado es comprobable. Un requisito es comprobable si, y sólo si, existe algún proceso con que una persona o máquina puede verificar que el producto del software reúne el requisito. En general cualquier requisito ambiguo no es comprobable.

Características de un buen SRS



❖ **Modificable**

Un SRS es modificable si, y sólo si, su estructura y estilo son tales que puede hacerse cualquier cambio a los requerimientos fácilmente, completamente y de forma consistente mientras conserva la estructura y estilo

❖ **Trazabilidad**

Claridad del origen de cada requerimiento y su trazabilidad hacia los requerimientos para futuros desarrollos. Hacia adelante y hacia atrás

Consideraciones para un buen SRS

❖ Preparación conjunta del SRS

El SRS se debe preparar en conjunto con las partes intervinientes para lograr un buen acuerdo entre las partes.

❖ Evolución de SRS

El SRS debe evolucionar conjuntamente con el software, registrando los cambios, los responsables y aceptación de los mismos.

❖ Prototipos

El uso de prototipos se utiliza frecuentemente para la definición de requerimientos .

Consideraciones para un buen SRS

❖ Diseño incorporado en el SRS

El SRS puede incorporar los atributos o funciones externos al sistema, en particular las que describen el diseño para interactuar entre los subsistemas.

❖ Requerimientos incorporados en el SRS

Los detalles particulares de los requerimientos son anexados como documentos externos (CU, Plan de proyecto, Plan de aseguramiento de la calidad, etc.).

Partes de un SRS



FICHA DEL DOCUMENTO

CONTENIDO

1 INTRODUCCIÓN

1.1 Propósito

1.2 Alcance

1.3 Referencias

2 DESCRIPCIÓN GENERAL

2.1 Perspectiva del producto

2.2 Funcionalidad del producto

2.3 Características de los usuarios

2.4 Evolución previsible del sistema

3 REQUISITOS NO FUNCIONALES

3.1 Requisitos de rendimiento

3.2 Seguridad

3.3 Portabilidad

4 MANTENIMIENTO

5 APÉNDICES

Sección 1 del SRS

Introducción



❖ 1.1 Propósito

Se define el propósito del documento y se especifica a quién va dirigido el documento

❖ 1.2 Alcance o ámbito del sistema

Se da un nombre al futuro sistema . Se explica lo que el sistema hará y lo que no hará.

Se describen los *beneficios, objetivos* y *metas* que se espera alcanzar con el futuro sistema

❖ 1.3 Referencias

Se presenta una lista completa de todas las referencias de los documentos mencionados o utilizados para escribir el SRS.

Identificar cada documento por el título, número de reporte, fecha y publicación. Y las fuentes de las referencias de donde se obtuvieron.

Sección 2 del SRS

Descripción General



- ❖ Esta sección del SRS debe describir los factores generales que afectan el producto y sus requerimientos. No declara los requerimientos específicos. Los que se definen en detalle en Sección 3 del SRS.

- ❖ Esta sección normalmente consiste en:
 - Perspectiva del producto**
 - Funcionalidades del producto**
 - Características de los usuarios**
 - Evoluciones previsibles del sistema**

Sección 2 del SRS

Descripción General



❖ 2.1. Perspectiva del producto

Si el producto es independiente y totalmente autónomo, debe declararse que así es.

Si el SRS define un producto que es un componente de un sistema más grande entonces se debe relacionar los requerimientos de ese sistema más grande a la funcionalidad del software y debe identificar las interfaces entre ese sistema y el software.

Sección 2 del SRS

Descripción General



❖ **2.2. Funciones del sistema**

Se debe presentar un resumen de las funciones del futuro sistema.

Las funciones deberán mostrarse de forma organizada, y pueden utilizarse gráficos, siempre que reflejen las relaciones entre funciones y no el diseño del sistema.

Sección 2 del SRS

Descripción General



❖ **2.3. Características del Usuario**

Se deben describir las características generales de los usuarios intencionales del producto que incluye nivel educativo, experiencia, y la especialización técnica.

❖ **2.4. Evoluciones previsibles del sistema**

Se identifican requerimientos que serán implementados en futuras versiones

Requerimientos no funcionales



- ❖ Debe contener todos los requerimientos no funcionales del software a un nivel de detalle para permitirles a los diseñadores diseñar el sistema, y a los auditores probar que el sistema satisface esos requerimientos.



❖ 3.3.1 Requerimientos de rendimiento

Requerimientos relacionados con la carga que se espera tenga que soportar el sistema. Por ejemplo, nro de terminales, nro esperado de usuarios simultáneamente conectados, etc.

Todos estos requerimientos deben ser mensurables. Por ejemplo, indicando “el 95% de las transacciones deben realizarse en menos de 1 segundo”, en lugar de “los operadores no deben esperar a que se complete la transacción”.

Sección 3 del SRS

Requerimientos no funcionales

❖ 3.3.2 Seguridad

Especificación de elementos que protegerán al software de accesos, usos y sabotajes maliciosos, así como de modificaciones o destrucciones maliciosas o accidentales. Los requerimientos pueden especificar:

Empleo de técnicas criptográficas, Registro de ficheros con “logs” de actividad, Asignación de determinadas funcionalidades a determinados módulos, Restricciones de comunicación entre determinados módulos, Comprobaciones de integridad de información crítica.





❖ 3.3.3 Portabilidad

Especificación de atributos que debe presentar el software para facilitar su traslado a otras plataformas u entornos.

Pueden incluirse:

% componentes dependientes del servidor. % código dependiente del servidor. Uso de un determinado lenguaje por su portabilidad. Uso de un determinado compilador o plataforma de desarrollo. Uso de un determinado sistema operativo.

Sección 4 del SRS

Mantenimiento



- ❖ Identificación del tipo de mantenimiento necesario del sistema.
- ❖ Especificación de quien debe realizar las tareas de mantenimiento, por ejemplo usuarios, o un desarrollador.
- ❖ Especificación de cuando debe realizarse las tareas de mantenimiento. Por ejemplo, generación de estadísticas de acceso semanales y mensuales.

Sección 5 del SRS

Apéndices



Pueden contener todo tipo de información relevante para la SRS pero que, propiamente, no forme parte de la SRS.

Por ejemplo:

Casos de Uso

Historias de usuario



Comunicación del proyecto

Comunicación



- ❖ ¿Qué pasa cuando realizamos presentaciones acompañadas de un software tipo power point o prezi?

menos
más

Regla de presentación

Guy Kawasaki, experto en marketing y publicidad, recomienda atenerse a la regla del **“10/20/30”**: la presentación debe contar con no más de 10 diapositivas, no sobrepasar los 20 minutos de duración y no contener tipografías con cuerpo menor a 30. Busca en todo momento cómo simplificar la información y aliviar la diapositiva de elementos innecesarios.

Comunicación - Consejos



No leas la presentación al pie de la letra.



No mires la pantalla cuando expongas.



Usa la zona de notas de las diapositivas.



Colócate en el centro del escenario.

Comunicación - Consejos



Escribe menos,
una sola idea por
diapositiva.



No animes el
texto de las
diapositivas.



Tamaño de la
letra, usa un
tamaño que sea
legible a cualquier
distancia, se
recomienda usar
un tamaño de
fuente de 32.



Usa un tipo de
letra que se lea
bien, que no use
adornos o
remates, como es
el caso de la
Times New
Roman o Courier.

Comunicación - Consejos



Usa fondos claros
y sencillos.



Usa titulares, no
apartados.



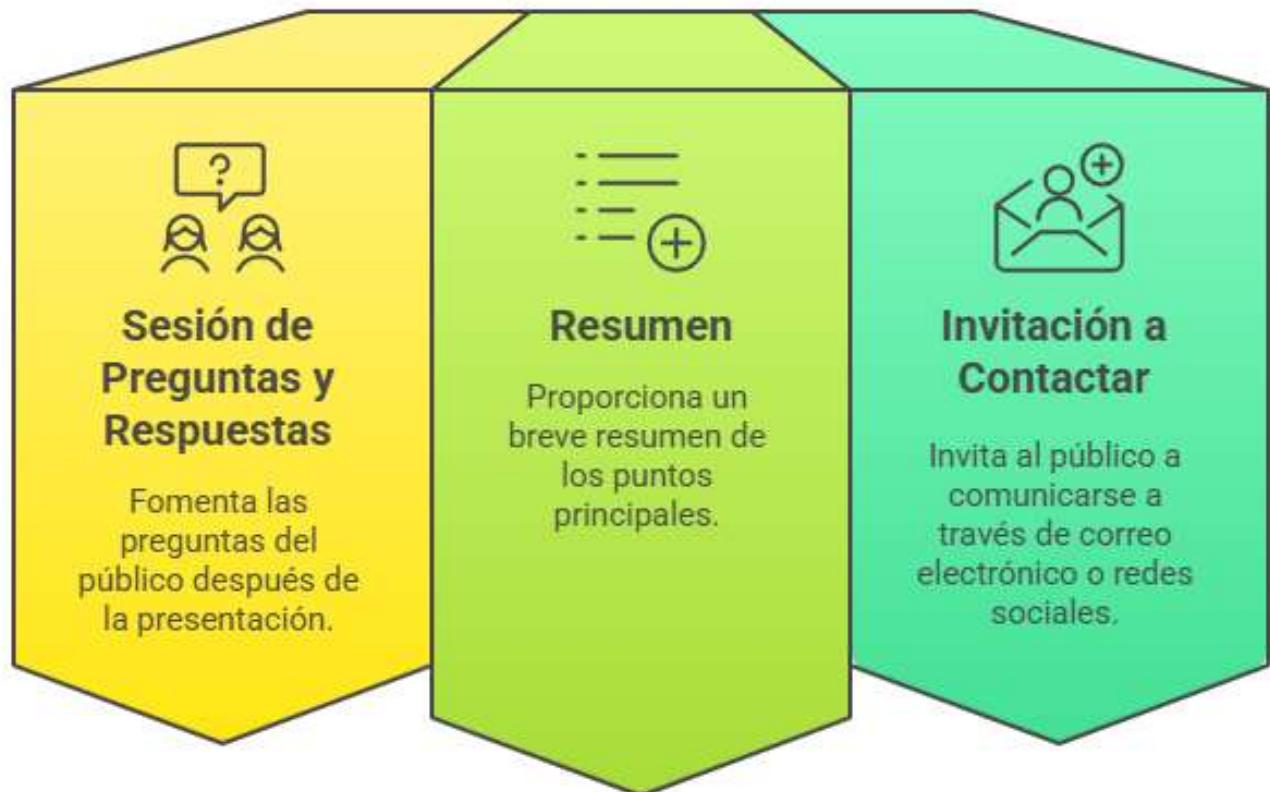
Cuidado con los
bullets (flechas,
puntos,
asteriscos,
símbolos que se
ponen delante de
una frase).



Diseña las
diapositivas para
ser entendidas en
3 segundos.

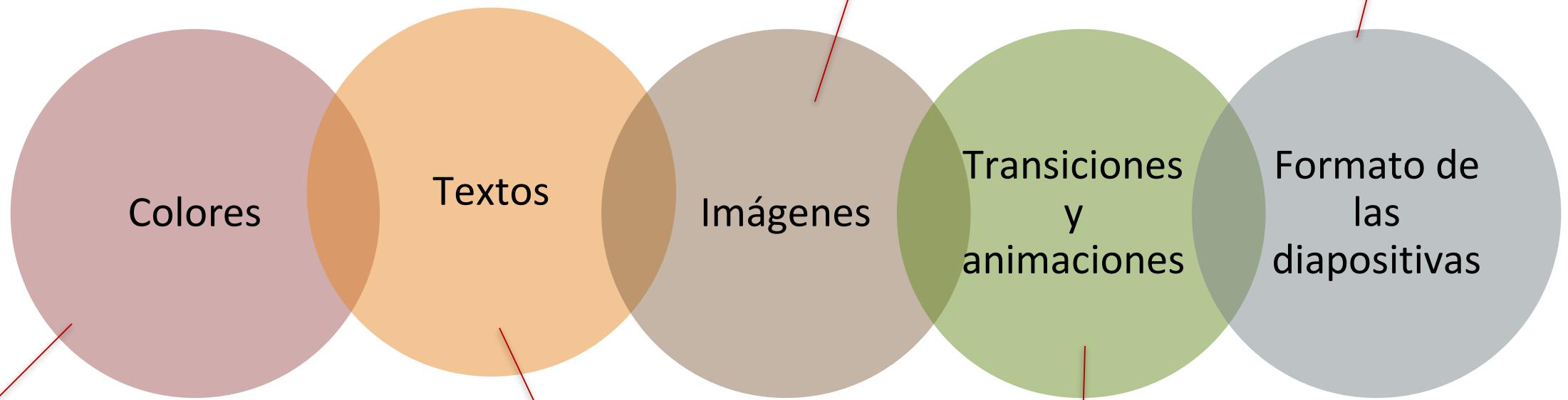
Comunicación - Consejos

Ideas para Cerrar la Presentación



Comunicación

❖ Presentaciones detalles estéticos



Colores muy oscuros para el texto y pasteles pálidos para el fondo, como amarillo claro, aunque algunos prefieren texto claro sobre fondo oscuro.

Mantenga el formato de fuente (tipo de letra, tamaño, títulos, etc.). Se debe usar un tipo de letra claro y fácil de leer como Arial, Tahoma o Verdana

Las ayudas visuales como gráficos, mapas, dibujos o fotografías, entre otras, se deben utilizar para permitir a la audiencia visualizar conceptos que de otra forma resultarían difíciles de entender

El formato de las diapositivas debe ser el de un esquema, en el que cada párrafo representará una idea.

Use transiciones naturales, como el texto que cae o aparece desde la izquierda. No se exceda con las transiciones y animaciones de texto, pues no todas las diapositivas requieren efectos especiales

Algunos tips más en general para el ppt que se pide

La presentación debe ser llamativa desde lo visual, debe ofrecer algo distintivo, disruptivo o novedoso.

Debe invitar a ser vista por quien recibirá el mensaje.

El contenido debe ser afirmativo en tiempo presente, por ejemplo del estilo "Esta app permite...."

Cosas que no deben faltar:

- » Nombre del proyecto
- » Nombre de la empresa de desarrollo
- » Objetivo
- » Características o funcionalidades principales y/o diferenciadoras
- » Técnicas de recolección de datos que se emplean
- » Contacto



Video tipo elevator pitch

Comunicación



❖ ¿Cómo comunicar un proyecto o idea?

Les proponemos ver esta situación y luego realizaremos un análisis para ver las características y formas de comunicar

Elevator Pitch – 20 segundos!!

https://www.youtube.com/watch?v=2b3xG_Yjgvl



- ❖ ¿Cómo comunicar un proyecto o idea?
 - ✓ ¿Qué es lo primero que debe hacerse para arrancar la presentación de la idea/proyecto?
 - ✓ ¿Cómo se ofrece el proyecto?
 - ✓ ¿Cómo sugiere el video que se consiga el financiamiento?
 - ✓ ¿Cómo se sugiere continuar la presentación del proyecto?

Comunicación



❖ ¿Cómo hacer un Elevator pitch?

<https://www.youtube.com/watch?v=uv357YzY7-k>



Un “elevator pitch” es un mensaje corto de aproximadamente un minuto de duración para contar tu proyecto de manera diferenciadora

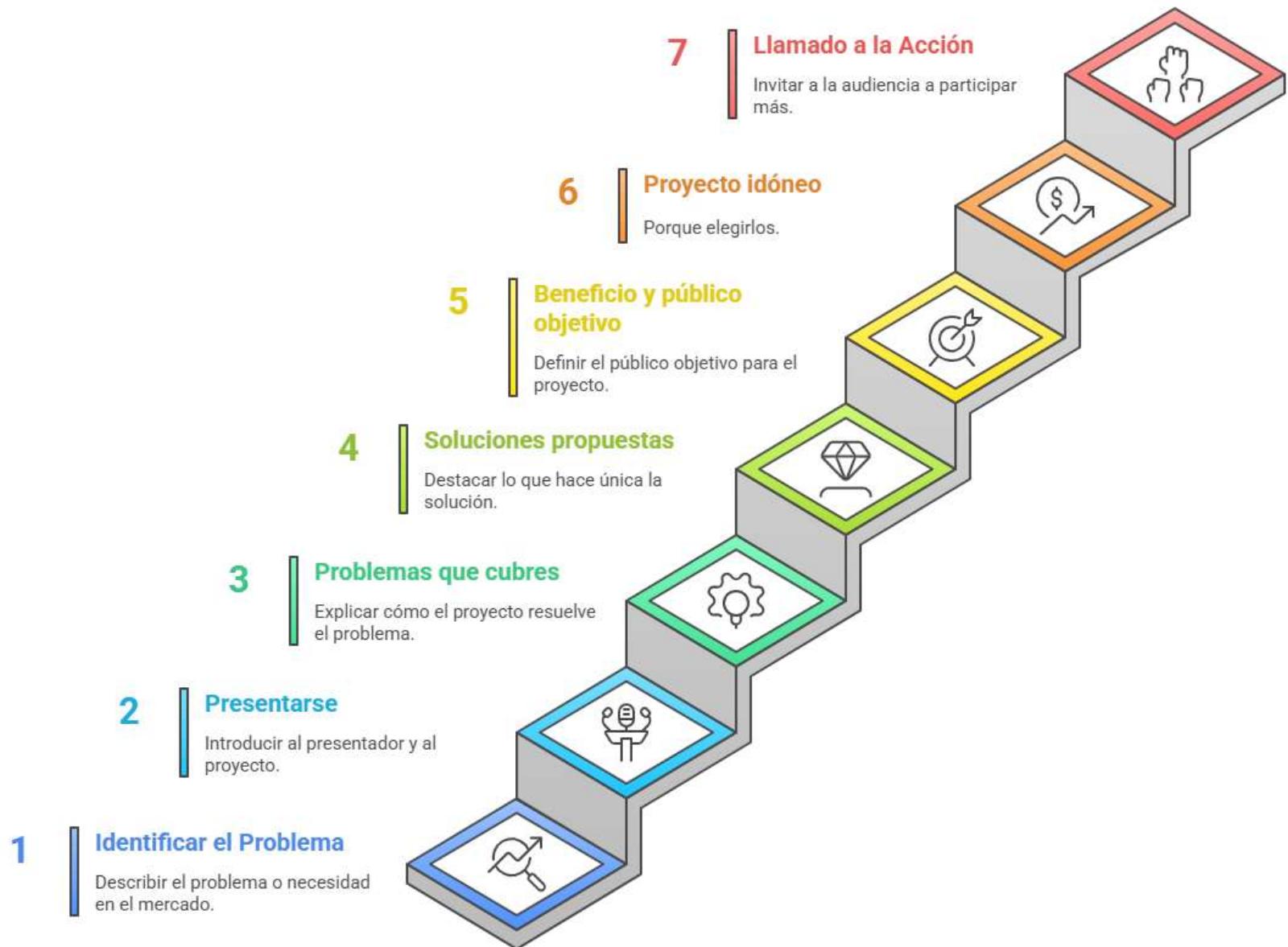
Comunicación

Pasos del video del Elevator pitch

- 1 • Afirmación o pregunta sorprendente para llamar la atención
- 2 • Presentación: contar quién eres
- 3 • Problemas o necesidades que cubres
- 4 • ¿Qué soluciones aportas?
- 5 • ¿Qué beneficio principal obtiene la gente contigo?
- 6 • ¿Por qué tu proyecto es el idóneo?
- 7 • Termina con una llamada a la acción final

Pasos

Pasos para un Pitch de Proyecto Exitoso



Tips para elaborar un guión de elevator pitch



Presentación oral en vivo, consejos



Algunos tips más para video grabado

Deben prestar atención a la comunicación NO verbal:

- La / las personas que aparecen en el video deben sonreir
- Utilizar los dedos para contar ideas o puntos importantes
- Mostrar y/o mover las manos en referencia a lo que se está contando

Referido a lo verbal:

- Deben poner diferentes énfasis en la voz para lograr comunicar efectivamente y resaltar cosas.
- Analizar cuándo modificar la voz y cuando utilizar gestos no verbales



INGENIERIA DE SOFTWARE II

2025

Clase 2



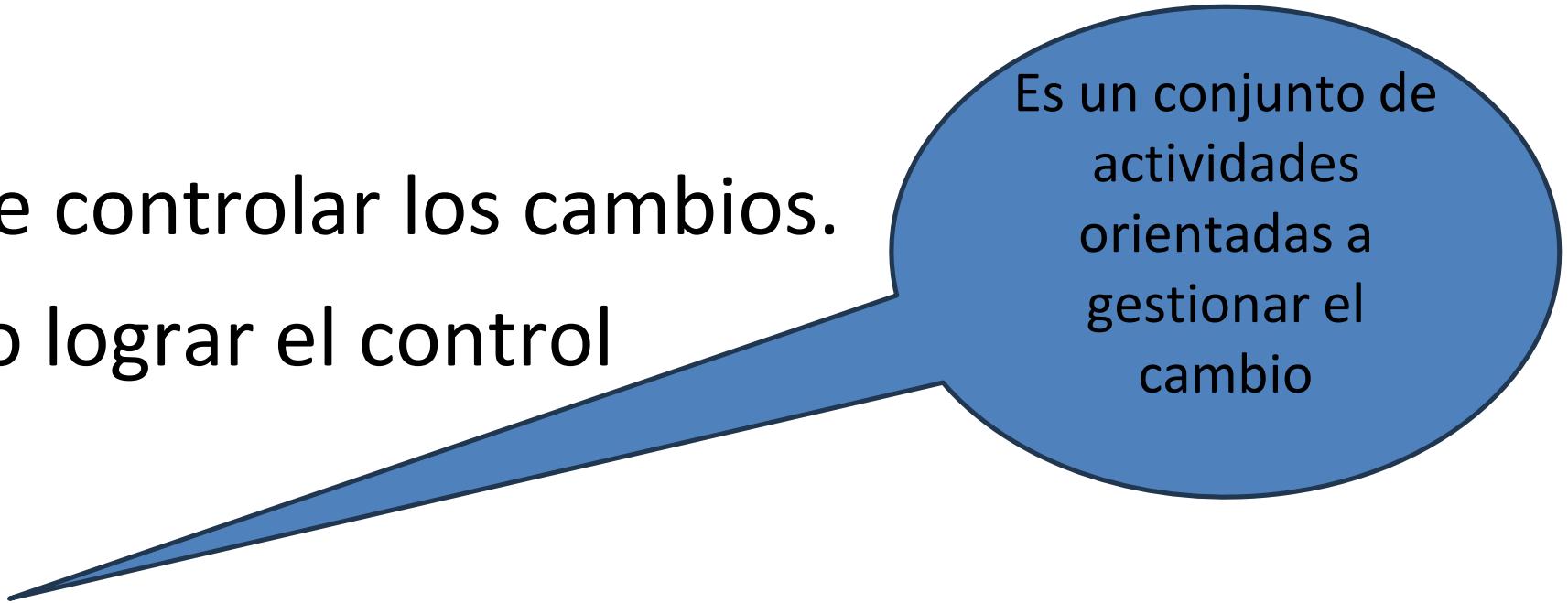
Contenidos

- ❖ Gestión de la Configuración del Software (GCS)
- ❖ Gestión de Proyectos
 - Planificación Organizativa

Gestión de la Configuración del Software (GCS)

Los cambios suceden ??

- Importancia de controlar los cambios.
- Perjuicio de no lograr el control



Es un conjunto de actividades orientadas a gestionar el cambio

GCS: ¿Qué es ?

Gestión de la Configuración del Software (GCS)

Gestión de Configuración es el proceso de:

- Identificar y definir los elementos en el sistema, controlando el cambio de estos elementos a lo largo de su ciclo de vida, registrando y reportando el estado de los elementos y las solicitudes de cambio, y verificando que los elementos estén completos y que sean los correctos.

Es una actividad de autoprotección que se aplica durante el proceso del software.

¿Quién hace la GCS?

Todos los involucrados en el proceso de software se relacionan en cierta medida con la gestión del cambio, pero en ocasiones se crean posiciones de apoyo especializadas para administrar el proceso de aseguramiento de la calidad del software.

Elementos de la GCS (ECS)

Se dividen en 3 categorías:



Programas

Aplicaciones de software utilizadas para tareas.



Productos de Trabajo

Resultados creados durante el proceso de software.



Datos

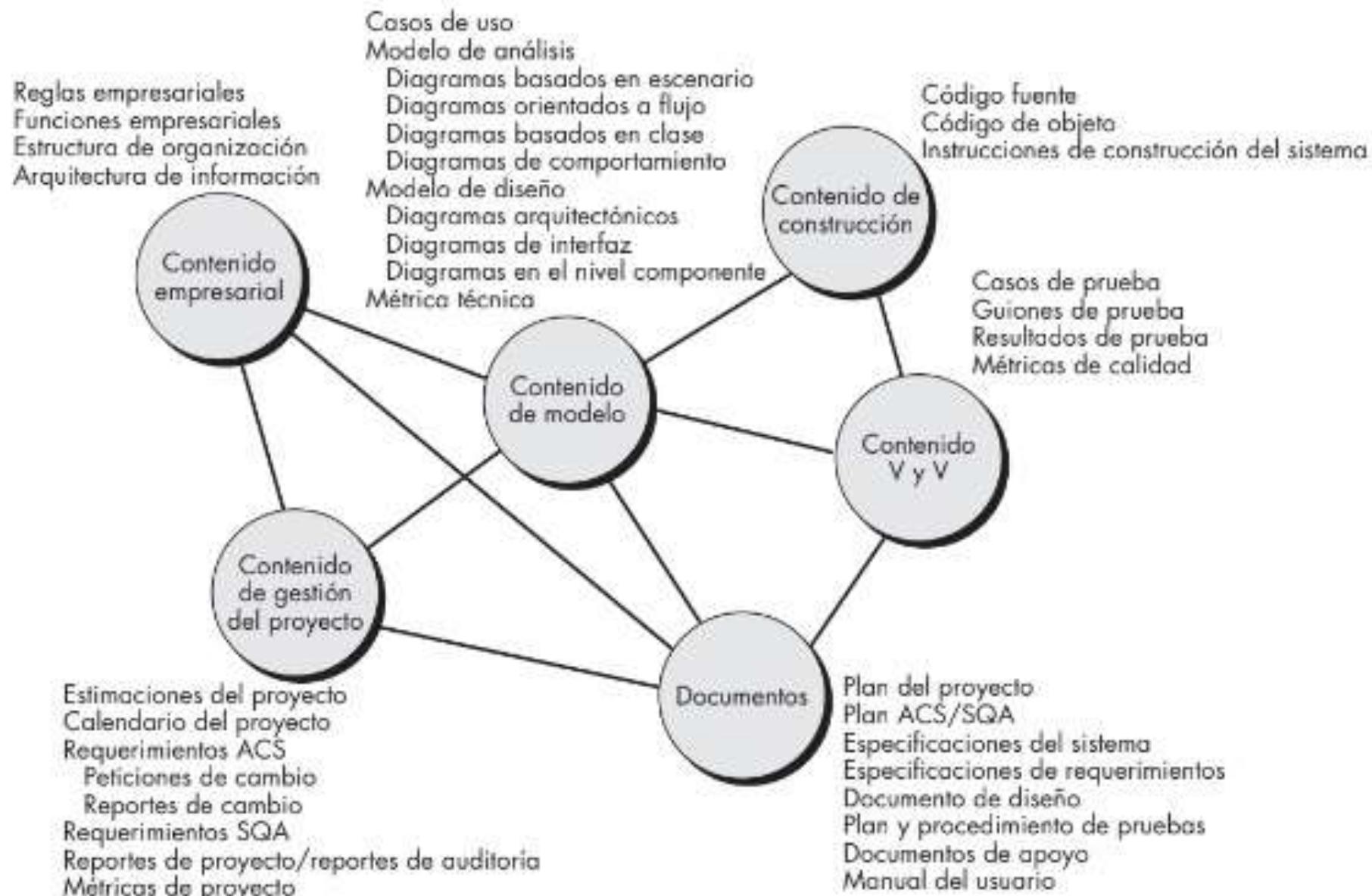
Información utilizada y generada por el software.

Elementos de la GCS (ECS)

¿Cuáles podrían ser elementos de la configuración?

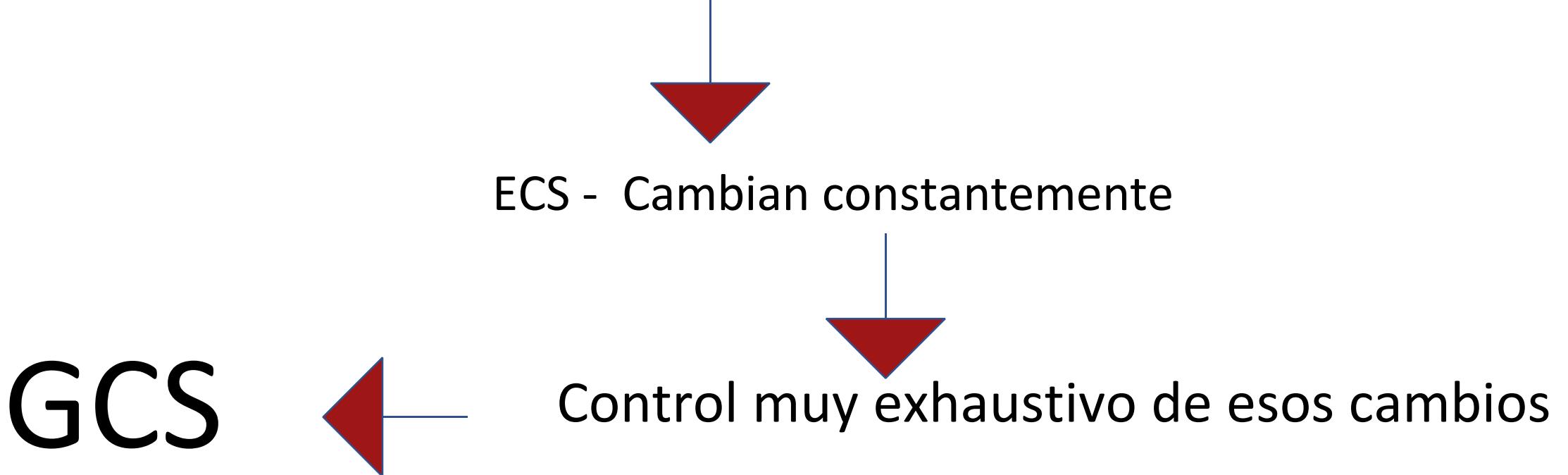
<ul style="list-style-type: none">✓ Especificación del sistema✓ Plan del proyecto software✓ Especificación de diseño:<ul style="list-style-type: none">✓ a) Diseño preliminar✓ b) Diseño detallado✓ Listados del código fuente✓ Planificación y procedimiento de prueba✓ Casos de prueba y resultados registrados	<ul style="list-style-type: none">✓ Manuales de operación y de instalación✓ Programas ejecutables✓ Descripción de la base de datos<ul style="list-style-type: none">a) Esquema, modelosb) Datos iniciales✓ Manual de usuario✓ Documentos de mantenimiento✓ Estándares y procedimientos de ingeniería del software
---	---

Organización de un repositorio de ECS



Gestión de la Configuración del Software (GCS)

Elementos de la configuración (ECS)

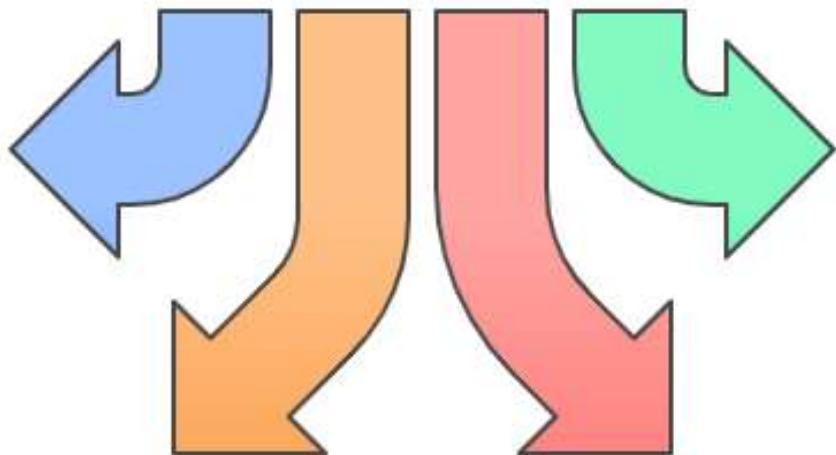


Gestión de la Configuración del Software (GCS)

¿Qué causa el cambio en la gestión de configuración del software?

Condiciones de Negocio

Cambios en el mercado o requisitos comerciales



Necesidades de las Partes Interesadas

Nuevas demandas de las partes interesadas

Reorganización

Cambios en la estructura o tamaño de la empresa, estructura de equipo o prioridades del proyecto

Restricciones Presupuestarias

Presiones presupuestarias o de tiempo

GCS – Línea base

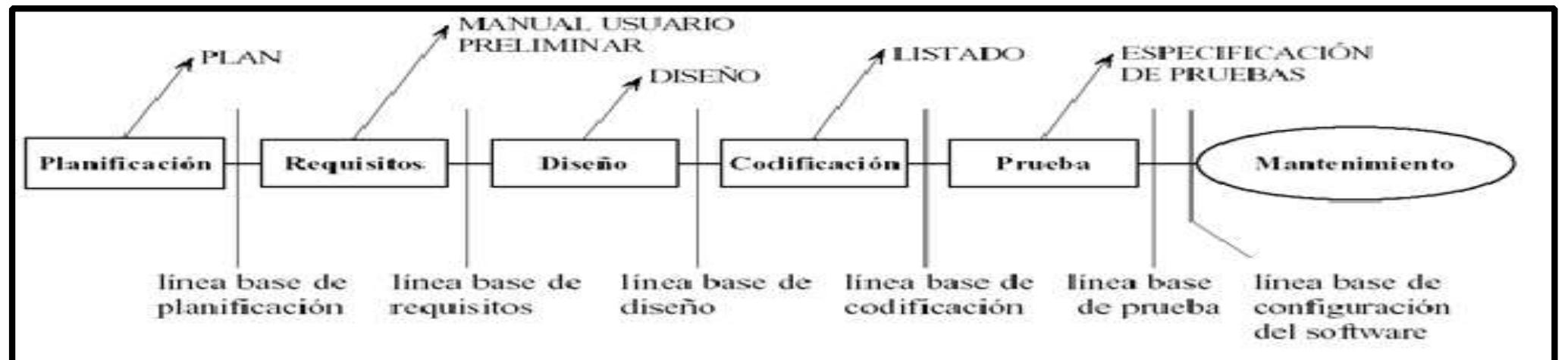
Una línea base es un concepto de GCS que nos ayuda a controlar los cambios

Definición de la IEEE

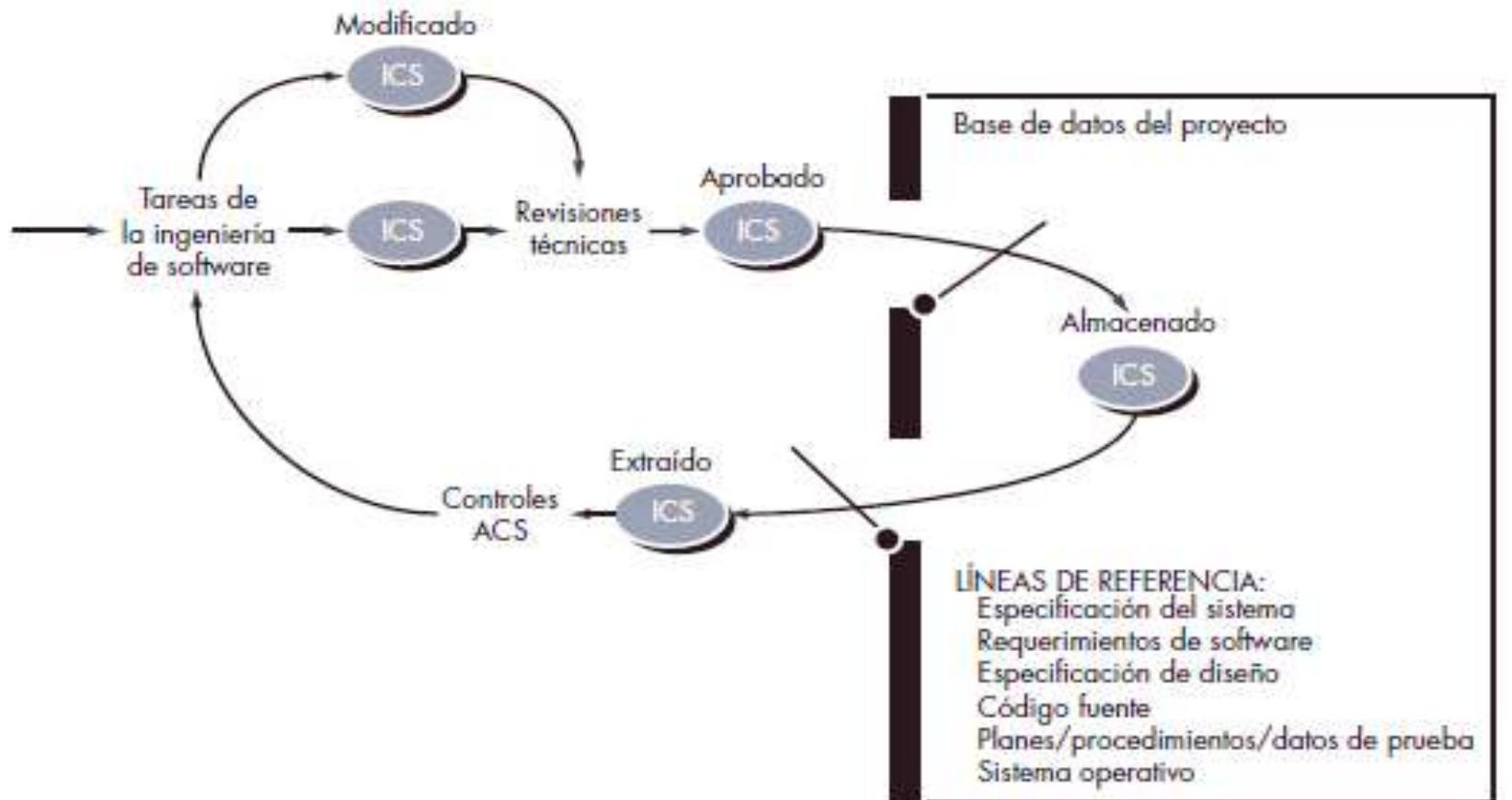
Una especificación o producto que se ha revisado formalmente y sobre el que se ha llegado a un acuerdo, y que de ahí en adelante sirve como base para un desarrollo posterior y que puede cambiarse solamente a través de procedimientos formales de control de cambio

En el contexto de la Ingeniería de Software:

Una línea base es un punto de referencia en el desarrollo del software que queda marcado por el envío de uno o más ECS y su aprobación



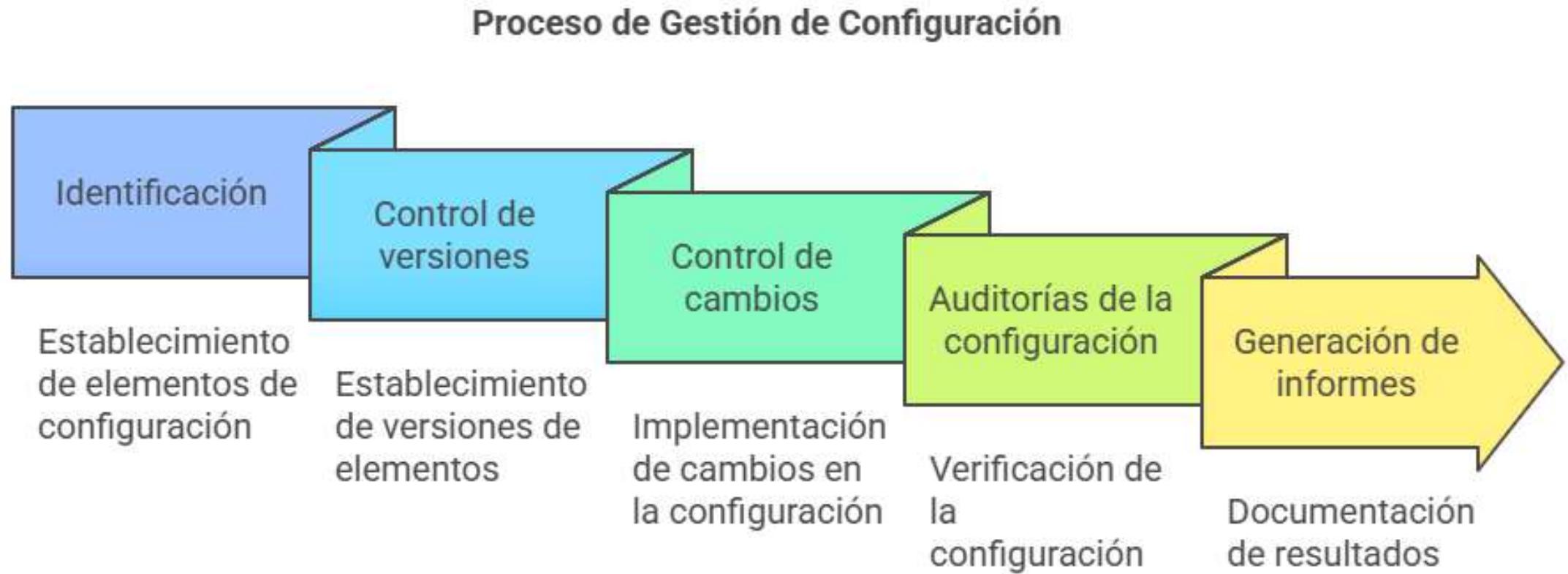
GCS – Línea base



GCS - Importancia

- ❖ ¿Cómo identifica y gestiona una organización las diferentes versiones existentes de un programa (y su documentación) de forma que se puedan introducir cambios eficientemente?
- ❖ ¿Cómo controla la organización los cambios antes y después de que el software sea distribuido al cliente?
- ❖ ¿Quién tiene la responsabilidad de aprobar y de asignar prioridades a los cambios?
- ❖ ¿Cómo podemos garantizar que los cambios se han llevado a cabo adecuadamente?
- ❖ ¿Qué mecanismo se usa para avisar a otros de los cambios realizados?

GCS - Proceso



GCS - Proceso

1- Identificación de los elementos en la GCS

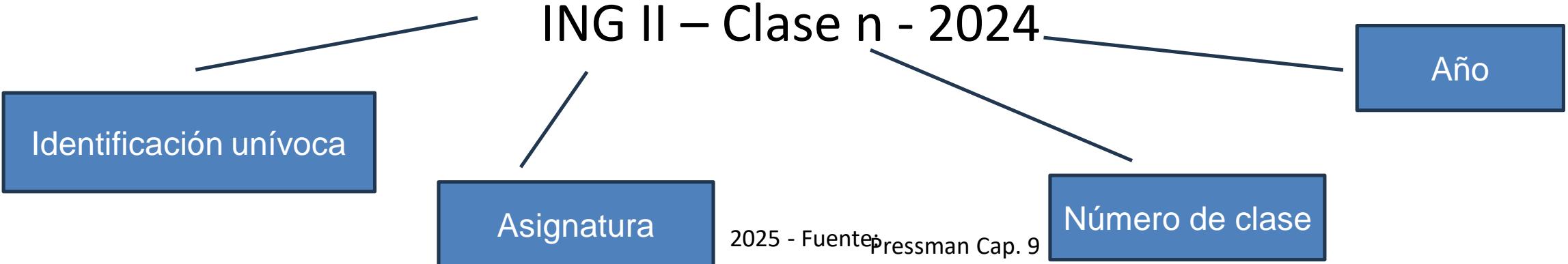
Nombre: cadena de caracteres sin ambigüedad

Descripción: lista de elementos de datos que identifican:

Tipo de ECS (documento, código fuente, datos)

Identificador del proyecto

Información de la versión y/o cambio



GCS - Proceso

2 - Control de versiones

Combinación de procedimientos y herramientas para gestionar las versiones de los ECS que se crean a lo largo del proceso de software.

Ejemplo de versiones

Un programa puede contener los módulos 1-2-3-4-5

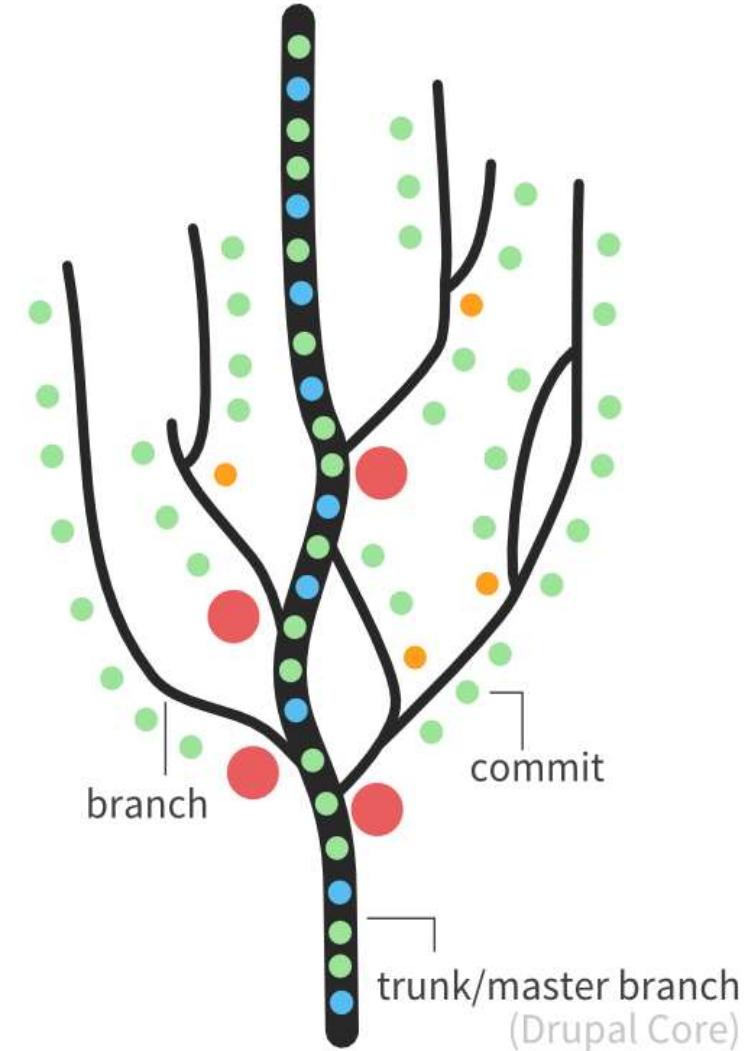
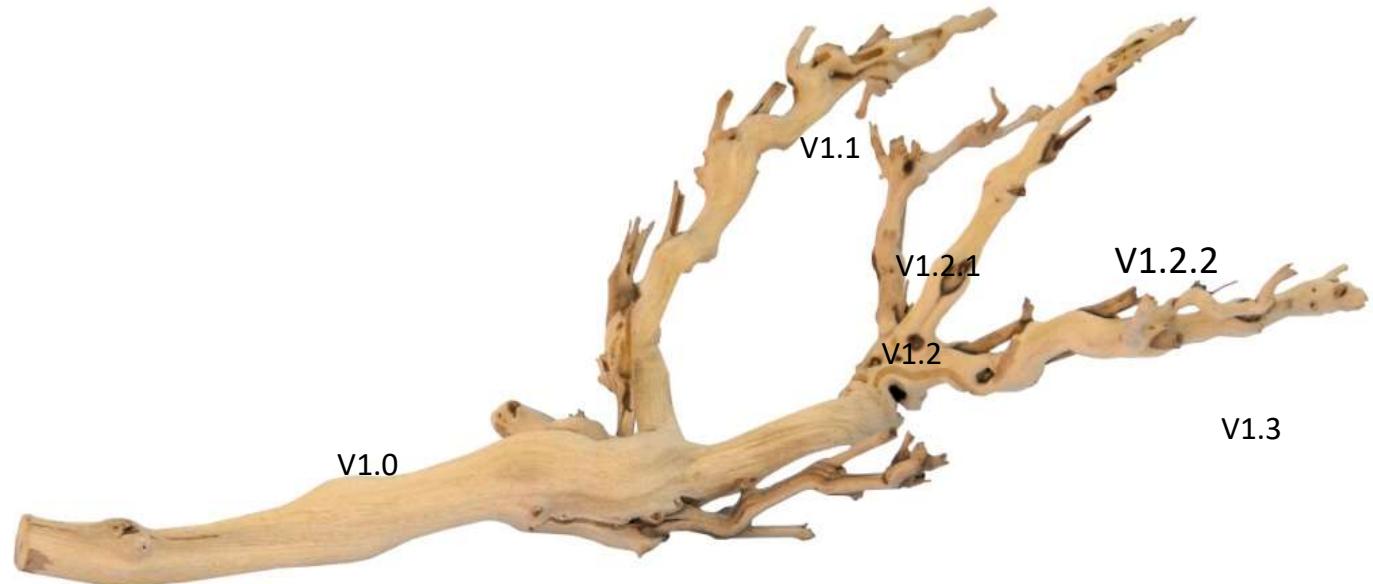
Una versión puede utilizar los módulos 1-2-3-5

Otra versión puede utilizar los módulos 1-2-4-5

Dos variantes de un mismo programa

GCS - Proceso

2 - Control de versiones



GCS - Proceso

2 - Control de versiones

Repositorio	Se almacenan los archivos actualizados e históricos de cambio del proyecto.
Versión	Determina un conjunto de archivos
Master	Conjunto de archivos principales del proyecto
Abrir rama – branch	Bifurcación del máster para trabajar sobre dos ramas de forma independiente
Desplegar – check-out	Copia de trabajo local desde el repositorio.
Publicar - Commit	Una copia de los cambios hechos a una copia local es escrita o integrada sobre repositorio
Conflicto	Problema entre las versiones de un mismo documento
Cambio – diff	Representa una modificación específica
Integración – Merge	Fusión entre dos ramas del proyecto
Actualización – sync o update	Integra los cambios que han sido hechos en el repositorio y las copias locales

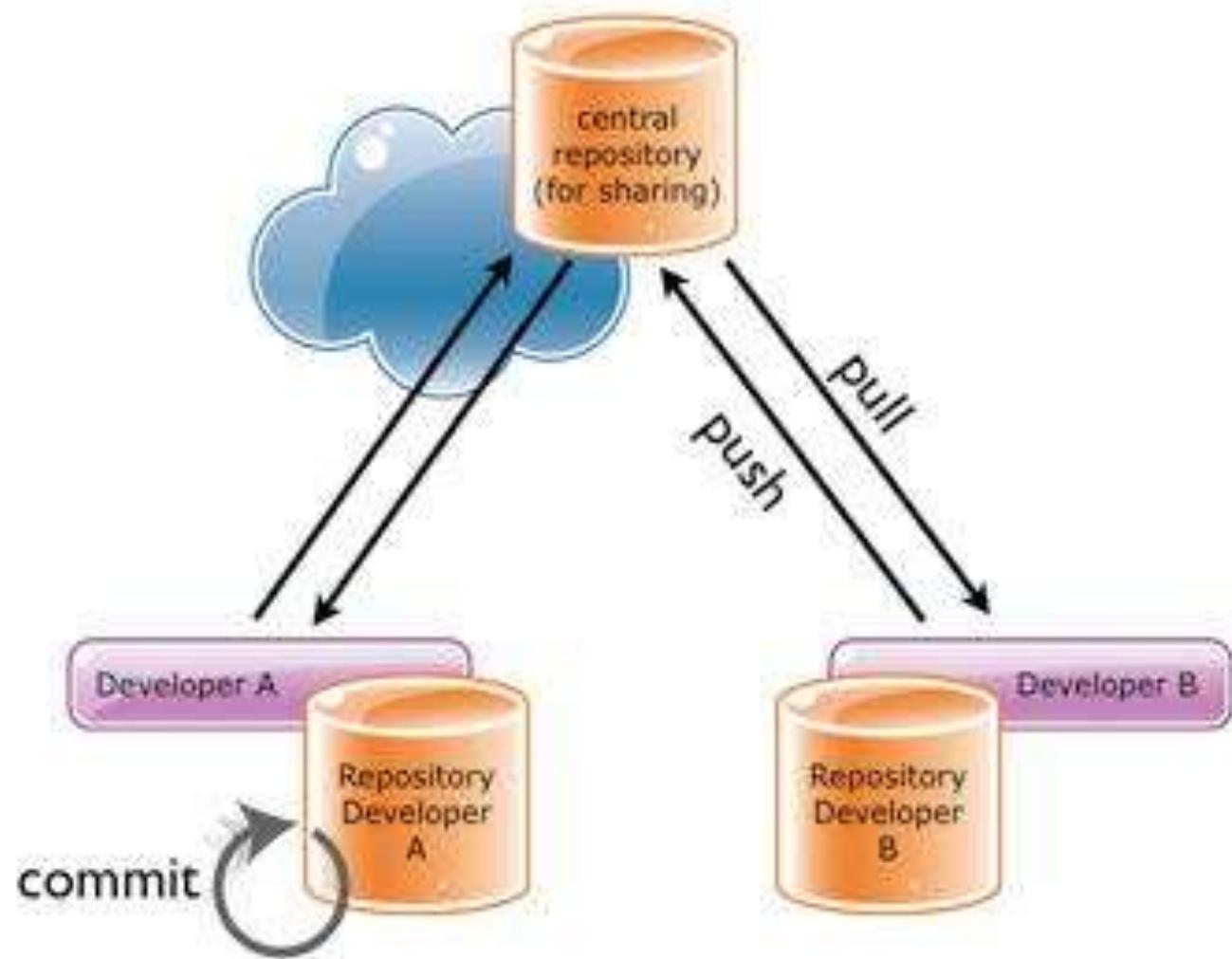
Gestión de la Configuración del Software (GCS)

» Proceso de la GCS 2 - Control de versiones

Puede utilizarse el software

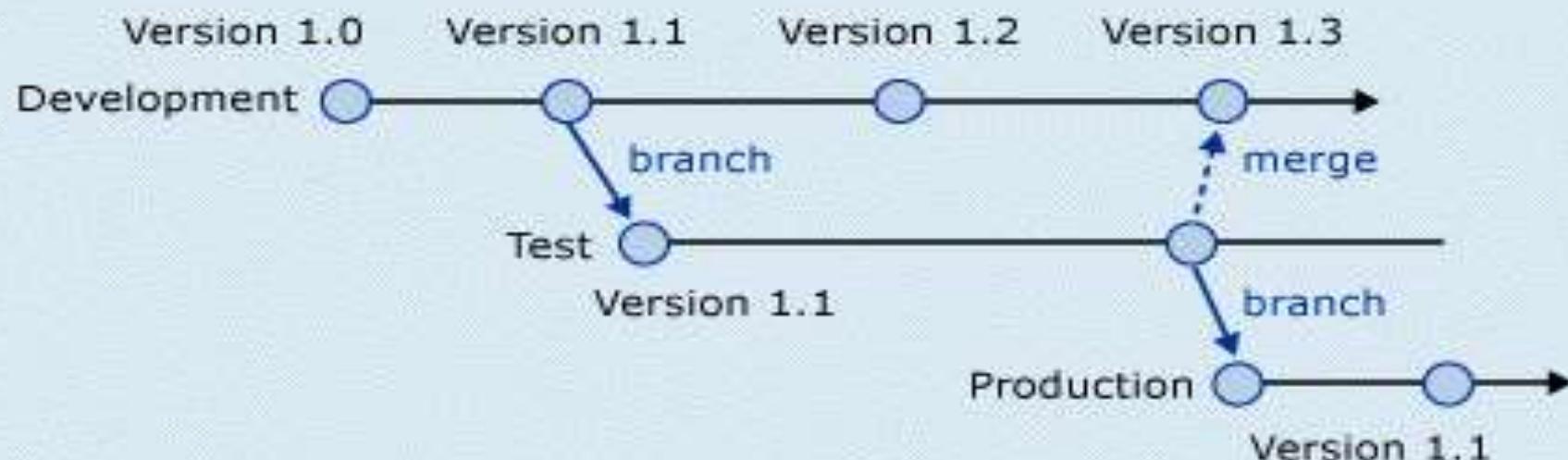
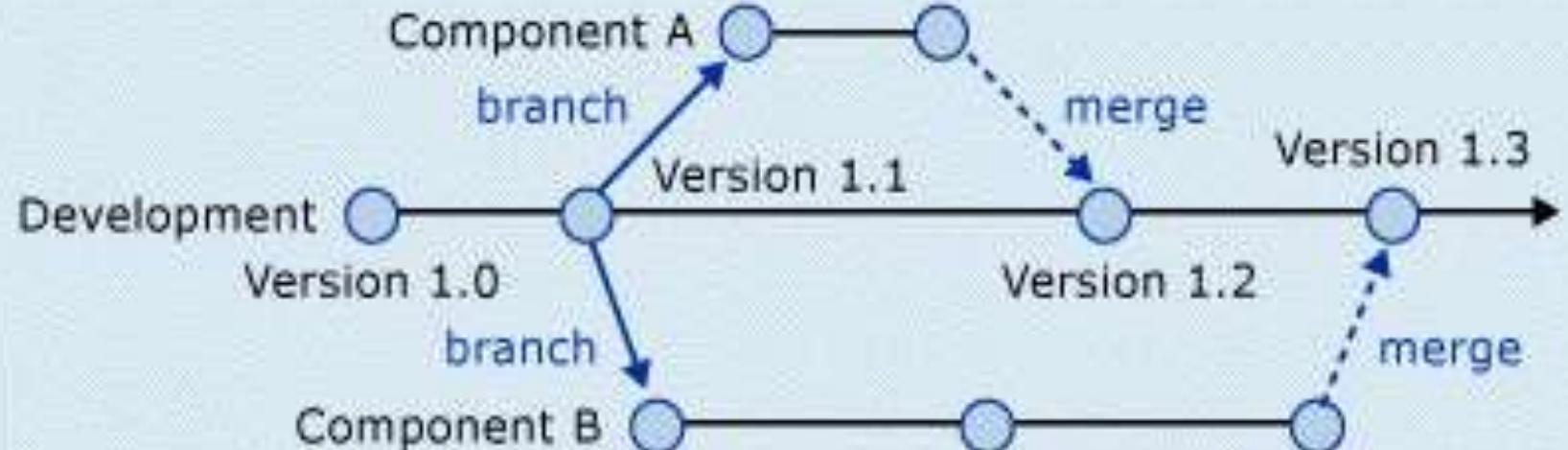
Concurrent Versions System (CVS)

<https://www.nongnu.org/cvs/>



GCS - Proceso

2 - Control de versiones



GCS - Proceso

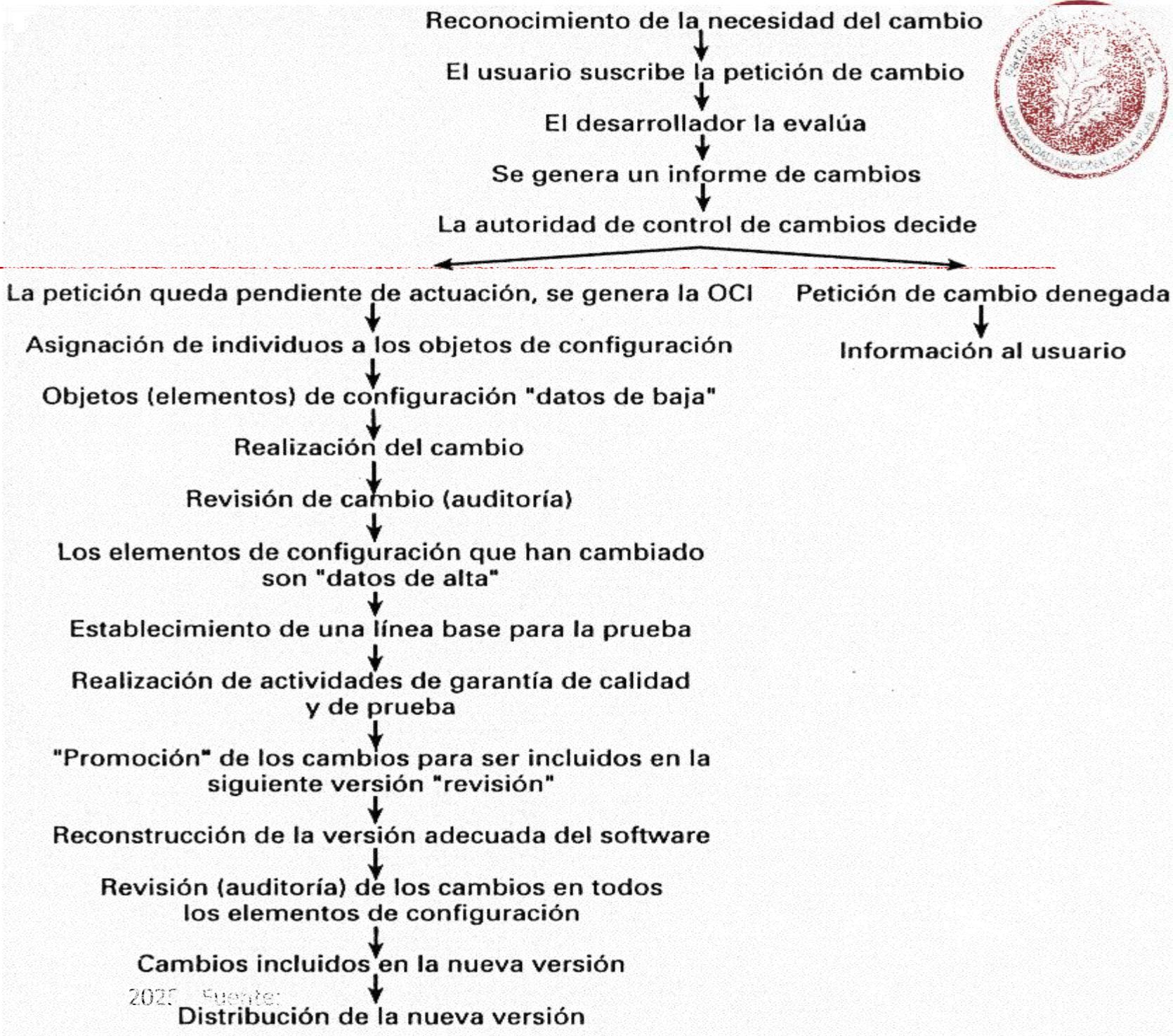
3 - Control de cambios

A lo largo del proyecto los cambios son inevitables y el control es vital para el desarrollo del mismo

Combina los procedimientos humanos y las herramientas adecuadas para proporcionar un mecanismo para el control del cambio

GCS - Proceso

3 -Control de cambios



GCS - Proceso

3 -Control de cambios

La autoridad de control de cambios (ACC) evalúa:

¿Cómo impactará el cambio en el hardware?

¿Cómo impactará el cambio en el rendimiento?

¿Cómo alterará el cambio la percepción del cliente sobre el producto?

¿Cómo afectará el cambio a la calidad y a la fiabilidad?

...

GCS - Proceso

4 - Auditoría de la configuración

La identificación y el control de versiones y el control de cambio, ayudan al equipo de desarrollo de software a mantener un orden, pero sólo se garantiza hasta que se ha generado la orden de cambio.

Cómo aseguramos que el cambio se ha realizado correctamente

Revisiones técnicas formales

Auditorías de configuración

GCS - Proceso

4 - Auditoría de la configuración responde:

¿Se ha hecho el cambio especificado en la Orden de Cambio? ¿Se han incorporado modificaciones adicionales?

¿Se ha llevado a cabo una RTF para evaluar la corrección técnica?

¿Se han seguido adecuadamente los estándares de IS?

¿Se han reflejado los cambios en el ECS: fecha, autor, atributos?

¿Se han seguido procedimientos de GCS para señalar el cambio, registrarlo y divulgarlo?

¿Se han actualizado adecuadamente todos los ECS relacionados?

GCS - Proceso

5 - Generación de informes de estado de la configuración (auditoría)

Responde

¿Qué pasó?

¿Quién lo hizo?

¿Cuándo pasó?

¿Qué más se vio afectado?

La generación de informes de estado de la configuración desempeña un papel vital en el éxito del proyecto



Ingeniería de software II

Gestión De Proyectos

Fuente:

¿Qué es un proyecto ?

»Un proyecto es un esfuerzo temporal que se lleva a cabo para crear un producto, servicio o resultado único.

»Características

Temporal

Tiene un comienzo y fin definido.

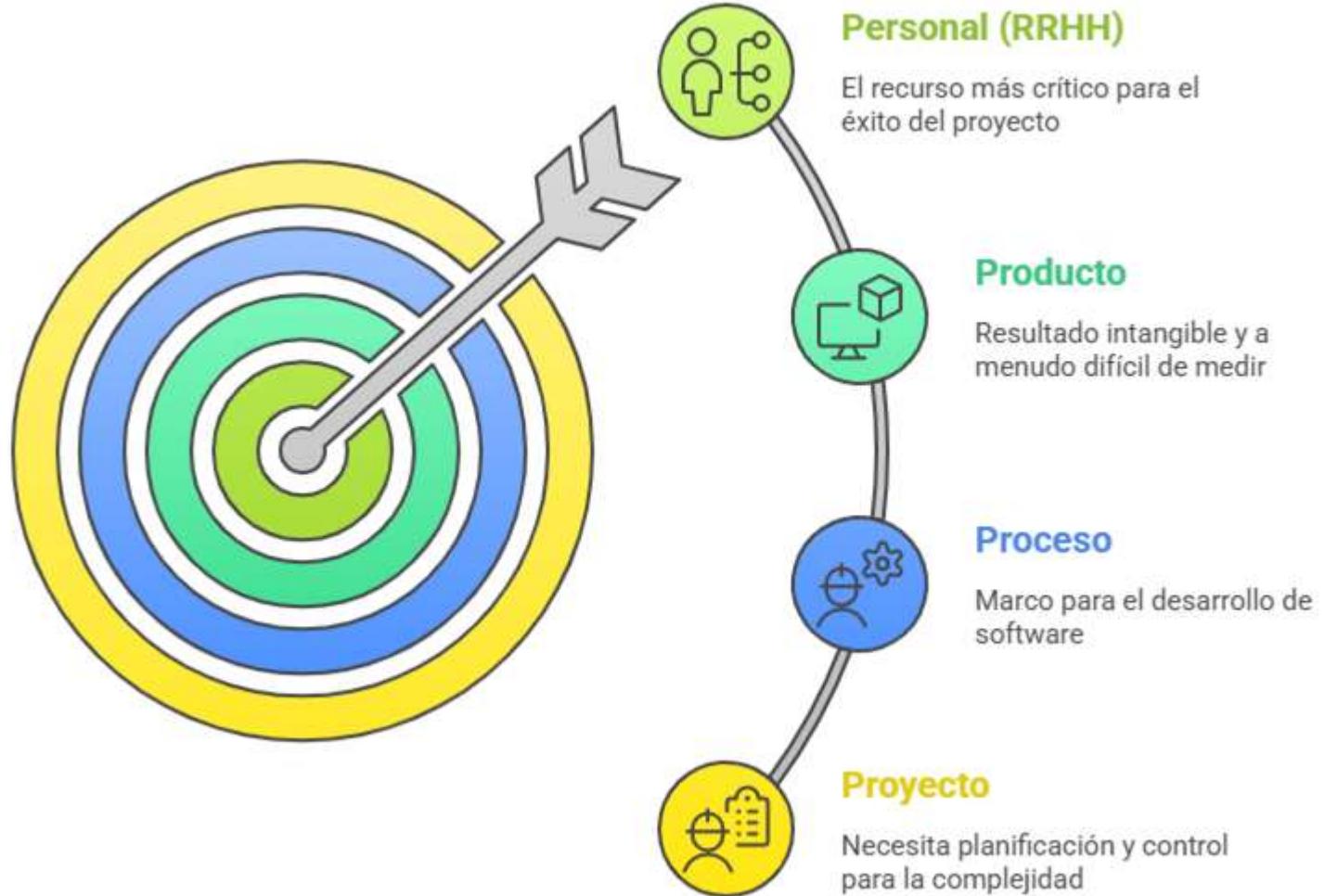
Resultado

Productos, servicios o resultados únicos

Elaboración gradual

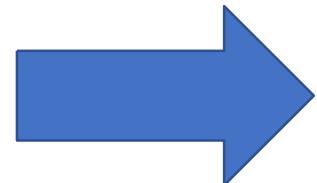
Desarrollar en pasos e ir aumentando mediante incrementos

Las 4 P de la Gestión de Proyectos de Software



Manifestación de una mala gestión de proyectos

- » Incumplimiento de plazos
- » Incremento de los costos
- » Entrega de productos de mala calidad



Perjuicio
económico



Elementos clave de la gestión de proyectos

- » Métricas
 - » Estimaciones
 - » Calendario temporal
 - » Organización del personal
 - » Análisis de riesgos
 - » Seguimiento y control
- }
- PLANIFICACIÓN**

La gestión de proyectos cubre todo el proceso de desarrollo

Gestión de Proyectos

Planificación



Fuente:



Planificación

»La planificación indica:

1. qué debe hacerse,
2. con qué recursos
3. y en qué orden.

Establece una secuencia operativa.

Planificación organizativa

- » El personal que trabaja en una organización de software es el activo más grande, representa el capital intelectual.
- » Una mala administración del personal es uno de los factores principales para el fracaso de los proyectos
- » Debido a su importancia, se creó el Modelo de madurez de capacidades de personal (P-CMM). Se reconoce que toda organización debe mejorar de manera continua su habilidad de atraer, desarrollar, motivar, organizar y conservar el personal.

Planificación organizativa

»Participantes

Gerentes ejecutivos (dueños del producto)

Definen los temas empresariales

Gerentes de proyecto (líderes de equipo)

Planifican, motivan, organizan y controla a los profesionales

Profesionales especializados

Aportan habilidades técnicas

Clientes

Especifican los requerimientos

Usuarios finales

Interactúan con el software

Líderes de equipo

El Equipo de software debe organizarse de manera que maximice las habilidades y capacidades de cada persona.

Las prácticas que realizan líderes ejemplares son:

- ❖ Modelar el camino: Practicar lo que predican. Demostrar compromiso con el equipo y el proyecto.
- ❖ Inspirar y visión compartida: Es importante motivar a los miembros del equipo, esto implica involucrar a las partes al principio del proceso de establecimiento de metas
- ❖ Desafiar el proceso: Tomar la iniciativa de buscar formas innovadoras para mejorar el trabajo. Animar a los miembros a experimentar y asumir riesgos.
- ❖ Permitir que otros actúen: Fomentar habilidades colaborativas del equipo, generando confianza y facilitando relaciones.
- ❖ Alentar: Celebrar los logros individuales. Generar espíritu comunitario.

El equipo de software

La mejor estructura de equipo depende del estilo gerencial de la organización, el número de personas que conformarán el equipo y sus niveles de habilidad, y la dificultad del problema en general.

Como regla general, el equipo no debe tener más de 10 miembros. De todos modos eso depende del tamaño del proyecto.

El equipo de software

Factores a considerar cuando se planea una estructura de equipo

1. Dificultad del problema a resolver
2. Tamaño del programa resultante
3. Tiempo que el equipo permanecerá unido
4. Grado en que puede dividirse en módulos el problema a resolver
5. Calidad y confiabilidad requerida por el sistema a construir
6. Rigidez de la fecha de entrega
7. Grado de sociabilidad requerido para el proyecto

El equipo de software

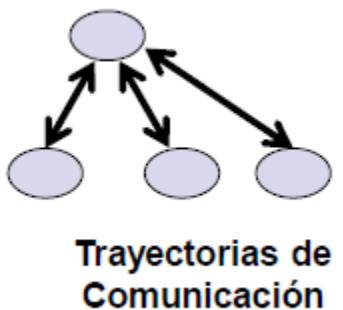
Para evitar :

1. Atmósfera de trabajo frenética-> El equipo tiene acceso a toda la información y las principales metas y objetivos no se modifican a menos que sea absolutamente necesario.
2. Fricción entre los miembros del equipo-> Se define responsabilidad a cada uno
3. Proceso de software fragmentado o mal coordinado-> Entender lo que se va a crear y permitir que el equipo seleccione el modelo de proceso
4. Definición imprecisa de roles-> Definir enfoques correctivos ante un miembro que no cumple
5. Exposición continua y repetida al fracaso->Utilización de técnicas basadas en la retroalimentación y solución de problemas

Paradigmas Organizacionales del equipo

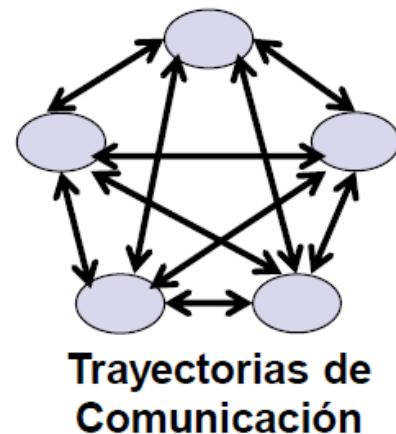
1- **Paradigma cerrado** estructura un equipo conforme a una jerarquía de autoridad tradicional. (Comunicación vertical entre jefe y miembros del equipo)

Tales equipos pueden trabajar bien cuando producen software muy similar al de esfuerzos anteriores, pero será menos probable que sean innovadores cuando trabajen dentro de este paradigma.



Paradigmas Organizacionales del equipo

2- **Paradigma abierto** intenta estructurar un equipo de manera que logre algunos de los controles asociados con el paradigma cerrado, pero también mucha de la innovación que ocurre cuando se usa el paradigma aleatorio. El trabajo se realiza de manera colaborativa; la gran comunicación y la toma de decisiones consensuadas constituyen las características de los equipos de paradigma abierto. Es un equipo democrático y con decisiones consensuadas.



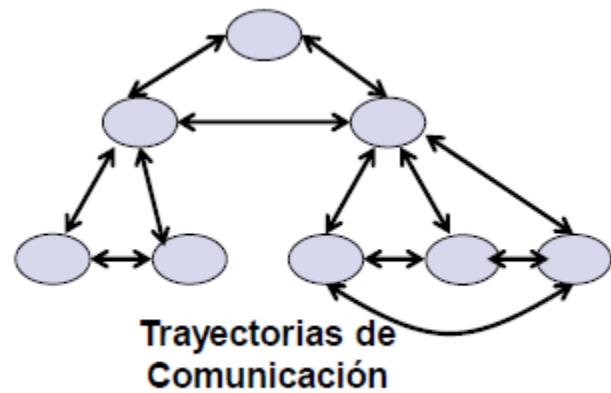
Las estructuras de equipo de este paradigma son muy adecuadas para la solución de problemas complejos, pero pueden no desempeñarse tan eficazmente como otros equipos.

Paradigmas Organizacionales del equipo

3- **Paradigma síncrono** se apoya en la compartimentalización natural de un problema y organiza a los miembros del equipo para trabajar en trozos del problema con poca comunicación activa entre ellos. Es un equipo descentralizado y controlado a la vez, utiliza la política de divide y vencerás.

Este tipo de organizaciones bueno cuando la complejidad y tamaño del problema es elevada.

El fraccionamiento del problema puede llevar a crear un producto no requerido debido a que los subgrupos no tienen adecuada comunicación.



Paradigmas Organizacionales del equipo

4- **Paradigma aleatorio** estructura un equipo de manera holgada y depende de la iniciativa individual de los miembros del equipo. Los miembros se organizan entre ellos y no se requiere de un líder definido. Las trayectorias de comunicación pueden ser cualquiera de los otros tres paradigmas.

Cuando se requiere innovación o avance tecnológico, destacarán los equipos que siguen este paradigma, pero pueden batallar cuando se requiera “desempeño ordenado”.

Resumen: Paradigmas Organizacionales del equipo



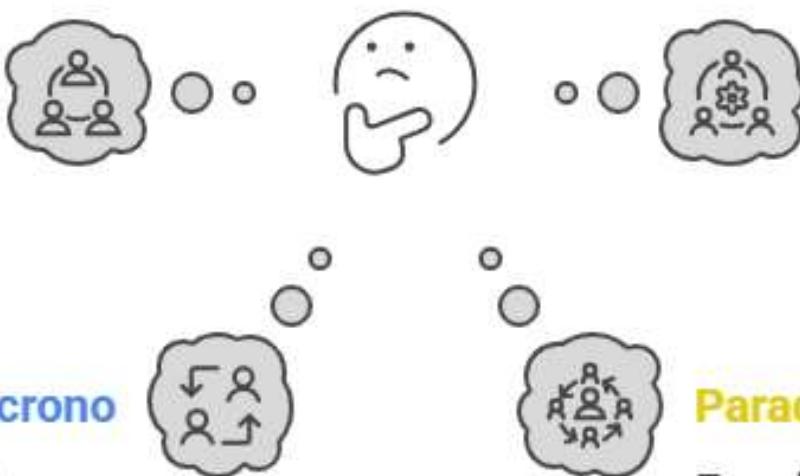
¿Qué paradigma organizacional debería adoptar el equipo?

Paradigma Cerrado

Adecuado para proyectos de software similares a los anteriores, pero puede limitar la innovación.

Paradigma Síncrono

Efectivo para problemas grandes y complejos con menos comunicación, pero riesgo de falta de coordinación.



Paradigma Abierto

Fomenta la innovación y la colaboración, ideal para la resolución de problemas complejos.

Paradigma Aleatorio

Excelente para la innovación, pero puede carecer de orden en el rendimiento.

El equipo de software

Sin importar la organización del equipo, el objetivo a alcanzar es lograr un equipo que muestre cohesión. Que se consolide.

Los miembros de un equipo consolidado son mucho más productivos y motivados.

Un grupo **cohesivo** tiene beneficios:

1. Puede establecer sus propios estándares de calidad
2. Los individuos aprenden de los demás y se apoyan mutuamente
3. El conocimiento se comparte
4. Se alienta la refactorización y el mejoramiento continuo

El equipo de software

Sin importar la organización del equipo, el objetivo a alcanzar es lograr un equipo que muestre cohesión. Que se consolide.

Los miembros de un equipo consolidado son mucho más productivos y motivados.

Un equipo no consolidado suele sufrir toxicidad de equipo :

1. Atmósfera de trabajo frenética
2. Fricción entre los miembros del equipo
3. Proceso de software fragmentado o mal coordinado
4. Definición imprecisa de roles
5. Exposición continua y repetida al fracaso

El equipo de software

»Comunicación Grupal

Las comunicaciones en un grupo se ven influenciadas por factores como: status de los miembros del grupo, tamaño del grupo, composición de hombres y mujeres, personalidades y canales de comunicación disponible.

Se deben establecer mecanismos para la comunicación formal e informal entre los miembros del equipo

La comunicación formal: a través de reuniones, escritos y otros canales no interactivos

La comunicación informal: Los miembros comparten ideas sobre la marcha.

El equipo de software ágil

Muchas organizaciones de software defienden el desarrollo ágil de software, lo que conlleva a crear equipos pequeños y muy motivados, lo que se denomina equipo ágil.

- Se hace hincapié en la competencia individual y colaboración grupal.
- Son autoorganizados.

El equipo de software ágil



Se parece a un
paradigma
aleatorio con
pocos miembros



Ingeniería de software II

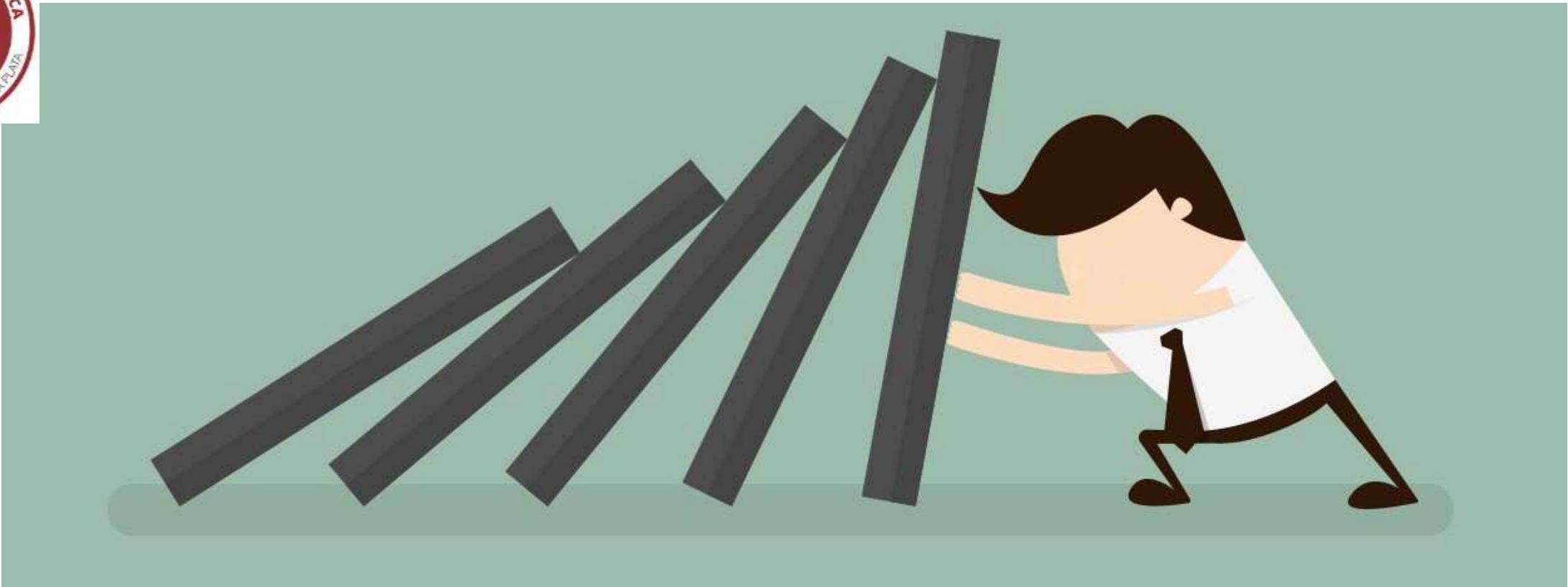
Gestión del Proyecto

Riesgos

2025



Riesgos



Gestión de Riesgos

- ¿Qué es un riesgo?
- Un riesgo es un evento no deseado que tiene consecuencias negativas.

Gestión de Riesgos

- Los gerentes deben *determinar* si pueden presentarse eventos no deseados durante el desarrollo o el mantenimiento, y *hacer planes* para evitar estos eventos, o, si son inevitables, minimizar sus consecuencias negativas.
- ANTICIPAR / EVITAR





Gestión de Riesgos en el desarrollo de software. “El riesgo concierne...

“...a lo que ocurrirá en el futuro”.

- ¿Cuáles son los riesgos que pueden hacer que fracase el proyecto?.

“...a como afectarán los cambios al desarrollo”

- ¿Cómo afectarán al éxito global y a los plazos los cambios en los requisitos del cliente, en las tecnologías de desarrollo, etc.?.

“....a las elecciones”

- ¿Qué métodos y herramientas debemos usar, cuánta gente debe estar involucrada, cuánta importancia hay que darle a la calidad?.

Gestión de Riesgos en el desarrollo de software



La “*deuda técnica*” es el término que se utiliza para describir los costos asociados al aplazamiento de actividades, tales como documentación y refactorización del software.



La *deuda técnica* que no se paga, resulta en un producto de mala calidad, documentación insuficiente, complejidad innecesaria.

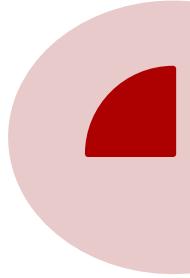


La *deuda técnica* implica que los costos (esfuerzo, tiempo, recursos) de luchar con temas técnicos se puede reducir si se afrontan los problemas al principio, en vez de dejarlos para el final.



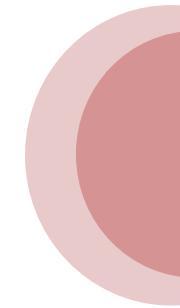
El desarrollo ágil no implica dejar de lado la gestión de riesgos, ya que podría llevar a obtener una *deuda técnica*

Gestión de Riesgos – Estrategias



Reactivas

reaccionar ante el problema y “gestionar la crisis” (Indiana Jones...).

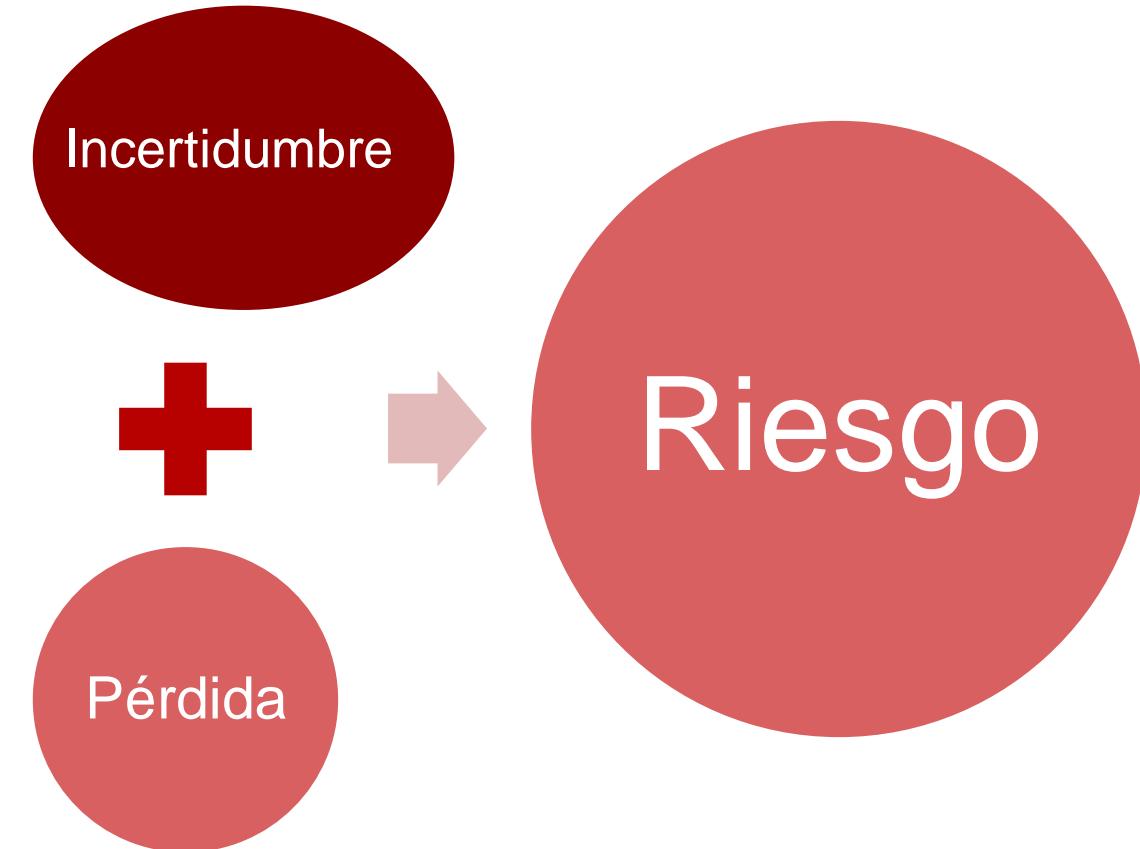


Proactivas

tener estrategias de tratamiento.



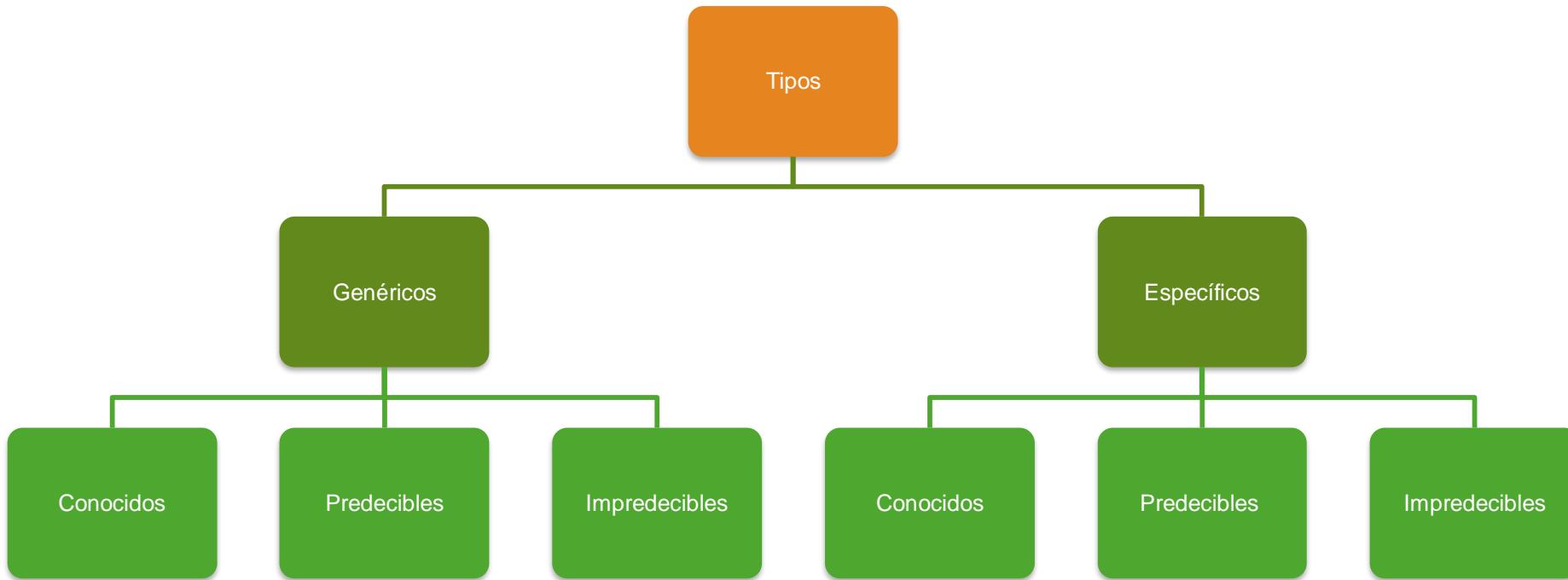
Riesgos de software



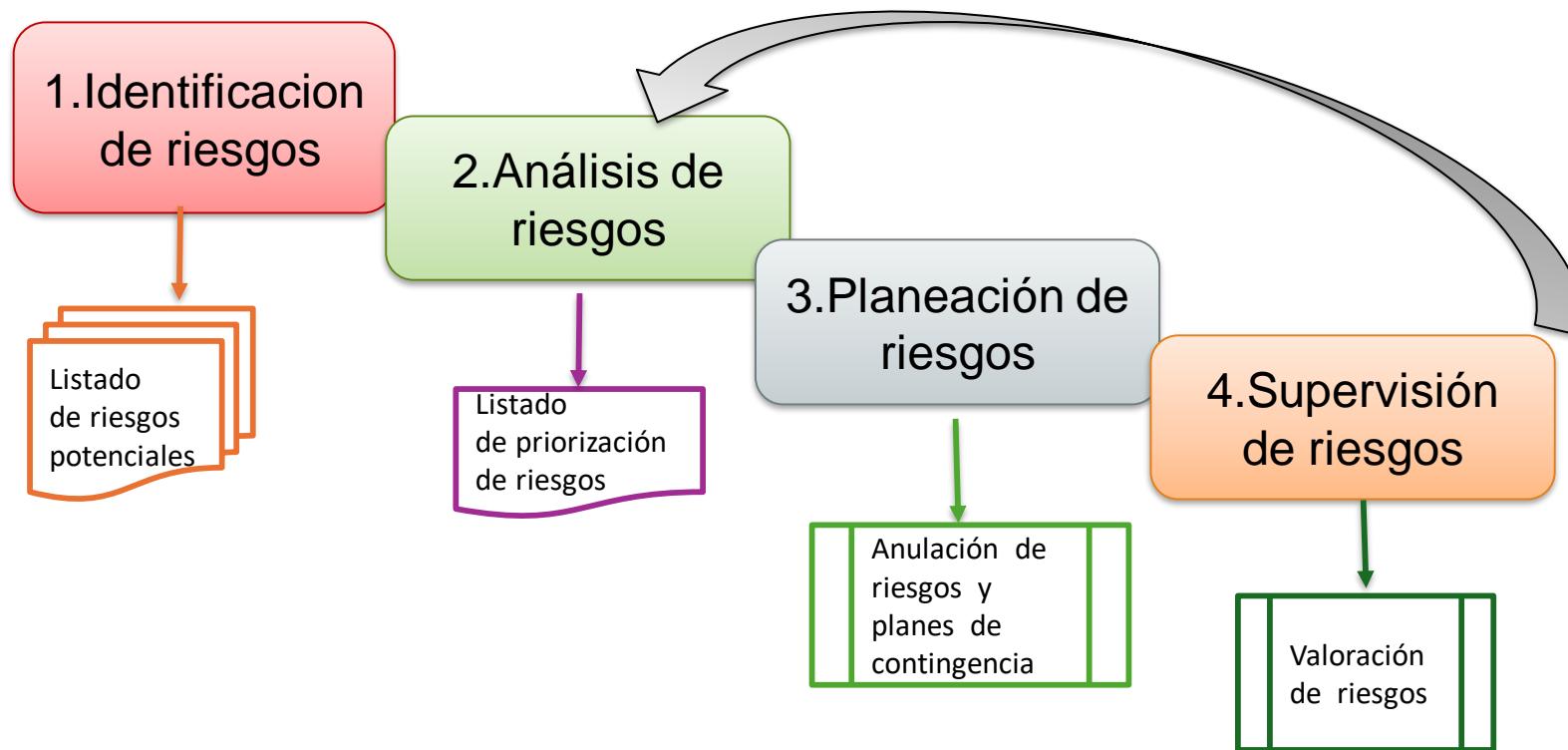
Categorización de los riesgos en el desarrollo de Software



Tipos de Riesgos



El Proceso de Gestión de Riesgos

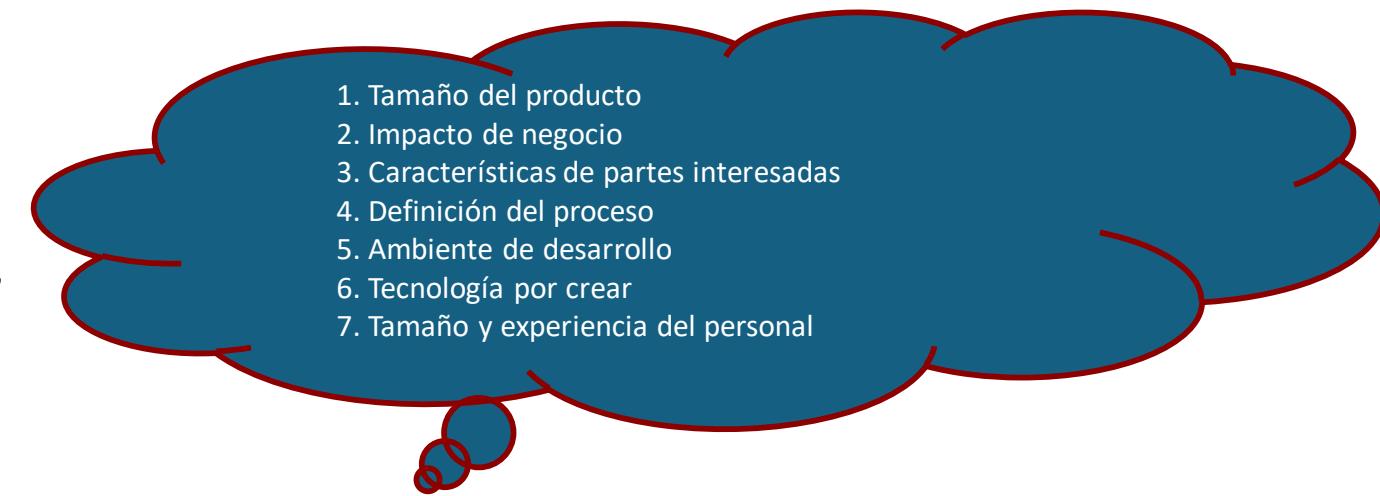


Proceso iterativo que debe documentarse

Administración de Riesgos

1. Identificación de riesgos

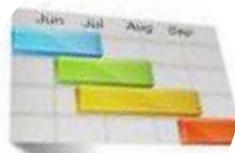
- “verdaderos riesgos”.
- “lista de comprobación de elementos de riesgo”



• Utiliza un enfoque de *tormenta de ideas* o en base a la *experiencia*.

Administración de Riesgos

1. Identificación de riesgos - Categorías



Del proyecto



Del producto



Del negocio



- Riesgos conocidos*
- Riesgos predecibles*
- Riesgos impredecibles*



Administración de Riesgos

1. Identificación de riesgos - Preguntas

- ✓ ¿Los gerentes de software y de cliente se reunieron formalmente para apoyar el proyecto?
- ✓ ¿Los usuarios finales se comprometen con el proyecto y sistema/producto que se va a construir?
- ✓ ¿El equipo y sus clientes entienden por completo los requisitos?
- ✓ ¿Los clientes se involucraron plenamente en la definición de los requisitos?
- ✓ ¿Los usuarios finales tienen expectativas realistas?
- ✓ ¿El ámbito del proyecto es estable?
- ✓ ¿El equipo tiene la mezcla correcta de habilidades?
- ✓ ¿Los requisitos del proyecto son estables?
- ✓ ¿El equipo tiene experiencia con la tecnología que se va a implementar?
- ✓ ¿El número de personas que hay en el equipo es adecuado para hacer el trabajo?
- ✓ ¿Todos los clientes/usuarios están de acuerdo en la importancia del proyecto y en los requisitos para el sistema/producto que se va a construir?

1.Identificación de riesgos

Listado de riesgos potenciales

Si la respuesta a alguna de estas preguntas es negativa, estamos frente a un/unos riesgo/s inminente/s. El grado de riesgo es directamente proporcional al nro. de respuestas negativas.

Categorización de los riesgos

Riesgo	Repercute en	Descripción
Rotación de personal	Proyecto	Personal experimentado abandonará el proyecto antes de que éste se termine.
Cambio administrativo	Proyecto	Habrá un cambio de gestión en la organización con diferentes prioridades.
Indisponibilidad de hardware	Proyecto	Hardware, que es esencial para el proyecto, no se entregará a tiempo.
Cambio de requerimientos	Proyecto y producto	Habrá mayor cantidad de cambios a los requerimientos que los anticipados.
Demoras en la especificación	Proyecto y producto	Especificaciones de interfaces esenciales no están disponibles a tiempo.
Subestimación del tamaño	Proyecto y producto	Se subestimó el tamaño del sistema.
Bajo rendimiento de las herramientas CASE	Producto	Las herramientas CASE, que apoyan el proyecto, no se desempeñan como se anticipaba.
Cambio tecnológico	Empresa	La tecnología subyacente sobre la cual se construye el sistema se sustituye con nueva tecnología.
Competencia de productos	Empresa	Un producto competitivo se comercializa antes de que el sistema esté completo.

Administración de Riesgos

2. Análisis de riesgos

Cada riesgo identificado

- ❖ probabilidad
- ❖ impacto.
- ❖ Se construye la tabla

1ra Columna	2da columna	3ra columna	4ta columna
todos los riesgos en desorden.	categoría del riesgo	probabilidad estimada del riesgo. (por consenso, o individualmente y sacar un promedio).	impacto

Riesgos	Categoría	Probabilidad	Impacto
El cliente cambiará los requisitos			
Falta de formación en las herramientas			

2. Análisis de riesgos

Listado de priorización de riesgos

Administración de Riesgos

2. Análisis de riesgos

- ❖ Establecer una escala que refleje la probabilidad observada de un riesgo

Bastante improbable : < 10%

Improbable : 10-25%

Moderado : 25-50%

Probable : 50-75%

Bastante probable : >75%

2. Análisis de riesgos

Listado de priorización de riesgos

Riesgos	Categoría	Probabilidad	Impacto
El cliente cambiará los requisitos	Proy	80%	
Falta de formación en las herramientas	Proy	80%	

Administración de Riesgo

2. Análisis de riesgos

Estimar el impacto en el proyecto:

Se ordena la lista por probabilidad e impacto y se traza una línea de corte.

- 1- **Catastrófico**: cancelación del proyecto
- 2- **Serio**: reducción de rendimiento, retrasos en la entrega, excesos importante en costo
- 3- **Tolerable**: reducciones mínimas de rendimiento, posibles retrasos, exceso en costo
- 4 –**Insignificante**: incidencia mínima en el desarrollo

Riesgos	Categoría	Probabilidad	Impacto
El cliente cambiará los requisitos	Proy	80%	2
Falta de formación en las herramientas	Proy	80%	3

Administración de Riesgos

2. Análisis de riesgos

Boehm recomienda

- ❖ identificar y supervisar los 10 riesgos más altos
- ❖ El número exacto de riesgos debe depender del proyecto.

No obstante debe ser un
número manejable.

- ❖ Los riesgos que queden encima de la línea serán los que se les preste atención.
- ❖ Los que queden debajo de la línea serán reevaluados y tendrán una prioridad de segundo orden.



Administración de Riesgos

2. Análisis de riesgos

Un factor de riesgo que tenga

Gran impacto pero poca probabilidad de que ocurra, no debería absorber un tiempo significativo

Los riesgos de gran impacto con una probabilidad de moderada a alta y los riesgos de poco impacto pero con gran probabilidad deberían tomarse en cuenta.



2. Análisis de riesgos

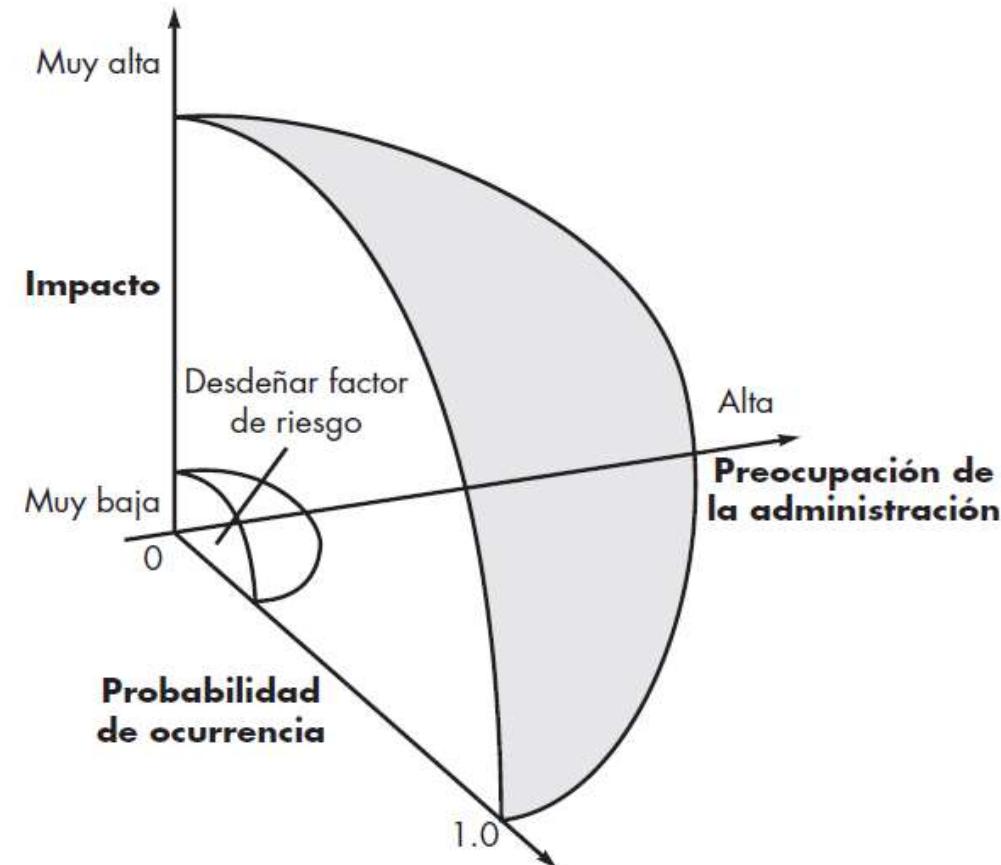
Listado de priorización de riesgos

Administración de Riesgos

2. Análisis de riesgos

2. Análisis de riesgos

Listado de priorización de riesgos



Ejemplo

Riesgos	Categoría	Probabilidad	Impacto
El cliente cambiará los requisitos	Proy	80 %	2
Falta de formación en las herramientas	Prod	80%	3
Menos reutilización de la prevista	Proy	70 %	2
La estimación del tamaño puede ser muy baja	Proy	60 %	2
Habrá muchos cambios de personal	Proy	60 %	2
La fecha de entrega estará muy ajustada	Proy	50%	2
Se perderán los presupuestos	Neg	40%	1

Línea de
Corte

Ejemplo

Línea de
Corte

Los usuarios finales se resisten al sistema	Neg	40%	3
La tecnología no alcanzará las expectativas	Prod	30%	1
Personal sin experiencia	Proy	30%	2
Mayor número de usuarios de los previstos	Neg	30%	3

Administración de Riesgos

3. Planeación

3. Planeación de riesgos



Anulación de riesgos y planes de contingencia	
---	--

Se consideran cada uno de los riesgos por encima de la línea de corte y se determina una estrategia a seguir:

Evitar el riesgo

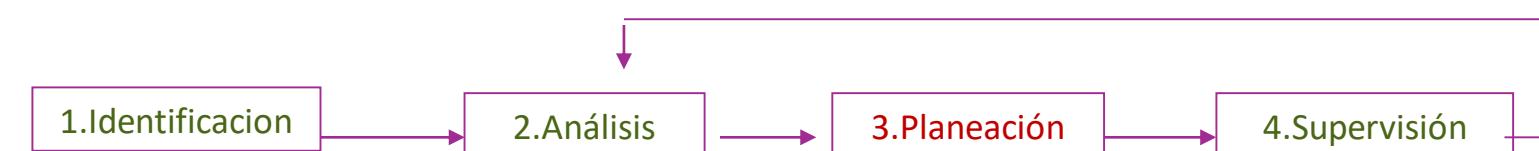
Siguiendo esta estrategia, el sistema se diseña de modo que no pueda ocurrir el evento.

Minimizar el riesgo

Siguiendo esta estrategia, la probabilidad que el riesgo se presente se reduce.

Plan de contingencia

Siguiendo esta estrategia se está preparado para lo peor. Se acepta la aparición del riesgo y es tratado de manera de minimizar las consecuencias.



Administración de Riesgos

3. Planeación

3. Planeación de riesgos

Anulación de riesgos y planes de contingencia

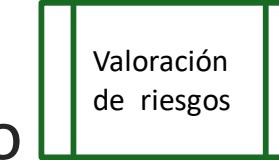
Riesgo	Estrategia
Problemas financieros de la organización	Prepare un documento informativo para altos ejecutivos en el que muestre cómo el proyecto realiza una aportación muy importante a las metas de la empresa y presente razones por las que los recortes al presupuesto del proyecto no serían efectivos en costo.
Problemas de reclutamiento	Alerte al cliente de dificultades potenciales y de la posibilidad de demoras; investigue la compra de componentes.
Enfermedad del personal	Reorganice los equipos de manera que haya más traslape de trabajo y, así, las personas comprendan las labores de los demás.
Componentes defectuosos	Sustituya los componentes potencialmente defectuosos con la compra de componentes de conocida fiabilidad.
Cambios de requerimientos	Obtenga información de seguimiento para valorar el efecto de cambiar los requerimientos; maximice la información que se oculta en el diseño.
Reestructuración de la organización	Prepare un documento informativo para altos ejecutivos en el que muestre cómo el proyecto realiza una aportación muy importante a las metas de la empresa.
Rendimiento de la base de datos	Investigue la posibilidad de comprar una base de datos de mayor rendimiento.
Subestimación del tiempo de desarrollo	Investigue los componentes comprados; indague el uso de un generador de programa.

Administración de Riesgos

4. Supervisión

- ❖ Evaluar si ha cambiado la probabilidad de cada riesgo
- ❖ Evaluar la efectividad de las estrategias propuestas.
- ❖ Detectar la ocurrencia de un riesgo que fue previsto
- ❖ Asegurar que se están cumpliendo los pasos definidos para cada riesgo
- ❖ Recopilar información para el futuro
- ❖ Determinar si existen nuevos riesgos
- ❖ Reevaluar periódicamente los riesgos

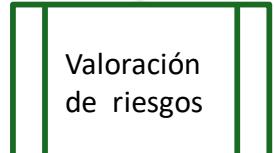
4. Supervisión
de riesgos



Administración de Riesgos

4. Supervisión

4. Supervisión
de riesgos



- ❖ Los riesgos deben monitorizarse comúnmente en todas las etapas del proyecto. En cada revisión administrativa, es necesario reflexionar y estudiar cada uno de los riesgos clave por separado.
- ❖ También hay que decidir si es más o menos probable que surja el riesgo, y si cambiaron la gravedad y las consecuencias del riesgo



Ejercicio de gestión de riesgos

Estás liderando un equipo de desarrollo de software encargado de construir un Sistema de Gestión de Stock para una cadena de tiendas minoristas de productos de limpieza. Cada tienda utilizará el mismo sistema. El sistema debe gestionar el inventario de productos, realizar ventas, actualizar automáticamente el stock, generar informes de ventas e inventario y órdenes de reposición de stock. Debe integrarse al sistema central para recopilar la información de cada stock de cada tienda.

Identifica y analiza posibles riesgos que podrían surgir durante el desarrollo del proyecto. Utiliza la metodología de análisis de riesgos propuesta que incluye la identificación de riesgos, la categorización del riesgo (proyecto, producto, negocio/empresa), la asignación de la probabilidad de que ocurra y la evaluación del impacto (catastrófico, serio, tolerable, insignificante). Finalmente agrega la planeación del riesgo



Ejercicio de gestión de riesgos

- 1. Armar listado de posibles riesgos, ¿Cuáles pueden ser?. Cada uno anota los posibles riesgos y luego comparamos entre todos.**



Ejercicio 1. Identificación de riesgos

1. Requerimientos Cambiantes:

Justificación: La experiencia muestra que los cambios en los requisitos del cliente pueden surgir durante el desarrollo del proyecto debido a una comprensión incompleta de las necesidades iniciales o a cambios en el entorno del negocio. Estos cambios pueden impactar significativamente en la planificación y la ejecución del proyecto.

2. Falta de Experiencia en Tecnologías Nuevas:

Justificación: La falta de experiencia en las tecnologías seleccionadas aumenta el riesgo de errores técnicos, demoras y problemas de calidad. La adopción de nuevas tecnologías sin un entendimiento adecuado puede llevar a la toma de decisiones incorrectas y afectar negativamente la entrega del producto.

3. Problemas de Integración:

Justificación: La integración de sistemas puede ser compleja, especialmente en proyectos que involucran múltiples componentes o sistemas externos. Los problemas de integración pueden generar fallos en el funcionamiento del sistema y afectar la experiencia del usuario y la operatividad del negocio.

4. Falta de Recursos Humanos:

Justificación: La escasez de personal cualificado puede afectar la productividad y la calidad del trabajo. Los proyectos de desarrollo de software requieren un equipo competente, y la falta de recursos humanos adecuados en momentos críticos puede impactar negativamente en la ejecución del proyecto.



Ejercicio 1. Identificación de riesgos

5. Cambio en las Tendencias del Mercado:

Justificación: Los cambios en las tendencias del mercado pueden afectar la demanda de productos o servicios. La falta de adaptación a estas tendencias puede llevar a la obsolescencia del producto o a la pérdida de cuota de mercado, lo que afectaría directamente al éxito del negocio.

6. Fallos en la Seguridad del Producto:

Justificación: Con el aumento de las amenazas ciberneticas, los fallos en la seguridad del producto pueden resultar en pérdida de datos, violaciones de privacidad y daño a la reputación de la empresa. La seguridad del producto es crucial para proteger la información sensible y mantener la confianza del cliente.

7. Rendimiento Inadecuado del Sistema:

Justificación: Un rendimiento deficiente del sistema puede afectar la experiencia del usuario y la eficiencia operativa. La lentitud o los fallos en el rendimiento pueden disminuir la satisfacción del cliente y afectar la percepción del producto.

8. Problemas de Escalabilidad:

Justificación: La falta de escalabilidad puede limitar el crecimiento del sistema y causar problemas cuando la demanda aumenta. La capacidad del sistema para manejar un mayor volumen de usuarios o datos es esencial para el éxito a largo plazo.



Ejercicio 1. Identificación de riesgos

9. Cambios en la Legislación Fiscal:

Justificación: Los cambios en la legislación fiscal pueden tener un impacto significativo en los costos y la estructura financiera de la empresa. No adaptarse a estos cambios puede resultar en sanciones financieras y problemas legales.

10. Problemas de Suministro:

Justificación: Dependiendo de la cadena de suministro para el aprovisionamiento de materiales críticos, cualquier interrupción en la cadena puede resultar en retrasos en la producción, pérdida de ingresos y afectar la capacidad de cumplir con la demanda del mercado



Ejercicio de gestión de riesgos

2. Analizamos los riesgos, armamos tabla y colocamos: riesgo, categoría, probabilidad e impacto.

Ejercicio 2. Análisis de Riesgos

No.	Riesgo	Categoría	Probabilidad	Impacto
1	Requerimientos Cambiantes	Proyecto	60%	Serio
2	Falta de Experiencia en Tecnologías Nuevas	Producto	70%	Catastrófico
3	Problemas de Integración	Proyecto	75%	Serio
4	Falta de Recursos Humanos	Proyecto	70%	Serio
5	Cambio en las Tendencias del Mercado	Negocio/Empresa	80%	Serio
6	Fallos en la Seguridad del Producto	Producto	60%	Serio
7	Rendimiento Inadecuado del Sistema	Producto	70%	Serio
8	Problemas de Escalabilidad	Producto	40%	Moderado
9	Cambios en la Legislación Fiscal	Negocio/Empresa	40%	Serio
10	Problemas de Suministro	Negocio/Empresa	60%	Serio



Ejercicio de gestión de riesgos

2. Analizamos los riesgos: ¿Cuál es la línea de corte del ejemplo?



Ejercicio 2. Línea de corte

Riesgos con una probabilidad alta (superior al 70%) y un impacto serio o catastrófico se considerarán críticos.

1. Falta de Experiencia en Tecnologías Nuevas:

1. Probabilidad: 70%
2. Impacto: Catastrófico

2. Cambios en las Tendencias del Mercado:

1. Probabilidad: 80%
2. Impacto: Serio

3. Problemas de Integración:

1. Probabilidad: 75%
2. Impacto: Serio

4. Falta de Recursos Humanos:

1. Probabilidad: 70%
2. Impacto: Serio

5. Rendimiento Inadecuado del Sistema:

1. Probabilidad: 70%
2. Impacto: Serio



Ejercicio de gestión de riesgos

3. Planificamos los riesgos: ¿Cuales son cada una de las estrategias de mitigación para los riegos por arriba de la línea de corte?

Ejercicio 3. Planeación

Riesgo	Estrategias de Mitigación
Falta de Experiencia en Tecnologías Nuevas	- Contratar o capacitar al personal con experiencia relevante. - Realizar pruebas de concepto y prototipos. - Establecer colaboraciones con expertos externos.
Cambios en las Tendencias del Mercado	- Formar un equipo de inteligencia de mercado. - Diseñar el producto de manera flexible. - Diversificar la oferta de productos.
Problemas de Integración	- Implementar pruebas continuas de integración. - Mantener una comunicación constante con propietarios de sistemas externos.- Designar un equipo dedicado para la gestión de la integración.
Falta de Recursos Humanos	- Implementar una estrategia de adquisición temprana. - Desarrollar planes de contingencia. - Distribuir tareas críticas entre varios miembros del equipo.
Rendimiento Inadecuado del Sistema	- Realizar pruebas de rendimiento exhaustivas. - Implementar herramientas de monitoreo continuo. - Mantener un enfoque proactivo en la optimización del código y la arquitectura.



Ingeniería de software II

Gestión del Proyecto
Planificación temporal



Planificación Temporal

Ingeniería de Software II – 2025



Planificación Temporal

Es una actividad que distribuye el esfuerzo estimado a lo largo de la duración prevista del proyecto

La precisión de la planificación temporal es muy importante para no generar clientes insatisfechos, costos adicionales, reducción del impacto en el mercado, etc.

Planificación Temporal

- ❖ La ***calendarización del proyecto*** de software es una acción que distribuye el esfuerzo estimado a través de la duración planificada del proyecto, asignando el esfuerzo a ***tareas*** específicas del desarrollo de software.

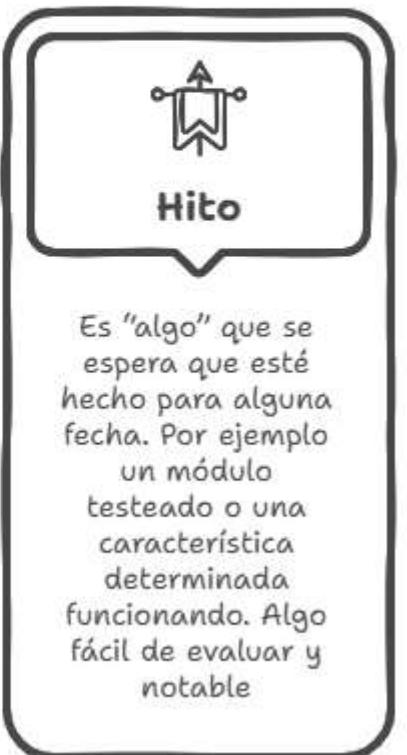
Calendarización

- » Las fechas de los proyectos pueden ser de dos tipos:



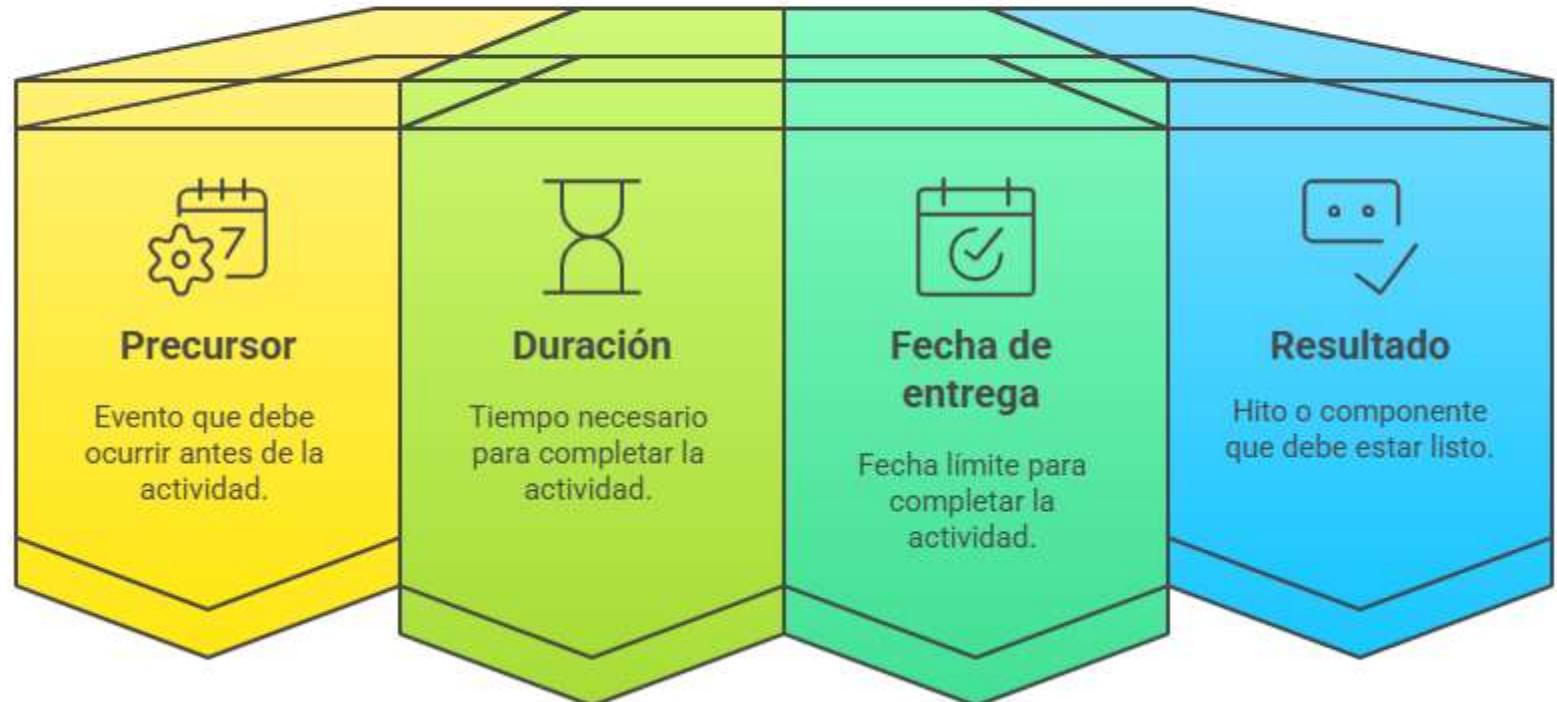
Planificación Temporal-Calendarización

Planificación temporal
está compuesta por:



Planificación Temporal

Una tarea se describe por 4 parámetros



Made with ➡ Napkin

Planificación Temporal-Calendarización

Red de tareas

- ❖ Es una representación gráfica del flujo de las tareas desde el inicio hasta el fin de un proyecto
- ❖ En algunos casos los conjuntos de tareas permiten realizar algunas actividades en paralelo.
- ❖ Representan la secuencia de las tareas y su interdependencia

Planificación Temporal-Calendarización

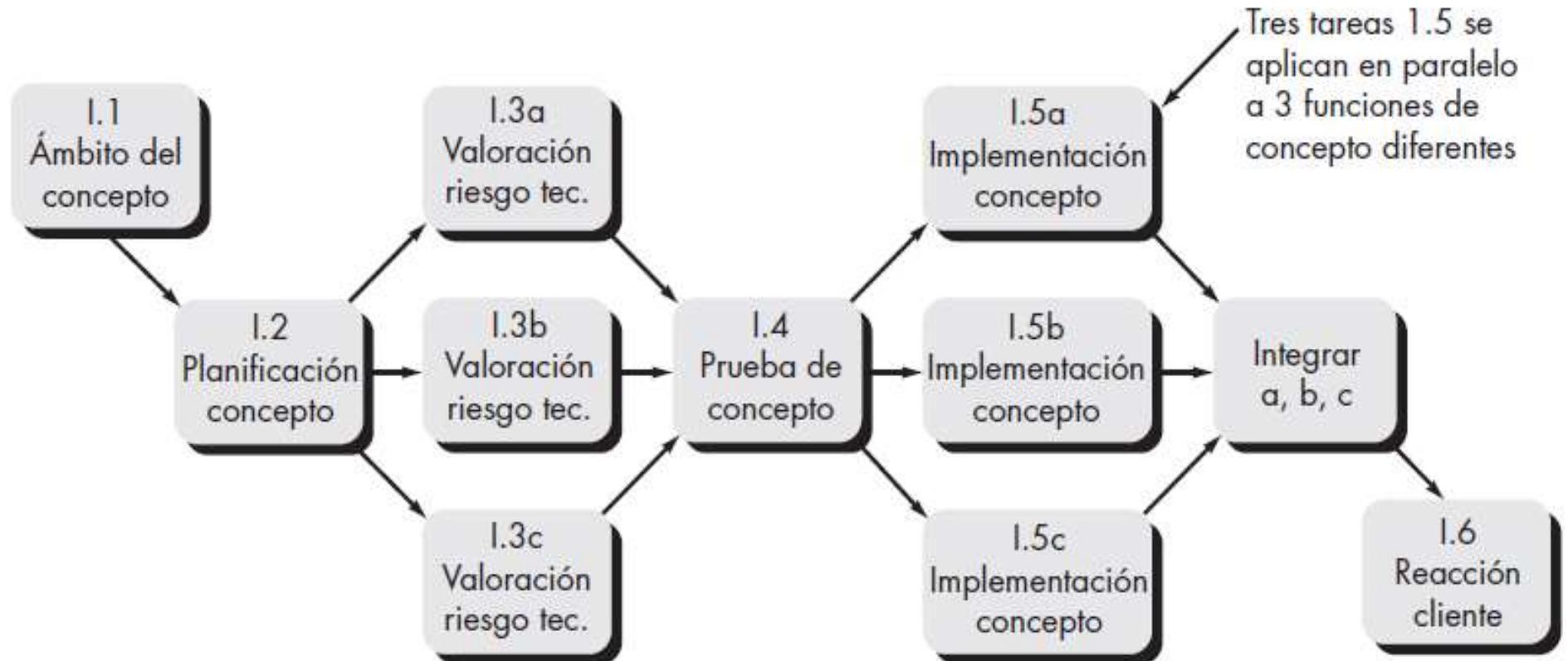
El conjunto de tareas variará dependiendo del tipo de proyecto y grado de rigor.

Los factores que influyen en el conjunto de tareas a elegir son :

- ❖ Tamaño del proyecto
 - ❖ Número de usuarios potenciales
 - ❖ Criticidad del proyecto
 - ❖ Estabilidad de los requerimientos
 - ❖ Facilidad de comunicación con el cliente/usuario
 - ❖ Madurez de la tecnología aplicable
 - ❖ Restricciones
- entre otros

Planificación Temporal

Red de tareas



Método de planificación temporal

GANTT

Un diagrama de Gantt es una herramienta visual de gestión de proyectos utilizada para planificar y rastrear el progreso de diversas tareas y actividades dentro del ciclo de vida de un proyecto de software.

Funciona como una línea de tiempo visual, ofreciendo una visión general de alto nivel de los cronogramas del proyecto, lo que simplifica la gestión de planes complejos que involucran múltiples equipos y plazos cambiantes.

Método de planificación temporal

GANTT

Componentes de un Diagrama de Gantt en la Gestión de Proyectos de Software



Representación gráfica

Diagrama de GANTT

2025

Tareas	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5
I.1.1 Identificar necesidades y beneficios Reunión con clientes Identificar necesidades y restricciones del proyecto Establecer enunciado del producto <i>Hito: Definición de enunciado del producto</i>					
I.1.2 Definir salida/control/entrada (SCE) deseados Ámbito funciones del teclado Ámbito funciones entrada de voz Ámbito modos de interacción Ámbito diagnóstico de documento Ámbito de otras funciones PP Documento SCE FTR: Revisar SCE con cliente Revisar SCE según se requiera <i>Hito: Definición de SCE</i>					
I.1.3 Definir función/comportamiento Definir funciones teclado Definir funciones entrada de voz Describir modos de interacción Describir corrector vocabulario/gramática Describir otras funciones PP FTR: Revisar definición SCE con cliente Revisar según se requiera <i>Hito: Definición SCE completa</i>					
I.1.4 Aislar elementos de software <i>Hito: Definición elementos de software</i>					
I.1.5 Investigar disponibilidad software existente Investigar componentes edición de texto Investigar componentes entrada de voz Investigar componentes manejo de archivos Investigar componentes corrector vocabulario/gramática <i>Hito: Identificación componentes reutilizables</i>					
I.1.6 Definir factibilidad técnica Evaluar entrada de voz Evaluar corrector gramatical <i>Hito: Valoración factibilidad técnica</i>					
I.1.7 Hacer estimaciones rápidas de tamaño					
I.1.8 Crear una definición de ámbito Revisar documento de ámbito con cliente Revisar documento según se requiera <i>Hito: Documento de ámbito completo</i>					

PERT y CPM

PERT, o Técnica de Evaluación y Revisión de Programas, es una herramienta de gestión de proyectos utilizada para analizar las estimaciones de tareas en un cronograma y evaluar las opciones de la ruta crítica, especialmente cuando las duraciones de las tareas son inciertas.

14

Por otro lado, **CPM**, o Método de la Ruta Crítica, es un método de gestión de proyectos que identifica la secuencia de actividades que determinan el tiempo mínimo de finalización de un proyecto, centrándose en tareas bien definidas con duraciones conocidas. Ambos métodos ayudan a desglosar proyectos complejos en tareas manejables, visualizar cronogramas, identificar dependencias y, en última instancia, apuntan a la finalización del proyecto a tiempo.

Método de planificación temporal

PERT (Program Evaluation & Review Technique):

- ❖ Creado para proyectos del programa de defensa del gobierno norteamericano entre 1958 y 1959.
- ❖ Se utiliza para controlar la ejecución de proyectos con gran número de actividades que implican investigación, desarrollo y pruebas.
- ❖ Red de tareas con Fechas tempranas, tardías, Camino crítico
- ❖ Probabilístico

Método de planificación temporal

CPM (Critical Path Method):

- ❖ Desarrollado para dos empresas americanas entre 1956 y 1958.
- ❖ Se utiliza en proyectos en los que hay poca incertidumbre en las estimaciones.
- ❖ Tiempo de inicio temprano y tardío
- ❖ Determinístico

Método de planificación temporal

❖ PERT y CPM

Actualmente se ha tomado lo mejor de ambos
vuelto uno solo, conocido como *Método del Camino*

1. *Establecer lista de tareas*

2. *Fijar dependencia entre las tareas*

3. *Construir la red*

4. *Numerar los nodos*

5. *Calcular la fecha de inicio*

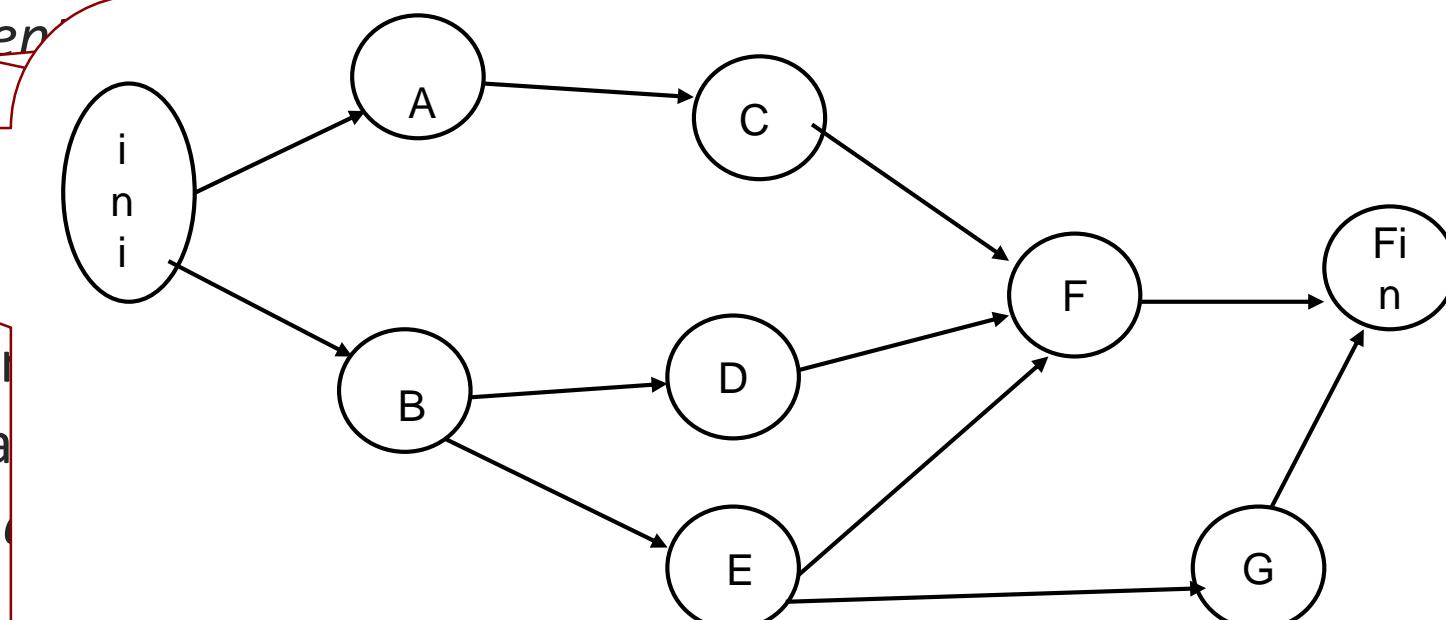
$Tei = \text{Fecha temprana}$

$Tai = \text{Fecha tardía}$

6. *Calcular el camino crítico*

$\Rightarrow Tei = Tai$

Tarea	Precedida por	Duración
A	-	2
B	-	5
C	A	4
D	B	7



Método de planificación temporal

PERT - CPM

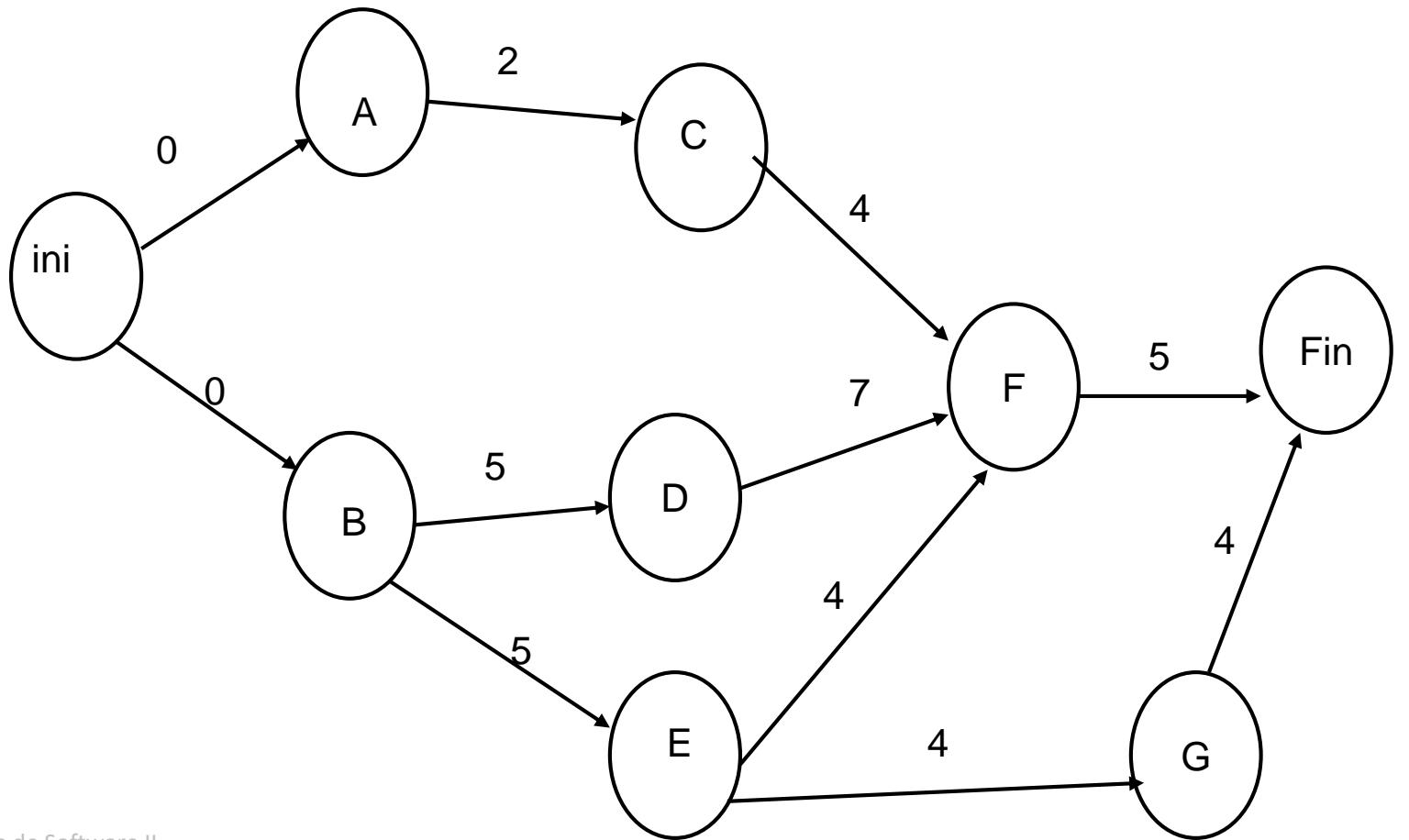
Ejemplo

1. *Establecer lista de tareas*
2. *Fijar dependencia entre tareas y duración*

Tarea	Precedida por	Duración
A	-	2
B	-	5
C	A	4
D	B	7
E	B	4
F	C-D-E	5
G	E	4

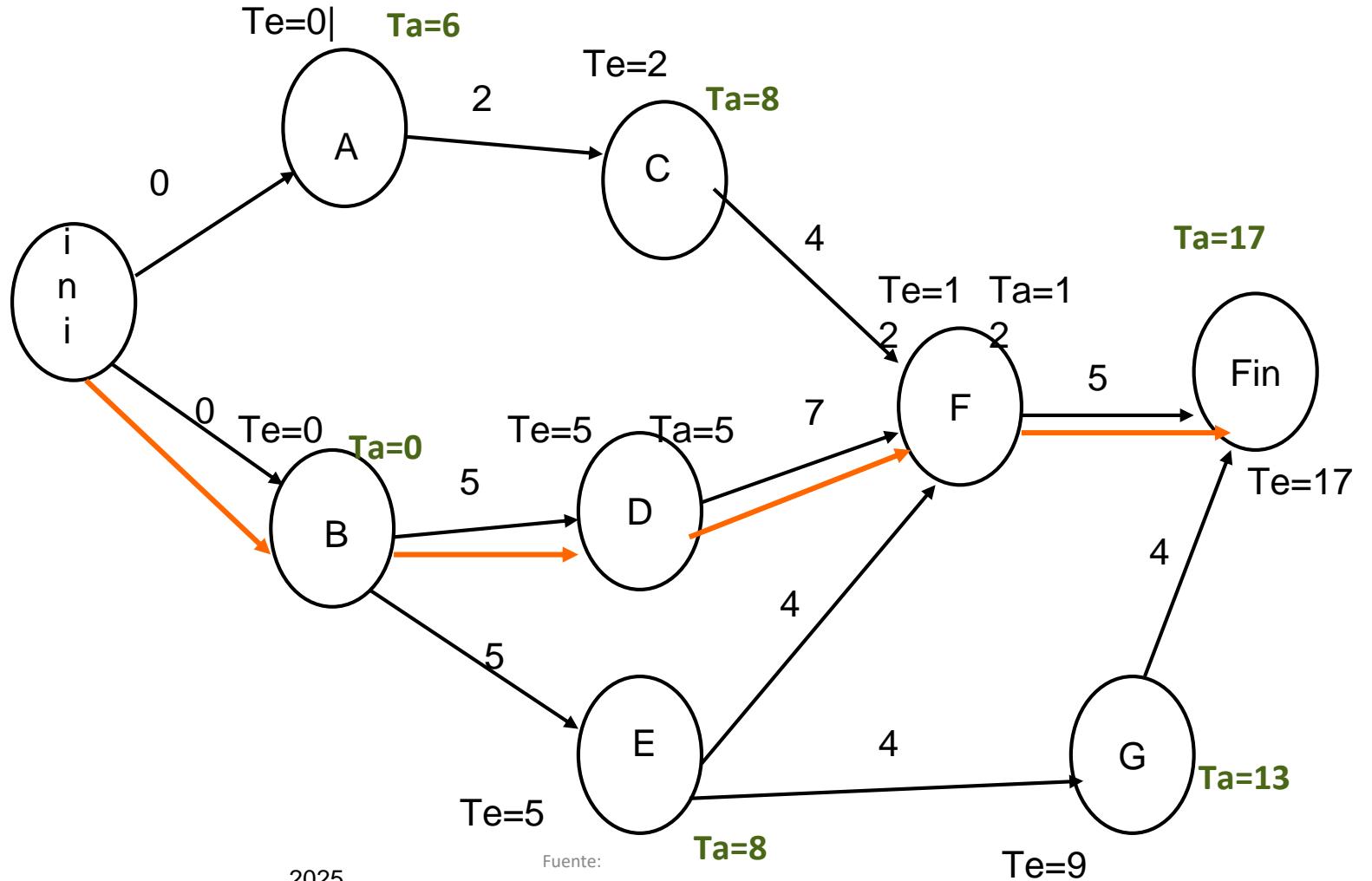
Método de planificación temporal

PERT - CPM



Método de planificación temporal

PERT - CPM



Método de planificación temporal PERT - CPM

❖ Fechas Tempranas

$$TeJ = Tel + tIJ$$

Donde

TeJ = fecha más temprana del nodo destino

Tel = fecha más temprana del nodo origen

tIJ = duración de la tarea desde el nodo I hasta el nodo J

Si hay más de un camino ... Max (TeJ1, TeJ2..)

Tarea	Precedida por	Duración
A	-	2
B	-	5
C	A	4
D	B	7

Método de planificación temporal

PERT - CPM

❖ Fechas Tardías

$$Tal = TaJ - tIJ$$

Donde

Tal = fecha más tardía del nodo origen

TaJ = fecha más tardía del nodo destino

tIJ = duración de la tarea desde el nodo I hasta el nodo J

Si hay más de un camino ... Min (TaJ1, TaJ2..)

Método de planificación temporal

PERT - CPM

❖ Margen Total

$$Mt = TaJ - Tel - tIJ$$

Donde

TaJ = fecha tardía del nodo destino

Tel = fecha temprana del nodo origen

tIJ = duración de la tarea desde el nodo I hasta el nodo J

OBSERVAR que el Margen total también puede calcularse como

$$MtJ = TaJ - TeJ$$

Es decir como la diferencia entre la fecha más tardía y más temprana del mismo nodo



Método de planificación temporal

PERT - CPM

¿Qué ocurre cuando tengo un margen total de por ej. 6 días?

Significa que la tarea puede iniciarse con 6 días de retraso sin que ello afecte a la duración total del proyecto.

Método de planificación temporal

PERT - CPM

❖ ¿Qué ocurre cuando el margen total es 0?

Significa que no hay margen y que esa tarea hay que iniciarla y finalizarla en las fechas más tempranas.

Puntualmente estas tareas con margen cero serían críticas.

- ❖ El camino formado por una sucesión de tareas críticas recibe el nombre de camino crítico.
- ❖ El camino crítico puede obtenerse utilizando el cálculo del margen total.

Método de planificación temporal

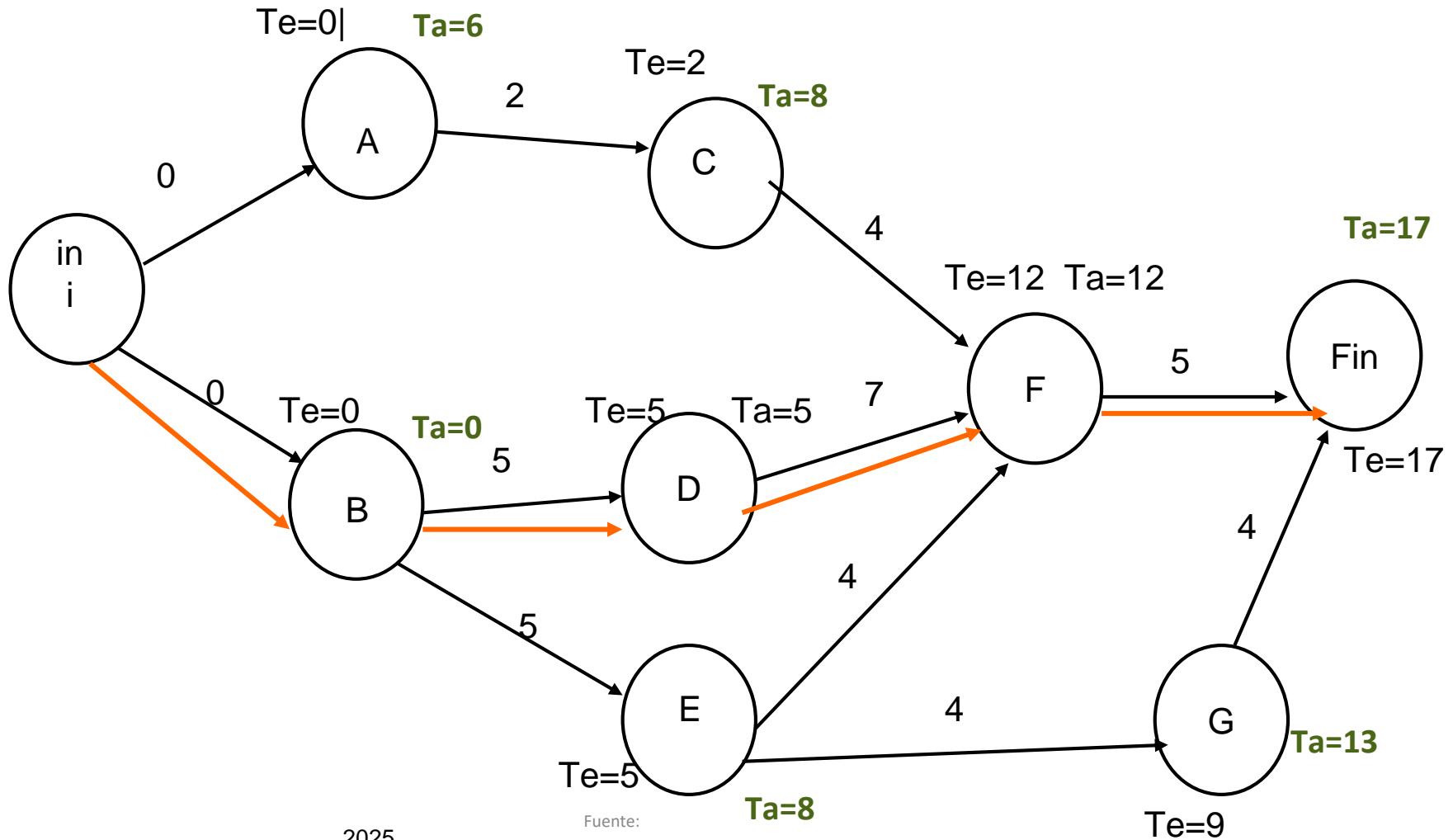
PERT - CPM

Ejemplo

Tarea	Precedida por	Duración
A	-	2
B	-	5
C	A	4
D	B	7
E	B	4
F	C-D-E	5
G	E	4

Método de planificación temporal

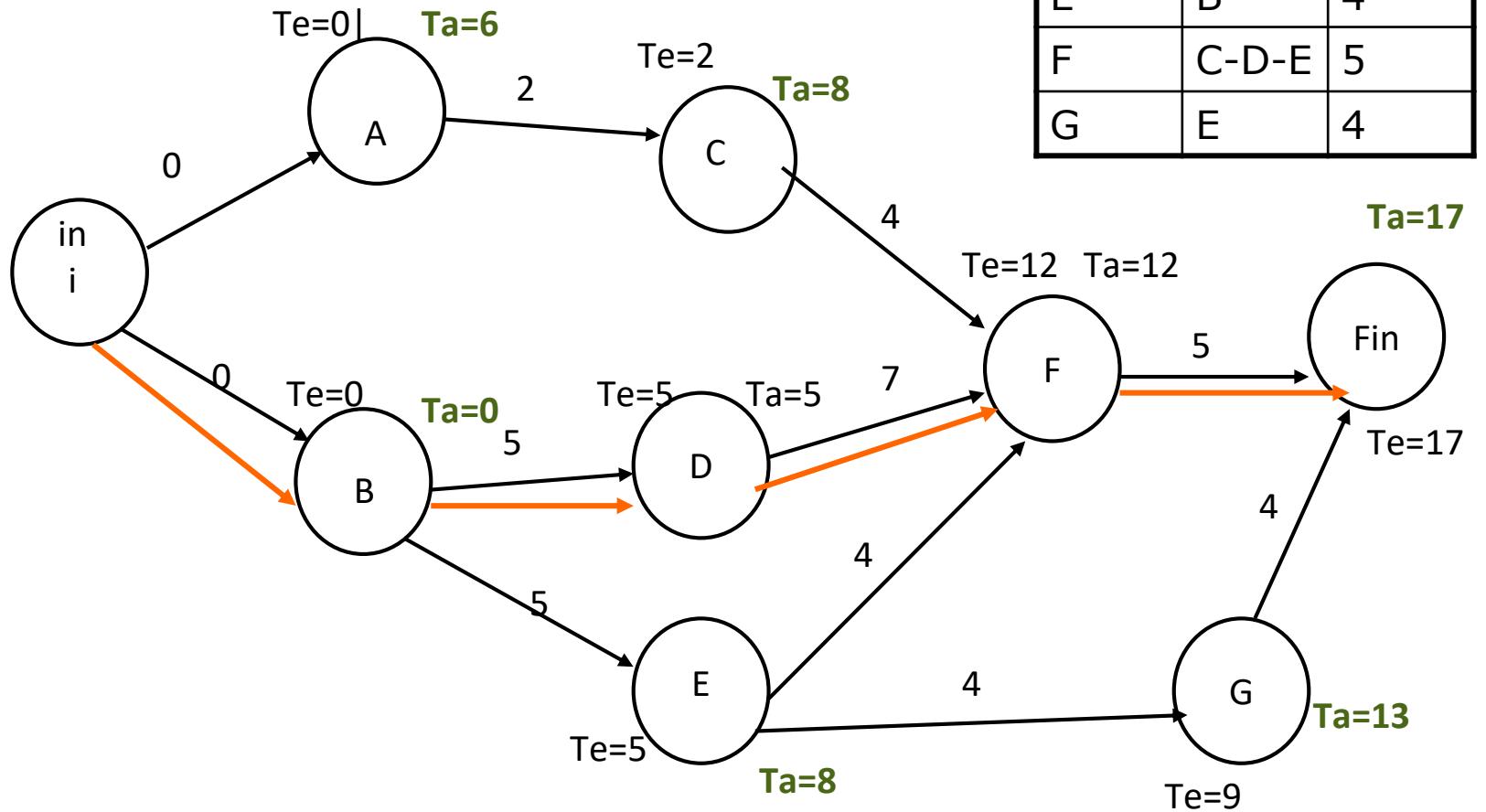
PERT - CPM



Método de planificación temporal PERT - CPM



Tarea	Prece d.	Dur.
A	-	2
B	-	5
C	A	4
D	B	7
E	B	4
F	C-D-E	5
G	E	4



Método de planificación temporal

PERT - CPM

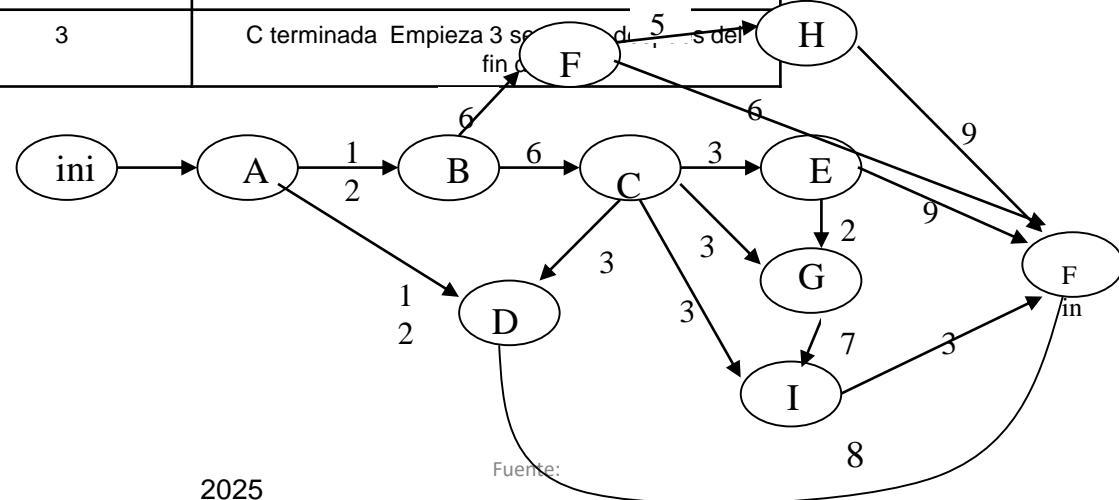


Tarea	Duración	Restricciones
A	12	
B	5	A terminada
C	3	Empieza 1 semana después de terminada B
D	8	A terminada C terminada
E	9	C terminada
F	6	Empieza 6 semanas después del comienzo de B
G	4	C terminada. Empieza 2 semanas después del comienzo de E
H	9	Empieza 1 semana antes del fin de F
I	3	C terminada Empieza 3 semanas después del fin de G

Método de planificación temporal PERT - CPM

Ejemplo

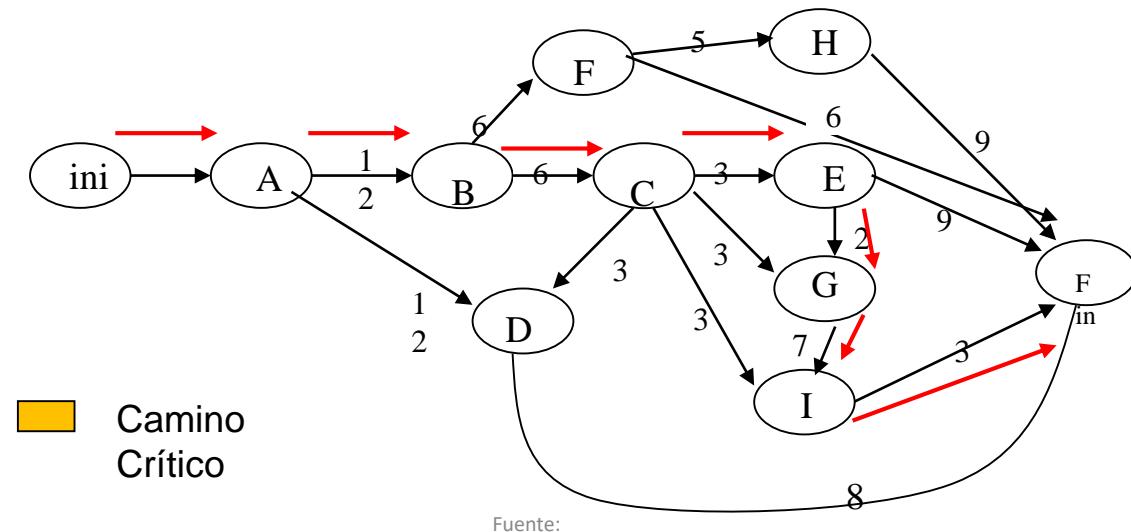
Tarea	Duración (semanas)	Restricciones
A	12	
B	5	A terminada
C	3	Empieza 1 semana después de terminada B
D	8	A terminada C terminada
E	9	C terminada
F	6	Empieza 6 semanas después del comienzo de B
G	4	C terminada. Empieza 2 semanas después del comienzo de E
H	9	Empieza 1 semana antes del fin de F
I	3	C terminada. Empieza 3 semanas después del fin de F



PERT - CPM

Tarea	Duración (semanas)	Restricciones	Te	Ta
A	12		0	0
B	5	A terminada	12	12
C	3	Empieza 1 semana después de terminada B	18	18
D	8	A terminada C terminada	21	25
E	9	C terminada	21	21
F	6	Empieza 6 semanas después del comienzo de B	18	19
G	4	C terminada. Empieza 2 semanas después del comienzo de E	23	23
H	9	Empieza 1 semana antes del fin de F	23	24
I	3	C terminada Empieza 3 semanas después del fin de G	30	30
Fin	0	-	33	33

Ejemplo



PERT-CPM



Datos que se obtienen
del PERT - CPM

 Camino crítico	El camino crítico determina las tareas esenciales para el proyecto.
 Fecha temprana inicio	Indica el tiempo disponible para cada actividad.
 Fecha tardía inicio	Muestra cuándo puede comenzar una tarea.
 Final más temprano	Indica el último momento para iniciar sin retrasos.
 Final más tardío	Define el momento más pronto en que puede concluir una tarea.
 Margen total	Establece el último momento para finalizar una tarea.
	Representa el tiempo total que se puede retrasar sin afectar el proyecto.

Planificación temporal

❖ ¿QUÉ HACER CUANDO UNA TAREA SE SALE DE LA AGENDA?

Revisar el impacto sobre la fecha de entrega

Reasignar recursos

La inclusión de más personas en el desarrollo no siempre genera aumento en la productividad

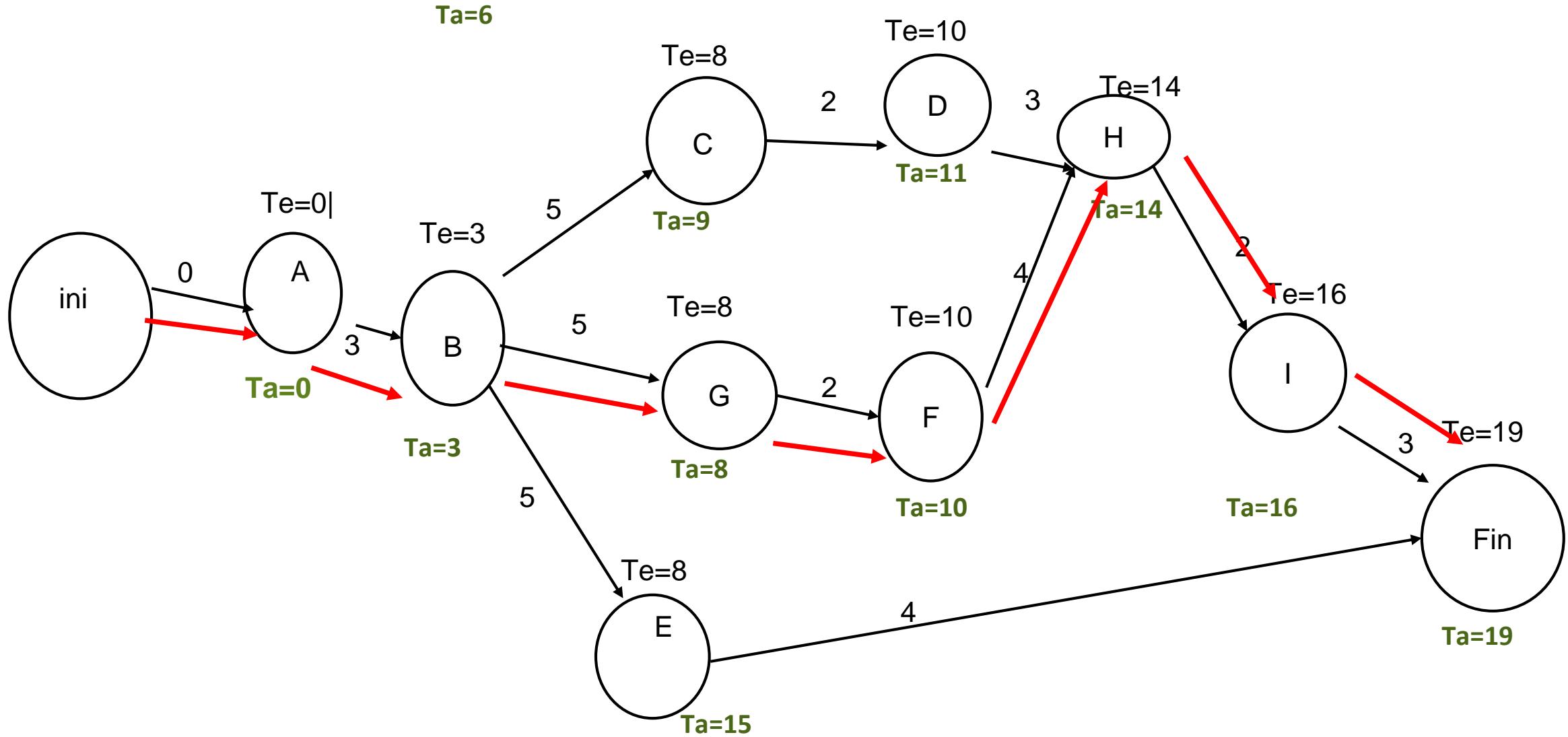
Reordenar tareas

Modificar entrega

Ejemplo

ID	DESCRIPCION	PREDEC	TIEMPO
A	Hacer los cimientos		3
B	Erigir la estructura	A	5
C	Poner las vigas del techo	B	2
D	Revestir el techo	C	3
E	Cableado eléctrico	B	4
F	Paredes exteriores	G	4
G	Colocar ventanas	B	2
H	Paredes interiores	D,F	2
I	Pintura exterior e interior	F,H	3

Método de planificación temporal PERT - CPM





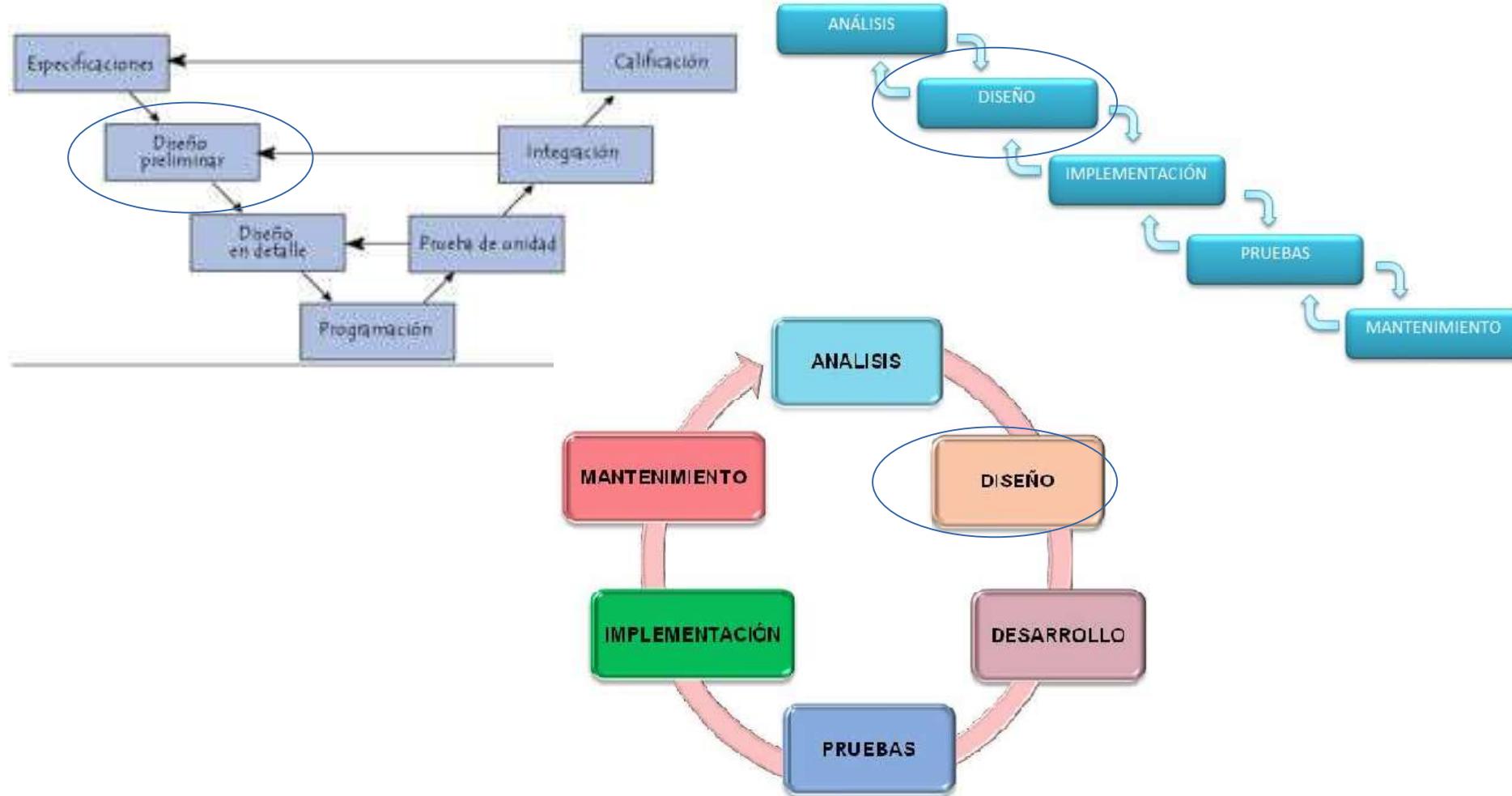
Diseño de Software - Conceptos

Ingeniería de software II - 2025

Etapa de Diseño en los modelos de desarrollo de Software



37



Diseño de Software



¿Qué es?



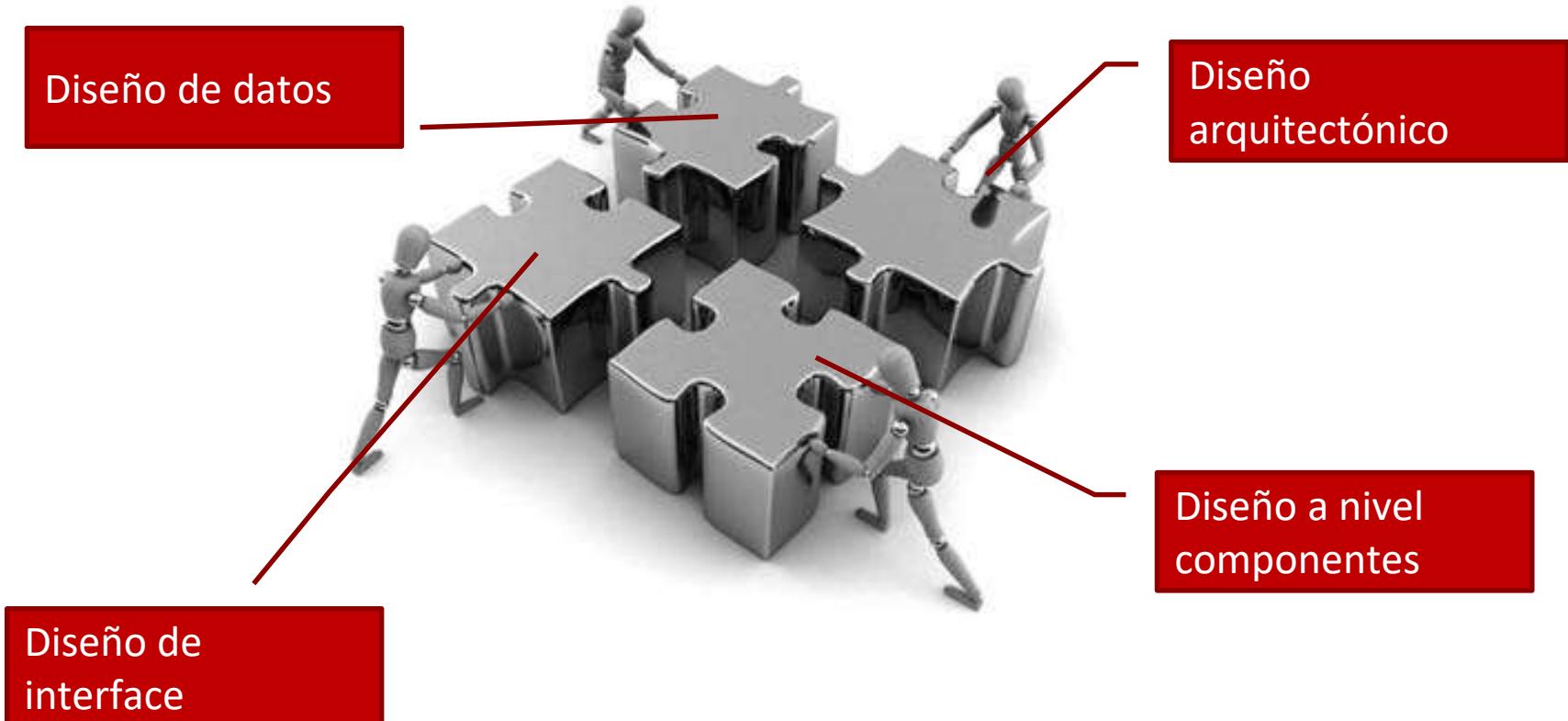
Áreas



¿Por qué es
importante?

38

Diseño de Software - Tipos



39

Diseño de Software - Tipos

Diseño de datos



40

- Transforma el modelo del dominio, obtenido del análisis, en estructuras de datos, objetos de datos, relaciones , etc.

Diseño de Software - Tipos

Diseño
arquitectónico

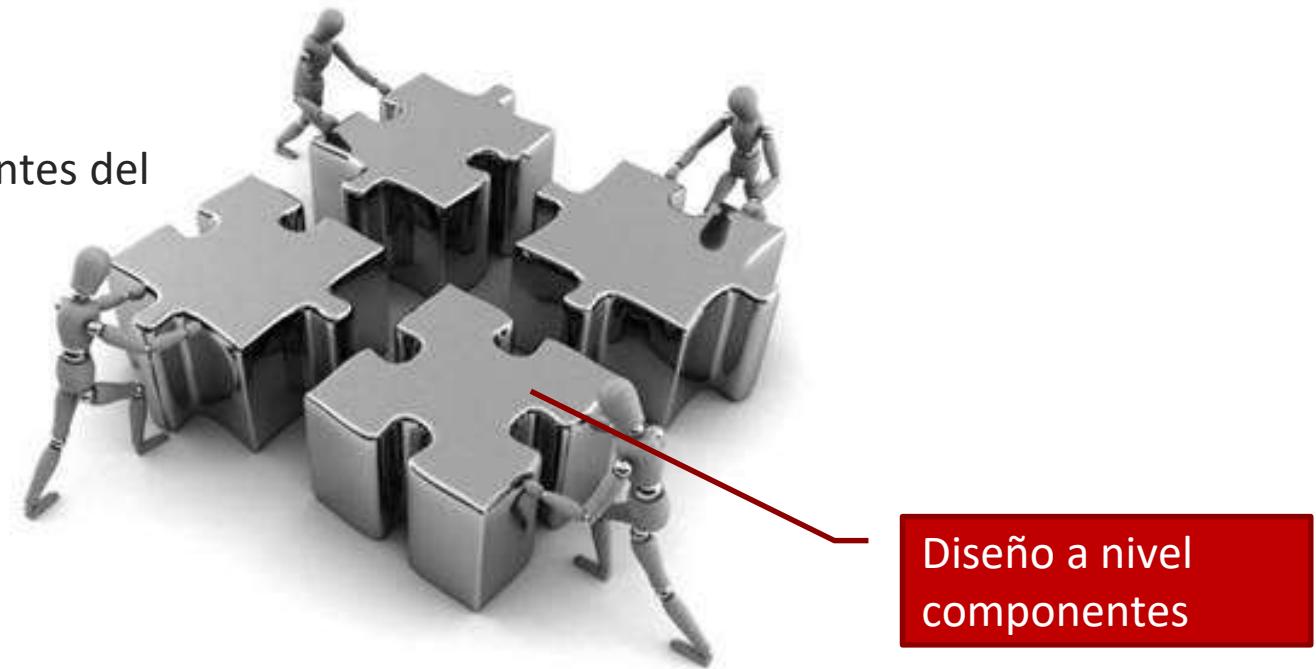


- Define la relación entre los elementos estructurales del software, los estilos arquitectónicos, patrones de diseño, etc.

41

Diseño de Software - Tipos

- Transforma los elementos estructurales de la arquitectura de software en una descripción procedural de los componentes del software.

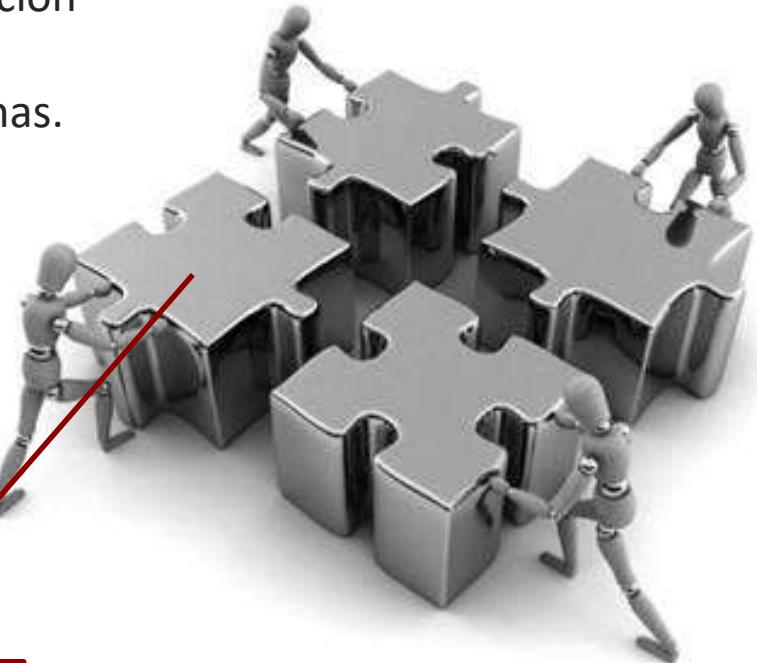


Diseño a nivel
componentes

42

Diseño de Software - Tipos

- Describe la forma de comunicación dentro del mismo sistema, con otros sistemas, y con las personas.



Diseño de
interface

43

Diseño de Software - Características para su evaluación



44

- ❖ Deberá *implementar* todos los requerimientos explícitos del modelo de requerimientos, e *incorporar* todos los requerimientos implícitos que desea el cliente.
- ❖ Deberá ser una *guía* legible y comprensible para aquellos que generan código y para aquellos que dan soporte al software.
- ❖ Deberá proporcionar una imagen/visión completa del software. (Completitud)

Criterios técnicos para un buen diseño



1. Deberá presentar una estructura arquitectónica que:

Se haya creado mediante patrones de diseño reconocibles, que esté formado por componentes con buen diseño y se implemente en forma evolutiva.

1. Deberá ser modular.
2. Deberá contener distintas representaciones.
3. Deberá conducir a estructuras de datos adecuadas y que procedan de patrones de datos reconocibles.
4. Deberá conducir a componentes que presenten características funcionales independientes.
5. Deberá conducir a interfaces que reduzcan la complejidad de las conexiones entre los módulos y con el entorno externo
6. Deberá derivarse mediante un método repetitivo y controlado por la información obtenida durante el análisis de los requisitos del software.
7. Deberá representarse por medio de una notación que comunique de manera eficaz su significado.

45

Evolución del diseño de software

Es un proceso continuo que abarca mas de seis décadas
Desde Programas modulares y refinamiento de estructuras de software, a
enfoques orientado a objetos, orientado a modelos o a pruebas.
Todos tienen características comunes:

1. un mecanismo para traducir el modelo de requerimientos en una representación de diseño
2. una notación para representar los componentes funcionales y sus interfaces
3. heurísticas para refinamiento
4. lineamientos para evaluación de calidad

46

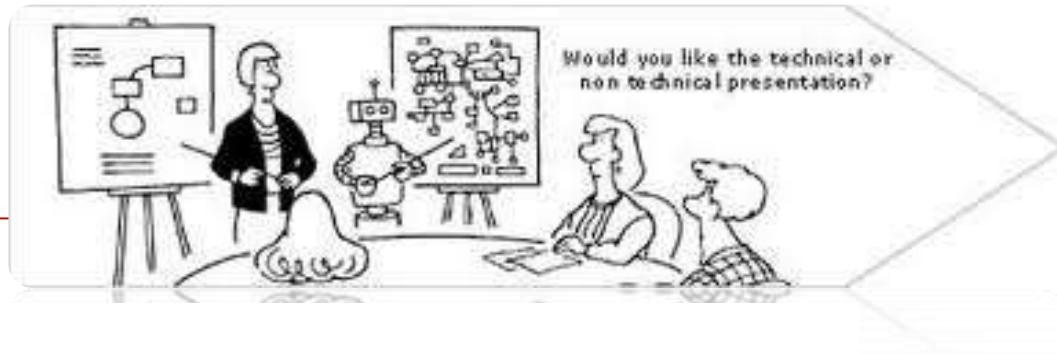
Sin importar el tipo de diseño, es necesario aplicar un conjunto de conceptos básicos.

Conceptos de Diseño - Importancia

- ❖ ¿Qué criterios se usan para dividir el software en sus componentes individuales?
- ❖ ¿Cómo se extraen los detalles de la función o la estructura de datos de la representación conceptual del software?
- ❖ ¿Cuáles son los criterios uniformes que definen la calidad técnica de un diseño de software?

47

Conceptos de Diseño



Conceptos Clave en Diseño de Software

48



Made with Napkin

Conceptos de Diseño



❖ Abstracción

Permite concentrarse en un problema a un nivel de *generalización* sin tener en cuenta los detalles de bajo nivel

Tipos :

Procedimental	<i>Secuencia “nombrada” de instrucciones que tienen una funcionalidad específica</i>
De datos	<i>Colección “nombrada” de datos que definen un objeto real</i>

49



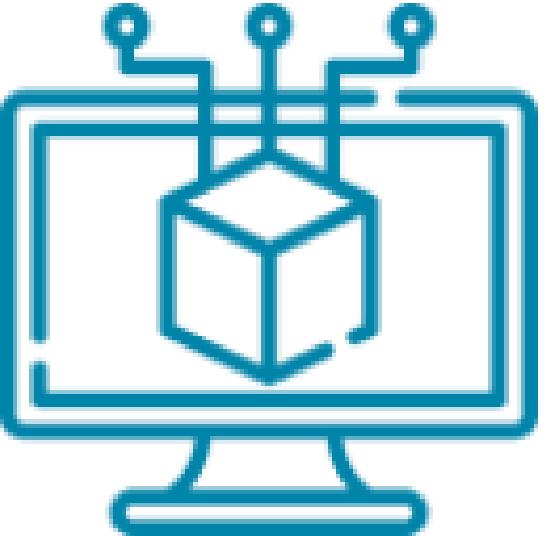
Conceptos de Diseño

❖ Arquitectura del software

Es la estructura general del software y las formas en que la estructura proporciona una integridad conceptual para un sistema.

Más adelante se estudiarán las arquitecturas con más detalle

50



Conceptos de Diseño

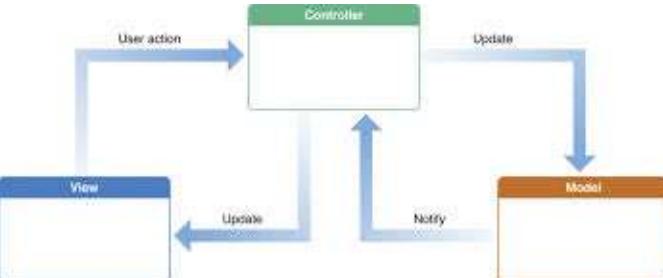
❖ Patrones

Describen una estructura de diseño que resuelve un problema particular dentro de un contexto específico.

Deben proporcionar una descripción que permita determinar si

- ✓ es aplicable al trabajo
- ✓ se puede reutilizar
- ✓ puede servir como guía para desarrollar un patrón similar pero diferente en cuanto a la funcionalidad o estructura.

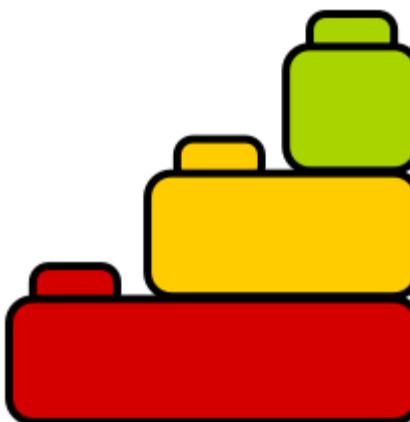
51



❖ Modularidad

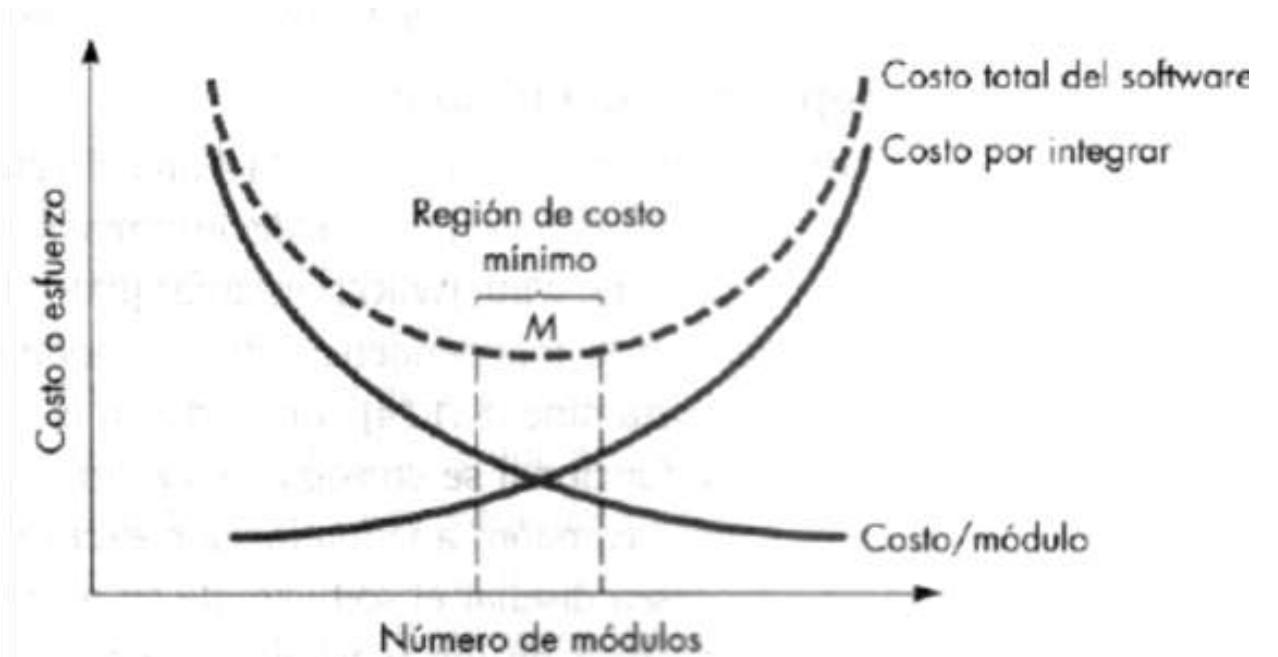
El software se divide en componentes nombrados y abordados por separado, llamados frecuentemente módulos, que se integran para satisfacer los requisitos del problema.

52



Conceptos de Diseño

- ❖ ¿Cuántos módulos tiene que tener un programa?



53

Conceptos de Diseño

❖ Ocultamiento de información

La información que está dentro un módulo es inaccesible a otros que no la necesiten.



54

Conceptos de Diseño

Independencia Funcional = *Modularidad + Abstracción + Ocultamiento de Información.*

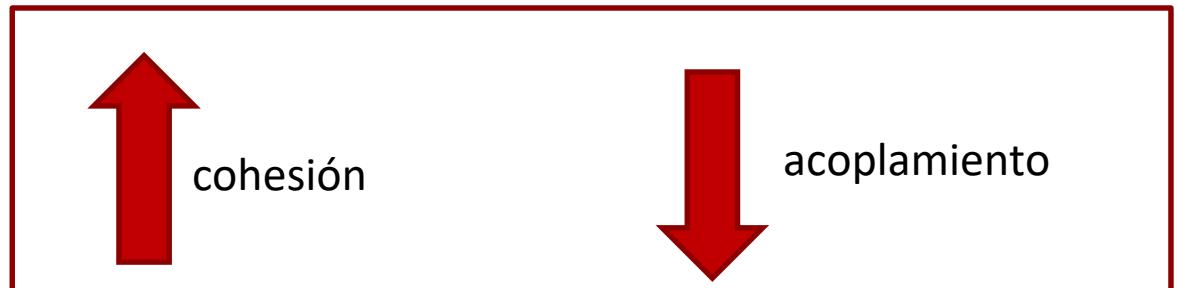


55



la cohesión y el acoplamiento entre los módulos

se busca...



Independencia Funcional

Cohesión (Coherente)

Medida de fuerza o relación funcional existente entre las sentencias o grupos de sentencias de un mismo módulo.

56



Alta cohesión



Baja cohesión

Independencia Funcional - Tipos de Cohesión

57

Cuando las sentencias se deben ejecutar en el mismo intervalo de tiempo

Más bajo

Cuando las sentencias tiene que ejecutarse en un orden específico

Cuando las sentencias de un módulo están relacionadas en el desarrollo de una única función

Coincidental

Lógica

Temporal

Procedimental

Comunicacional

Funcional

Cuando las sentencias llevan a cabo un conjunto de tareas que no están relacionadas o tienen

Cuando las sentencias se relacionan lógicamente

Cuando los elementos de procesamiento se centran en los datos de entrada y salida

Independencia Funcional

Acoplamiento

Es la medida de interconexión entre los módulos

Punto donde se realiza la entrada o referencia y los datos que pasan a través de la interfaz.

58



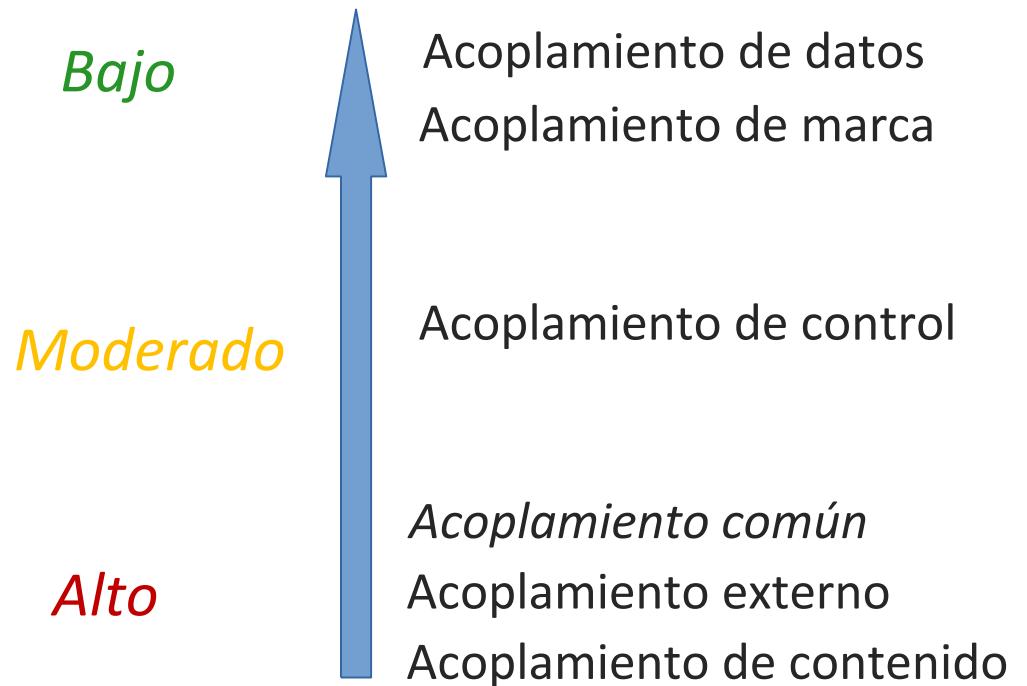
Bajo Acoplamiento



Alto Acomplamiento

Independencia Funcional

Niveles de Acoplamiento



59

Independencia Funcional

60



Conceptos de Diseño



❖ Refinamiento

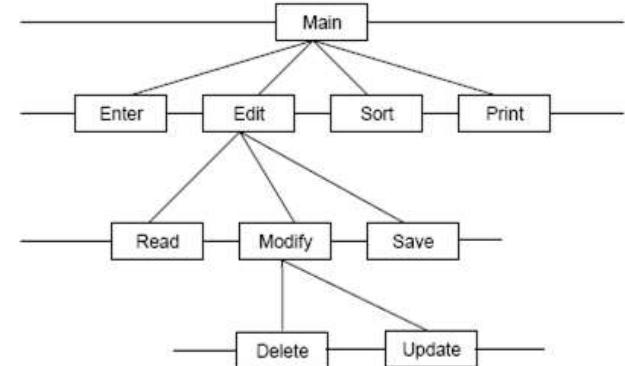
Se refina de manera sucesiva.

La abstracción y el refinamiento son conceptos *complementarios*.

La abstracción permite especificar procedimientos y datos sin considerar detalles de grado menor.

El refinamiento ayuda a revelar los detalles de grado menor mientras se realiza el diseño.

61



Conceptos de Diseño

❖ Refabricación o rediseño (Refactoring)

Técnica de reorganización que simplifica el diseño de un componente sin cambiar su función o comportamiento.



62

Principios del modelado de diseño

1. El diseño debe poder rastrearse hasta el modelo de requerimientos
2. Considerar siempre la arquitectura del sistema que se va a crear
3. El diseño de los datos es tan importante como el diseño de funciones
4. Las interfaces deben diseñarse con cuidado
5. El diseño de interfaz debe ajustarse a las necesidades del usuario, haciendo énfasis en la facilidad de uso
6. El diseño a nivel de componentes debe ser funcionalmente independiente
7. Los componentes deben tener un bajo acoplamiento
8. Las representaciones de diseño debe entenderse fácilmente
9. El diseño debe desarrollarse de manera iterativa
10. La creación de un modelo de diseño no impide metodología ágil.

63



Ingeniería de software II

Diseño de la interfaz del usuario - 2025

¿Qué es el diseño de una interface de usuario (UI- User Interface)?

»Se basa en diseño de computadoras, aplicaciones, máquinas, dispositivos de comunicación móvil, aplicaciones de software y sitios web enfocado en la experiencia de usuario y la interacción.

2

»*Normalmente es una actividad multidisciplinaria que involucra a varias ramas del diseño y el conocimiento como el diseño gráfico, industrial, web, de software y la ergonomía; y está implicado en un amplio rango de proyectos, desde sistemas para computadoras, vehículos hasta aviones comerciales*

¿Cuál es el objetivo de la UI?

El objetivo de la Interfaz de Usuario es **mantener la interacción con los destinatarios** de una forma más atractiva, centrando el diseño en ellos.

3

El diseño gráfico y diseño industrial basan sus conocimientos para que los **usuarios** aprendan **lo más rápido posible el funcionamiento** del software.

Las herramientas principales que utilizan son recursos como la gráfica, los pictogramas, lo estético y la simbología, sin afectar el funcionamiento técnico eficiente.

¿Qué rol tienen los ingenieros de software?

Diseño de la interfaz del usuario

Conceptos iniciales

- » Es la categoría de diseño que crea un medio de comunicación entre el hombre y la máquina.
- » Con **un conjunto de principios** se crea un formato de pantalla.
- » Es **necesario estudiar las preferencias de las personas** para producir tecnología que se adapte a los seres humanos.



Pero en la actualidad tendemos a estudiar sólo a la tecnología. El resultado es que se exige a las personas que se adapten a la tecnología.

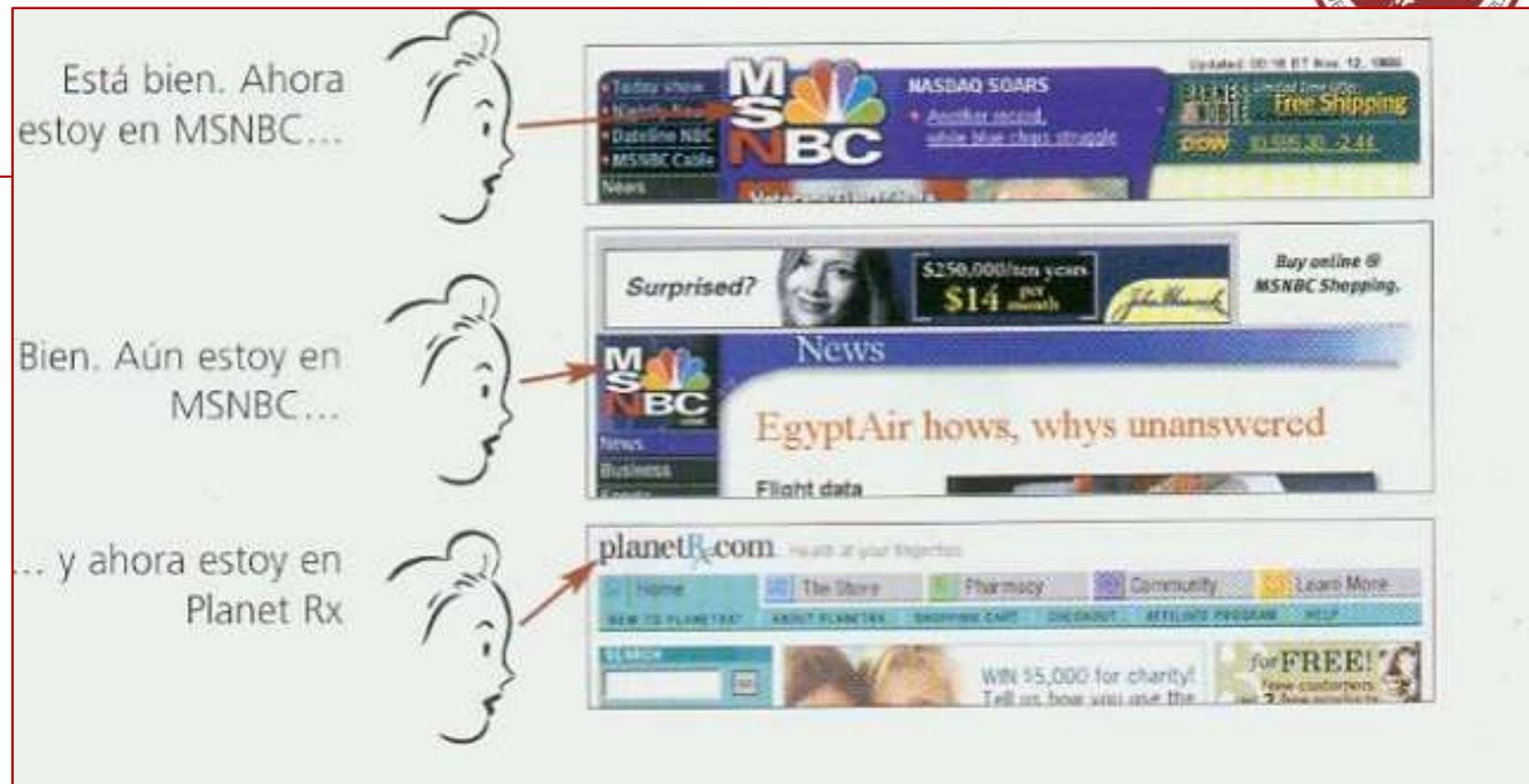
Es el momento de que la tecnología se adapte a las personas.

Diseño de la interfaz del usuario

Conceptos iniciales

» Una interfaz difícil de utilizar provoca que los usuarios cometan errores o incluso que se rehúsen a utilizar el sistema.

» Personas diferentes pueden tener estilos diferentes de percepción, comprensión y trabajo. **Diversidad**



» La interfaz debe contribuir a que el usuario consiga un **rápido acceso al contenido** de sistemas complejos, **sin pérdida de la comprensión** mientras se desplaza a través de la información.

Fuente:

Diseño de la interfaz del usuario. Conceptos iniciales

Conceptos de Diseño de Interfaz de Usuario



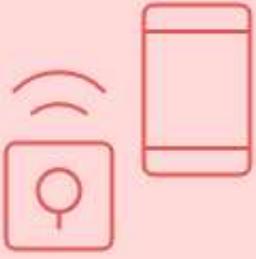
Tecnologías

Tecnologías que deben adaptarse al usuario. Estas incluyen hiperenlace, sonido, presentaciones tridimensionales, video, realidad virtual, etc.



Configuraciones de Hardware

Configuraciones de hardware como teclado, mouse, dispositivos de presentación gráfica, gafas de realidad virtual, reconocimiento de voz, etc.

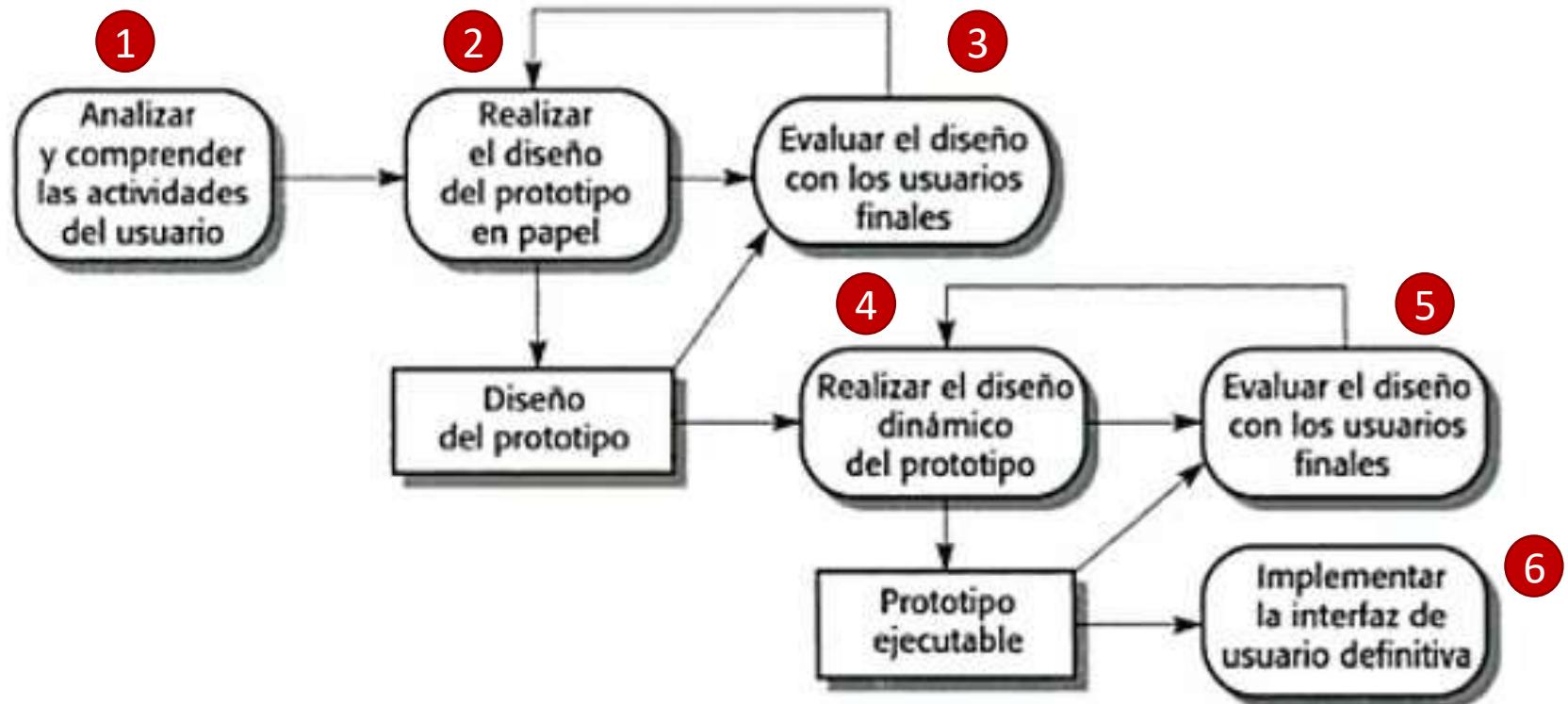


Dispositivos

Variedad de dispositivos: PC, equipos específicos, teléfonos celulares, televisores, etc.

Made with ➜ Napkin

6 principios para el diseño de la interfaz del usuario



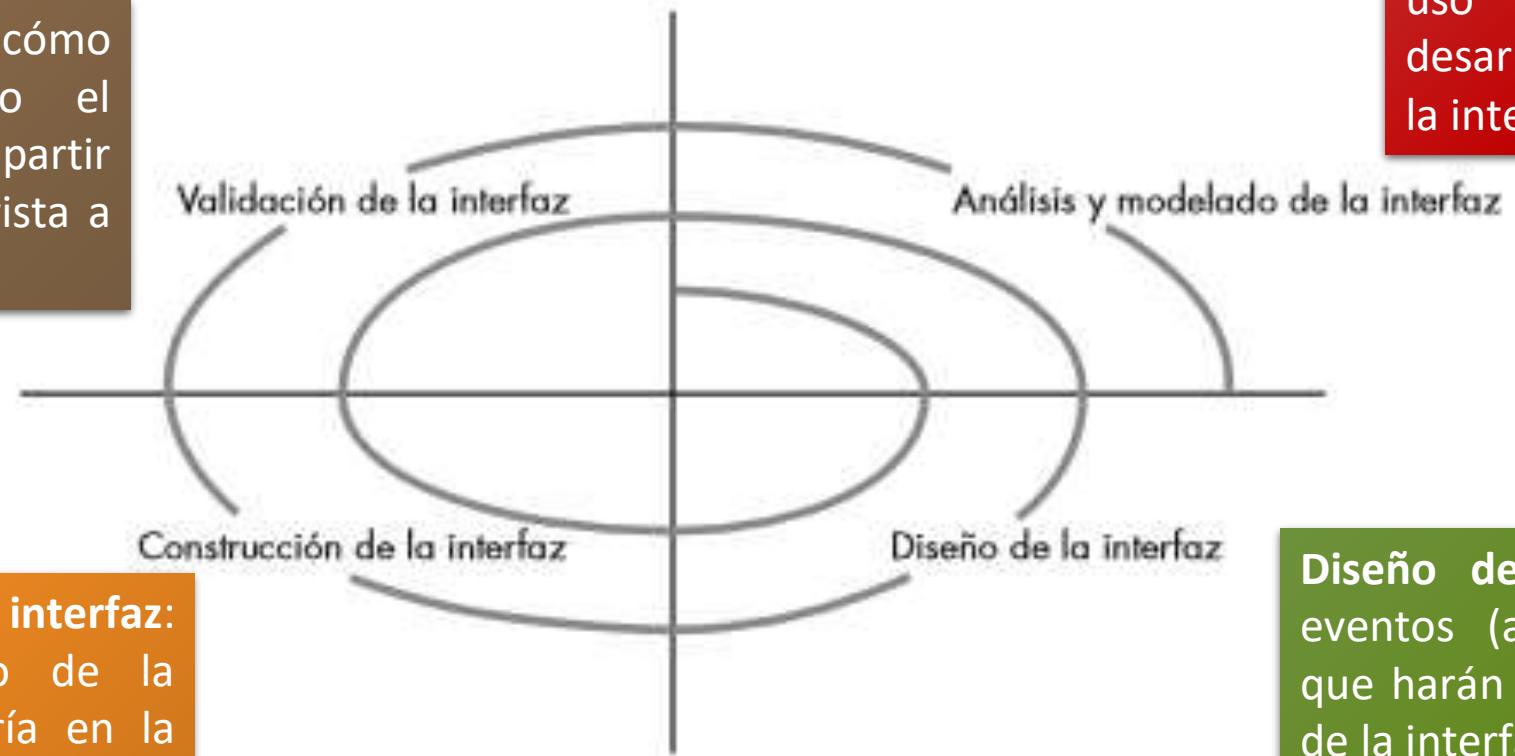
7

Diseño de la interfaz del usuario

Proceso

El proceso de análisis y diseño de interfaces de usuario es iterativo

Validación: Indicar cómo interpreta el usuario el estado del sistema a partir de la información provista a través de la interfaz.



Construcción de la interfaz: Ilustrar cada estado de la interfaz como lo vería en la realidad el usuario final.

Análisis y modelado: Definir objetos y acciones de la interfaz (operaciones) con el uso de la información desarrollada en el análisis de la interfaz

Diseño de la interfaz: Definir eventos (acciones del usuario) que harán que cambie el estado de la interfaz de usuario. Hay que modelar este comportamiento.

Diseño de experiencias de usuario (Ux)

El diseño de experiencias de usuario (Ux) es un conjunto de métodos aplicados al proceso de diseño que buscan satisfacer las necesidades del cliente y proporciona una buena experiencia a los usuarios destinatarios. (Allanwood & Beare 2015)

LO QUE LOS DISEÑADORES CREAN...



LO QUE LOS USUARIOS VEN...



Quiero comprar un billete.

¿Cómo comprobar las millas que me corresponden por mis frecuentes viajes?

The Tactical Traveler

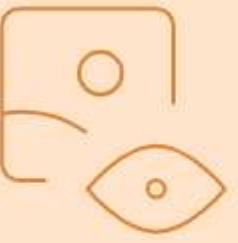
Tipos de diseño Ux

Tipos de diseño de Ux



Diseño de interacción

Se centra en la interacción entre el usuario y el producto, con el objetivo de lograr la satisfacción del usuario.



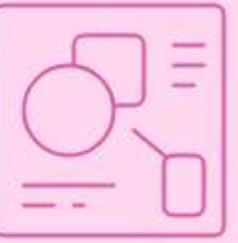
Diseño visual

Verifica la apariencia y la sensación de la navegación en la aplicación, incluyendo la eficiencia y el entretenimiento.



Investigación del usuario

Determina los deseos y necesidades de los clientes y usuarios.



Arquitectura de la información

Estructura y etiqueta el contenido para facilitar la recuperación de información por parte de los usuarios.

Made with ➡ Napkin

10

Fuente:

Etapas del diseño de experiencia de Usuario

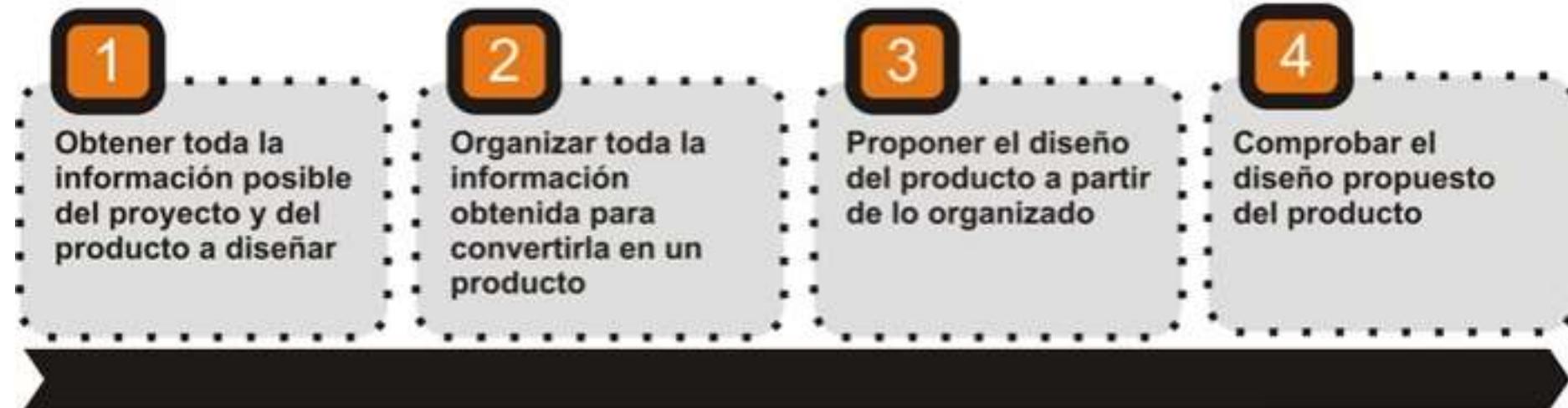
Proceso del Diseño de Experiencia de Usuario



11

Descripción de cada etapa de Ux

Etapas del Diseño de Experiencia de Usuario



INVESTIGACIÓN

ORGANIZACIÓN

DISEÑO

PRUEBA

12

(Ux) – Investigación de usuarios



Fuentes de Datos de Investigación de Usuarios

Encuestas
Recopilación de opiniones de los usuarios a través de cuestionarios estructurados.

Información de Ventas
Análisis de datos de ventas para comprender las preferencias de los usuarios.

Información de Mercadotecnia
Aprovechamiento de datos de marketing para obtener información sobre el comportamiento del usuario.

Charlas de Apoyo al Usuario
Extracción de necesidades del usuario a partir de interacciones de soporte.

13

Made with Napkin

Diseño de Interfaces y (Ux) – Información a relevar del usuario para crear el Perfil de usuario

- » Franja de edad
- » Etnia
- » Género
- » Experiencia
- » Nivel de ingresos
- » Idioma
- » Nivel de Estudios
- » Localización
- » Ocupación o profesión
- » Religión



14

Diseño de Interfaces y (Ux) – Relevamiento de la tarea: Contexto y ambiente de trabajo

- » Estudio de las partes implicadas en el sistema
- » Revisión de las competencias del producto
- » Recorridos del usuario dentro del sistema físico o virtual
- » ¿Qué trabajo realizará el usuario en circunstancias específicas?
- » ¿Qué tareas y subtareas se efectuarán cuando el usuario haga su trabajo?
- » ¿Qué dominio de problema específico manipulará el usuario al realizar su labor?
- » ¿Cuál es la secuencia de las tareas (el flujo del trabajo)? • ¿Cuál es la jerarquía de las tareas?

15

Esta etapa se realiza en paralelo a la generación de la especificación de requerimientos que se esté utilizando

Iceberg de UX

LA interface de usuario es la punta del Iceberg, el diseño más complejo está por debajo



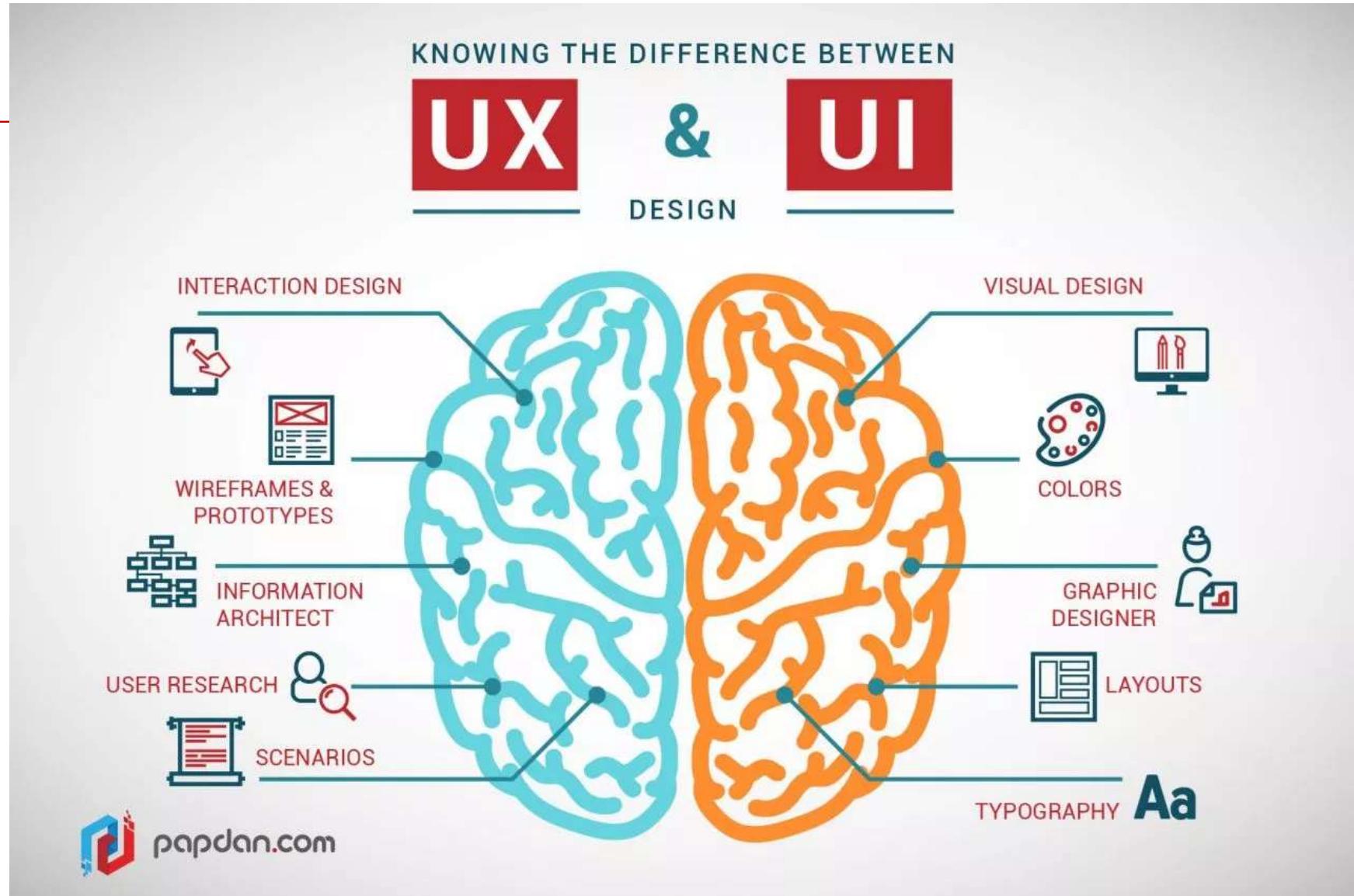
Diferencia entre Interfaz de usuario (UI) y Experiencia de usuario (UX)



17

Ambos conceptos colaboran en el diseño

18





Aspectos de diseño de interfaz

Reglas básicas del Diseño (reglas doradas del diseño)



Reglas básicas (Theo Mandel, 1997)

Interfaz Consistente

Mantener un diseño y funcionalidad uniformes en toda la interfaz



Control del Usuario

Permitir a los usuarios dirigir sus interacciones y experiencias



Carga de Memoria Reducida

Minimizar la necesidad de que los usuarios recuerden información

Made with Napkin

Reglas básicas del Diseño

Control al usuario

El usuario busca un sistema que reaccione a sus necesidades y lo ayude a hacer sus tareas.

- » Definir modos de interacción de forma que el usuario no realice acciones innecesarias
- » Proporcionar una interacción flexible
- » Incluir las opciones de interrumpir y deshacer
- » Depurar la interacción a medida que aumenta la destreza del usuario.
- » Ocultar al usuario ocasional los elementos técnicos internos
- » Diseñar interacción directa con los objetos que aparecen en pantalla

21

Reglas básicas del Diseño

Carga de memoria del usuario reducida

- » Reducir la demanda a corto plazo
- » Definir valores por defecto que tengan significado
- » Definir accesos directos intuitivos
- » El formato visual de la interfaz debe basarse en una metáfora de la realidad
- » Desglosar la información de manera progresiva

22

Reglas básicas del Diseño

Interfaz consistente

- » Permitir que el usuario incluya la tarea actual en un contexto que tenga algún significado
- » El usuario debe tener la capacidad de determinar de donde viene y hacia donde puede ir
- » Mantener consistencia en toda la familia de aplicaciones
- » Utilizar las mismas reglas de diseños para las mismas interacciones
- » Mantener modelos que son prácticos para el usuario, a menos que sea imprescindible cambiarlos

23

Reglas básicas del diseño

Factores Humanos (la sumamos a las 3 de Theo Mandel)

- » Percepción visual/auditiva/táctil
- » Memoria humana
- » Razonamiento
- » Capacitación
- » Comportamiento/Habilidad personales
- » Diversidad de usuarios

Usuarios casuales: Necesitan interfaces que los guíen.

Usuarios experimentados: Requieren interfaces ágiles.



Usabilidad - Concepto

-
- » La usabilidad **no proviene** de la estética, de mecanismos de interacción avanzados o de interfaces inteligentes. En vez de eso, se obtiene cuando la arquitectura de la interfaz se ajusta a las necesidades de las personas que la emplearán.
 - » Es ilusorio llegar a una definición formal de usabilidad. Es parte de lo semántico del software y las necesidades de las personas
 - » Donahue la define: "*La usabilidad es una medida de cuán bien un sistema de cómputo [...] facilita el aprendizaje, ayuda a quienes lo emplean a recordar lo aprendido, reduce la probabilidad de cometer errores, les permite ser eficientes y los deja satisfechos con el sistema.*"

25

Usabilidad - ¿Cuándo existe? (1)

La forma de determinar si existe “usabilidad” en un sistema que se construye es evaluarla o probarla. Los usuarios interactúan con el sistema y deben responder las preguntas siguientes:

- » ¿El sistema es utilizable sin ayuda o enseñanza continua?
- » ¿Las reglas de interacción ayudan a un usuario preparado a trabajar con eficiencia?
- » ¿Los mecanismos de interacción se hacen más flexibles a medida que los usuarios conocen más?
- » ¿Se ha adaptado el sistema al ambiente físico y social en el que se usará?
- » ¿El usuario está al tanto del estado del sistema? ¿Sabe en todo momento dónde está?
- » ¿La interfaz está estructurada de manera lógica y consistente?

26

Usabilidad - ¿Cuando existe? (2)

- » ¿Los mecanismos, iconos y procedimientos de interacción son consistentes en toda la interfaz?
- » ¿La interacción prevé errores y ayuda al usuario a corregirlos?
- » ¿La interfaz es tolerante a los errores que se cometan?
- » ¿Es sencilla la interacción?

27

Si cada una de estas preguntas obtiene un “SI” como respuesta, es probable que se haya logrado la usabilidad

Principios de Nielsen

- » Existen ciertos principios de diseño que enuncian el diálogo correcto que debe proveer una interfaz de usuario.
- » Estos principios fueron desarrollados por Jacob Nielsen y son utilizados para el diseño de interfaces y, como métricas de evaluación de interfaces ya desarrolladas.

28

Aunque estos principios fueron pensados inicialmente para interfaces textuales, sirven de base para el diseño preliminar de cualquier otro tipo de interfaz.

Los 10 principios de Usabilidad - JACOB NIELSEN



Característica	Descripción		
Diálogo simple y natural	Comunicación amigable e intuitiva	Salidas evidentes	Indicar claramente cómo salir
Lenguaje del usuario	Hablar el lenguaje del usuario	Mensajes de error	Comunicar errores de manera efectiva
Minimizar el uso de la memoria del usuario	Reducir la carga cognitiva en los usuarios	Prevención de errores	Diseñar para minimizar errores potenciales
Consistencia	Mantener uniformidad en la interfaz	Atajos	Ofrecer formas eficientes de realizar tareas
Feedback	Proporcionar respuestas informativas a las acciones	Ayudas	Proporcionar asistencia cuando sea necesario <small>Made with Napkin</small>

29

Principios de Nielsen

1.- Diálogo simple y natural: *Forma en que la interacción con el usuario debe llevarse a cabo.*

- » Realizar una escritura correcta, sin errores de tipeo
- » No mezclar información importante con la irrelevante
- » Distribución adecuada de la información
- » Prompts lógicamente bien diseñados
- » Evitar el uso excesivo de mayúsculas y de abreviaturas
- » Unificar el empleo de las funciones predefinidas



0

Principios de Nielsen

2.- Lenguaje del usuario: Emplear en el sistema un lenguaje familiar para el usuario, usar el lenguaje del usuario.

- No utilizar palabras técnicas, ni extranjeras
- Evitar el truncamiento excesivo de palabras
- Diseñar correctamente las entradas de datos
- Emplear un grado adecuado de información (ni excesivo ni escaso)



Principios de Nielsen

3.- Minimizar el uso de la memoria del usuario: Evitar que el usuario esfuerce su memoria para interactuar con el sistema.

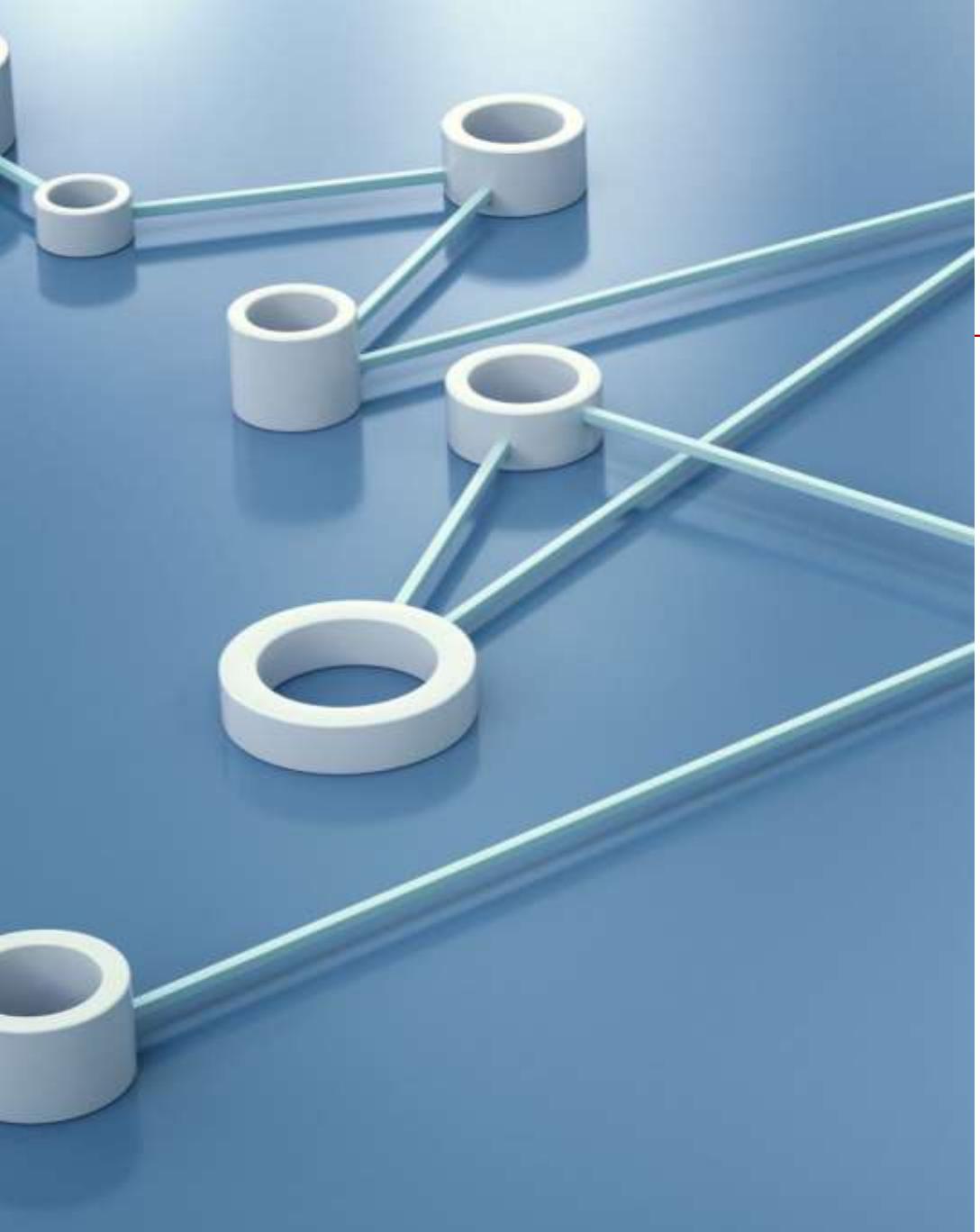
- Brindar información de contexto de la App
- Brindar información de la navegación y sesión actual del usuario
- Visualización de rangos de entrada admisibles, ejemplos, formatos



Principios de Nielsen

4.- Consistencia: Que no existan ambigüedades en el aspecto visual ni tecnológico en el diálogo o en el comportamiento del sistema.

- La consistencia es un punto clave para ofrecer confiabilidad y seguridad al sistema.
- Debe existir una consistencia terminológica y visual.



Principios de Nielsen

5.- Feedback: Es una respuesta gráfica o textual en la pantalla, frente a una acción del usuario. El sistema debe mantener al usuario informado de lo que está sucediendo.

- Brindar información de los estados de los procesos.
- Brindar información del estado del sistema y del usuario.
- Utilización de mensajes de aclaración, validaciones, confirmación y cierre.
- Realizar validaciones de los datos ingresados por el usuario.



Fuente:

Principios de Nielsen

6.- Salidas evidentes: Que el usuario tenga a su alcance de forma identifiable y accesible una opción de salida (exit) de la interfaz.

- Brindar salidas de cada pantalla.
- Salidas para cada contexto.
- Salidas para cada acción, tarea o transacción.
- Brindar salidas en cada estado.
- Visualización de opciones de cancelación, salida del sistema, suspender, deshacer, rehacer y modificación.



Principios de Nielsen

7.- Mensajes de error: *Información que brinda el sistema ante la presencia de un error. De qué forma se ayuda al sistema para que salga de la situación en la que se encuentra.*

- » Deben existir mensajes de error para ser usados en los momentos que corresponda.
- » Brindar información del error, explicar el error y dar alternativas a seguir.
- » Se deben categorizar los diferentes tipos de mensajes.
- » No deben existir mensajes de error intimidatorios.
- » Manejar adecuadamente la forma de aparición de los mensajes.



Principios de Nielsen



8.- Prevención de errores: *Evitar que el usuario llegue a una instancia de error.*

- » Brindar rangos de entradas posibles para que el usuario seleccione y no tenga que tipear (escribir).
- » Mostrar ejemplos, valores por defecto y formatos de entrada admisibles.
- » Brindar mecanismos de corrección automática en el ingreso de los datos.
- » Flexibilidad en las entradas de los usuarios.

Principios de Nielsen



9.- Atajos: *La interfaz debería proveer de alternativas de manejo para que resulte cómodo y amigable tanto para usuarios novatos como para usuarios experimentados.*

- » Brindar mecanismos alternativos para acelerar la interacción con el sistema
- » Brindar la posibilidad de reorganizar barras de herramientas, menús, de acuerdo con la necesidad del usuario
- » Brindar mecanismos de macros, definición de teclas de función



Principios de Nilsen

10.- Ayudas: *Componentes de asistencia para el usuario. Un mal diseño de las ayudas puede llegar a entorpecer y dificultar la usabilidad.*

- » Deben existir las ayudas en la interface y un manual.
 - » Se deben brindar diferentes tipos de ayuda : generales, contextuales, específicas, en línea.
 - » Las ayudas deben proveer diferentes formas de lectura.
 - » Se deben brindar diferentes mecanismos de asistencia como búsquedas, soporte en línea, e-mail del soporte técnico, acceso a las preguntas frecuentes.





Estilos de Interfaces

Tipos de Interfaces de Usuario

Tipos de Interfaces

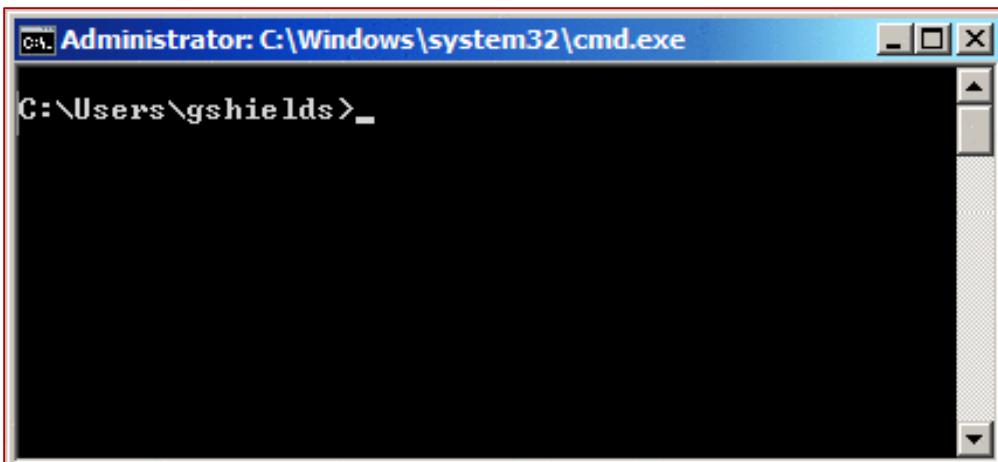


Tipos de Interfaces

Interfaz de comandos

- » Es la interfaz más elemental
 - Solo se interactúa con texto
- » Generalmente se interactúa desde una línea de comando de una consola de una aplicación en particular con el teclado
- » Características:
 - Poderoso y Flexible
 - Administración de errores pobre
 - Difícil de aprender

42



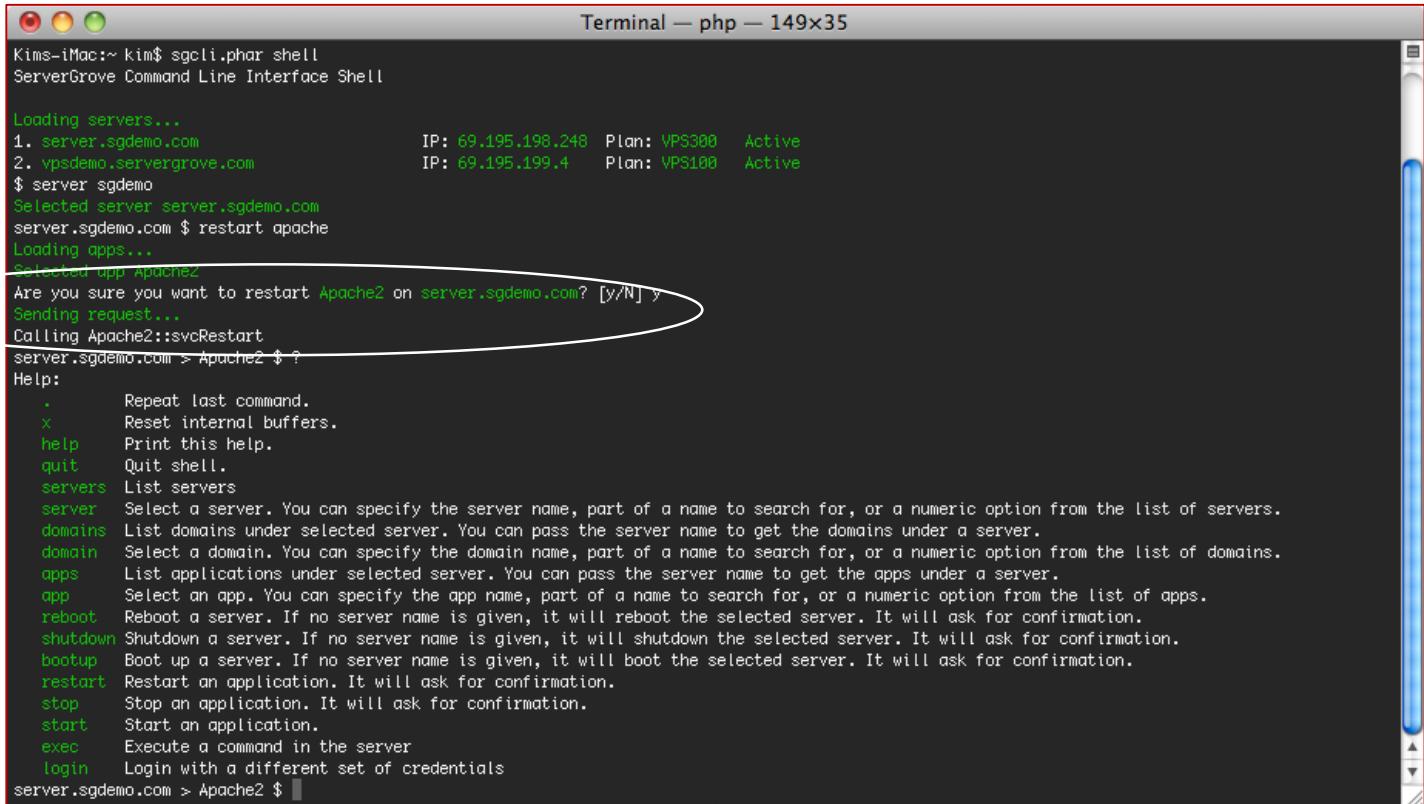
Consola del SO Windows

Tipos de Interfaces

Interfaz de comandos

»Comandos del tipo pregunta respuesta

43



```
Kims-iMac:~ kim$ sgcli.phar shell
ServerGrove Command Line Interface Shell

Loading servers...
1. server.sgdemo.com           IP: 69.195.198.248 Plan: VPS300 Active
2. vpsdemo.servergrove.com     IP: 69.195.199.4   Plan: VPS100 Active

$ server sgdemo
Selected server server.sgdemo.com
server.sgdemo.com $ restart apache
Loading apps...
Selected app Apache2
Are you sure you want to restart Apache2 on server.sgdemo.com? [y/N] y
Sending request...
Calling Apache2::svcRestart
server.sgdemo.com > Apache2 $ ?

Help:
.          Repeat last command.
x          Reset internal buffers.
help       Print this help.
quit      Quit shell.
servers   List servers
server    Select a server. You can specify the server name, part of a name to search for, or a numeric option from the list of servers.
domains  List domains under selected server. You can pass the server name to get the domains under a server.
domain   Select a domain. You can specify the domain name, part of a name to search for, or a numeric option from the list of domains.
apps     List applications under selected server. You can pass the server name to get the apps under a server.
app      Select an app. You can specify the app name, part of a name to search for, or a numeric option from the list of apps.
reboot   Reboot a server. If no server name is given, it will reboot the selected server. It will ask for confirmation.
shutdown Shutdown a server. If no server name is given, it will shutdown the selected server. It will ask for confirmation.
bootup   Boot up a server. If no server name is given, it will boot the selected server. It will ask for confirmation.
restart  Restart an application. It will ask for confirmation.
stop     Stop an application. It will ask for confirmation.
start    Start an application.
exec    Execute a command in the server
login   Login with a different set of credentials

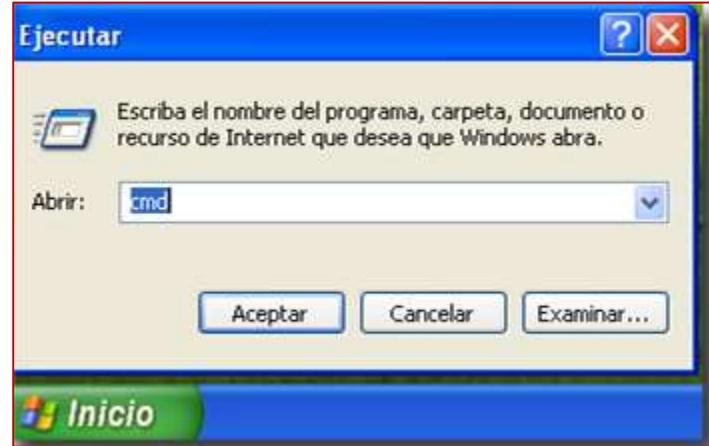
server.sgdemo.com > Apache2 $ |
```

Tipos de Interfaces

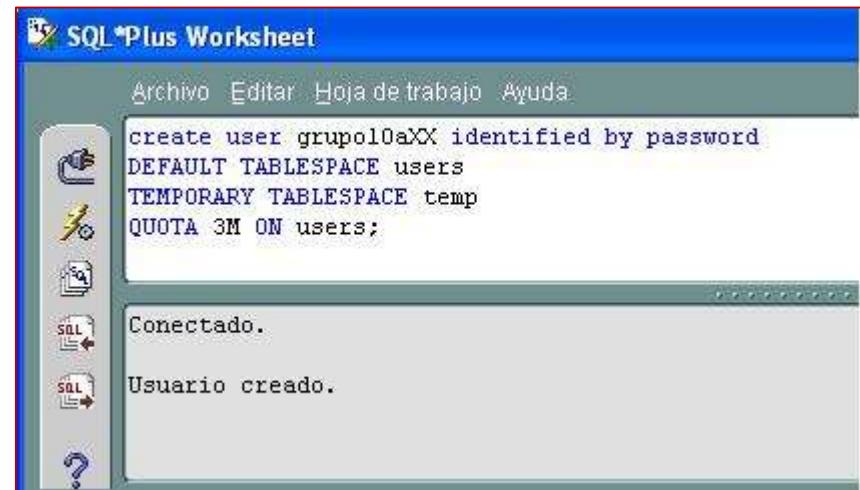
Interfaz de comandos

» Interfaz de comando a través de una interfaz gráfica

44



Ejecutar comandos de Windows

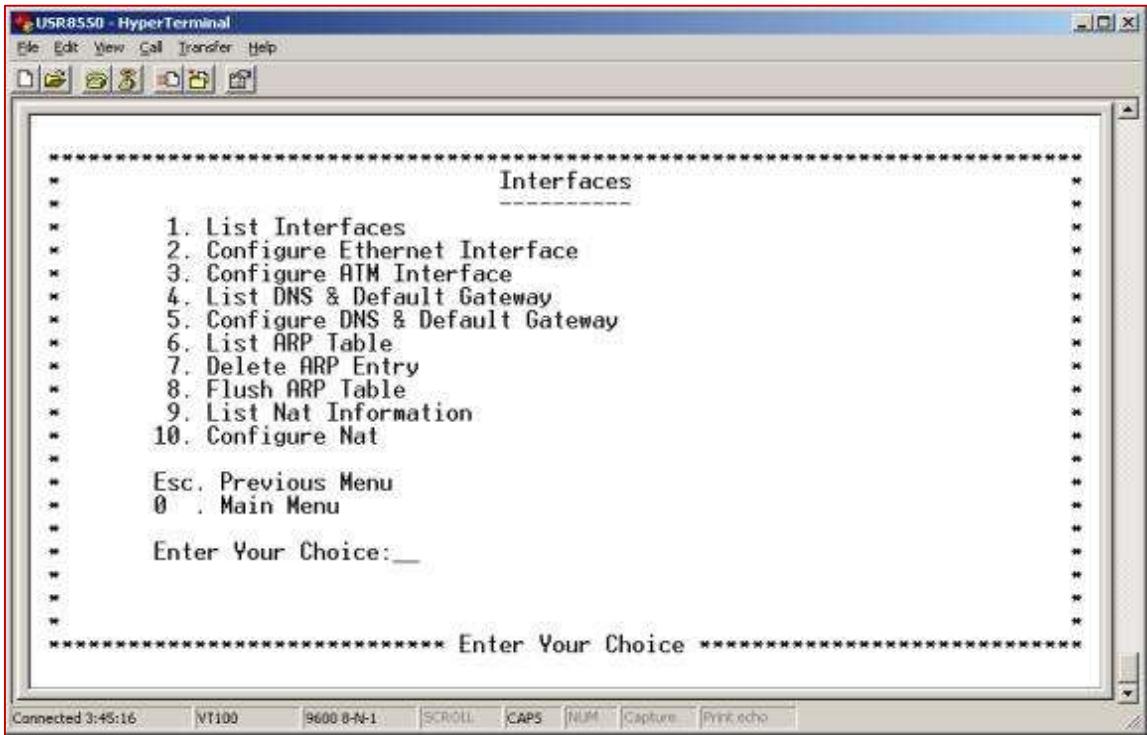


Ejecutar una consulta SQL utilizando la línea de comandos

Tipos de Interfaces

Interfaz de selección de menú

- » Se presentan un conjunto de opciones, que pueden ser seleccionadas por el usuario
 - » Solo se interactúa con los caracteres indicados
 - » Características:
 - Evita errores del usuario.
 - Lento para usuarios experimentados



45

Tipos de Interfaces

Interfaz gráfica de usuarios

- » Se caracterizan por la utilización de todo tipo de recursos visuales para la representación e interacción con el usuario.
- » Ventajas:
 - Son relativamente fáciles de aprender y utilizar.
 - Los usuarios cuentan con pantallas múltiples (ventanas) para interactuar con el sistema.
 - Se tiene acceso inmediato a cualquier punto de la pantalla.

46

Tipos de Interfaces

Interfaz gráfica de usuarios

» Ventanas



Fuente:

47

Tipos de Interfaces

Llenado de formularios

- » Introducción de datos sencilla en los campos de un formulario.
- » Es fácil de aprender pero ocupa mucho espacio en la pantalla



The screenshot shows a web form for sending a message to the government. The header includes the Spanish flag, the URL 'administracion.gob.es', and a search bar. The main section is titled 'Buzón de atención' and contains fields for 'Gestión Mensajes' (Message Management), 'Adjuntar Fichero' (Attach File), 'Identificación' (Identification), and a large text area for the message body. A note at the bottom specifies that file attachments must be less than 1 megabyte and that the message will receive a reference number. The footer contains terms about data protection and legal notices.

48

Más info

en:

<https://martinfowler.com/eaaDev/uiArchs.html>

Tipos de Interfaces

Interfaz gráfica de usuarios

»Iconos y Menús



49

Tipos de Interfaces

Interfaz de manipulación directa

» Interfaces de manipulación directa



Hardware Específico



Hardware Específico y evolución a la pantalla táctil

50

Tipos de Interfaces

Interfaz de manipulación directa

» Interfaces de manipulación directa táctil



51

Tipos de Interfaces

Reconocimiento de voz

»Comunicación con los dispositivos a través de la voz

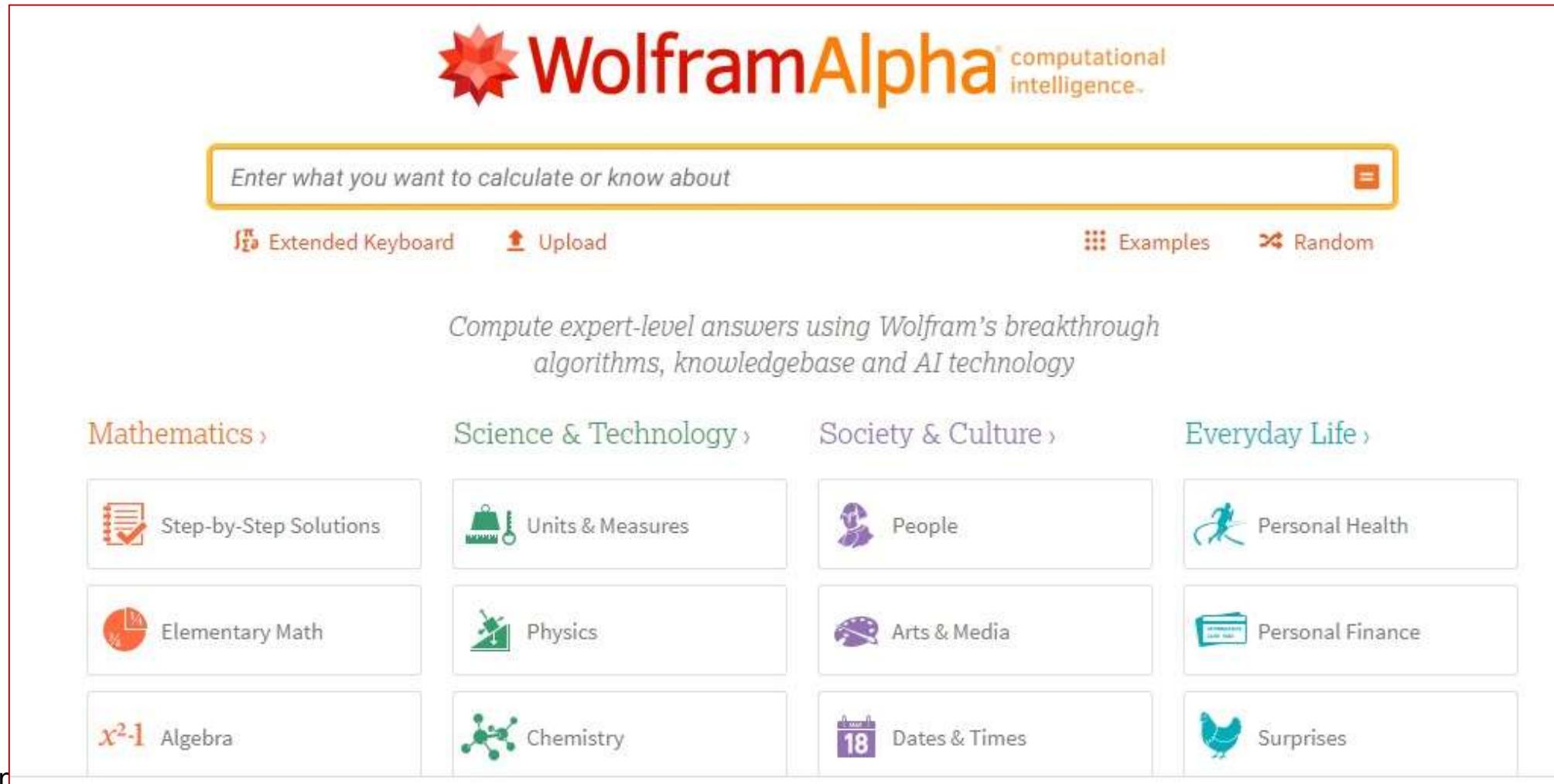


52

Tipos de Interfaces

Interfaz inteligente (IA)

»<https://www.wolframalpha.com/>



The screenshot shows the WolframAlpha search interface. At the top is the logo "WolframAlpha computational intelligence." Below it is a search bar with the placeholder "Enter what you want to calculate or know about". To the right of the search bar are buttons for "Extended Keyboard", "Upload", "Examples", and "Random". A large text below the search bar reads: "Compute expert-level answers using Wolfram's breakthrough algorithms, knowledgebase and AI technology". Below this are four main categories: Mathematics, Science & Technology, Society & Culture, and Everyday Life. Each category has three sub-options shown in a grid:

Mathematics	Science & Technology	Society & Culture	Everyday Life
Step-by-Step Solutions	Units & Measures	People	Personal Health
Elementary Math	Physics	Arts & Media	Personal Finance
Algebra	Chemistry	Dates & Times	Surprises

53

Tipos interfaces– Interfaz inteligente (IA)

Chatgpt <https://chat.openai.com/chat>



+ New chat

Software Engineer Q&A

IDLE es popular en Python.

Trabajo remoto seguro.

Park or Hat.

Identificación de Sentimientos

Experto en informática.

Clear conversations

Upgrade to Plus NEW

Light mode

Updates & FAQ

Log out

 Actúa como un experto ingeniero de software para responder estas preguntas

 ¡Por supuesto! Estoy listo para responder cualquier pregunta relacionada con la ingeniería de software. Adelante, dispara tus preguntas.

 En que se diferencia el diseño de experiencia de usuario del diseño de interfaz de usuario

 El diseño de experiencia de usuario (UX) y el diseño de interfaz de usuario (UI) son dos disciplinas diferentes pero relacionadas dentro del campo del diseño de software.

El diseño de experiencia de usuario se centra en el diseño general de cómo los usuarios interactúan con el software, y cómo se sienten al hacerlo. Se trata de comprender las necesidades y expectativas de los usuarios y diseñar soluciones que sean intuitivas, fáciles de usar y agradables. El diseño de ex Stop generating implica la

Send a message...

ChatGPT Mar 14 Version. Free Research Preview. Our goal is to make AI systems more natural and safe to interact with. Your feedback will help us improve.

Tipos de Interfaces

Interfaces para diferentes dispositivos



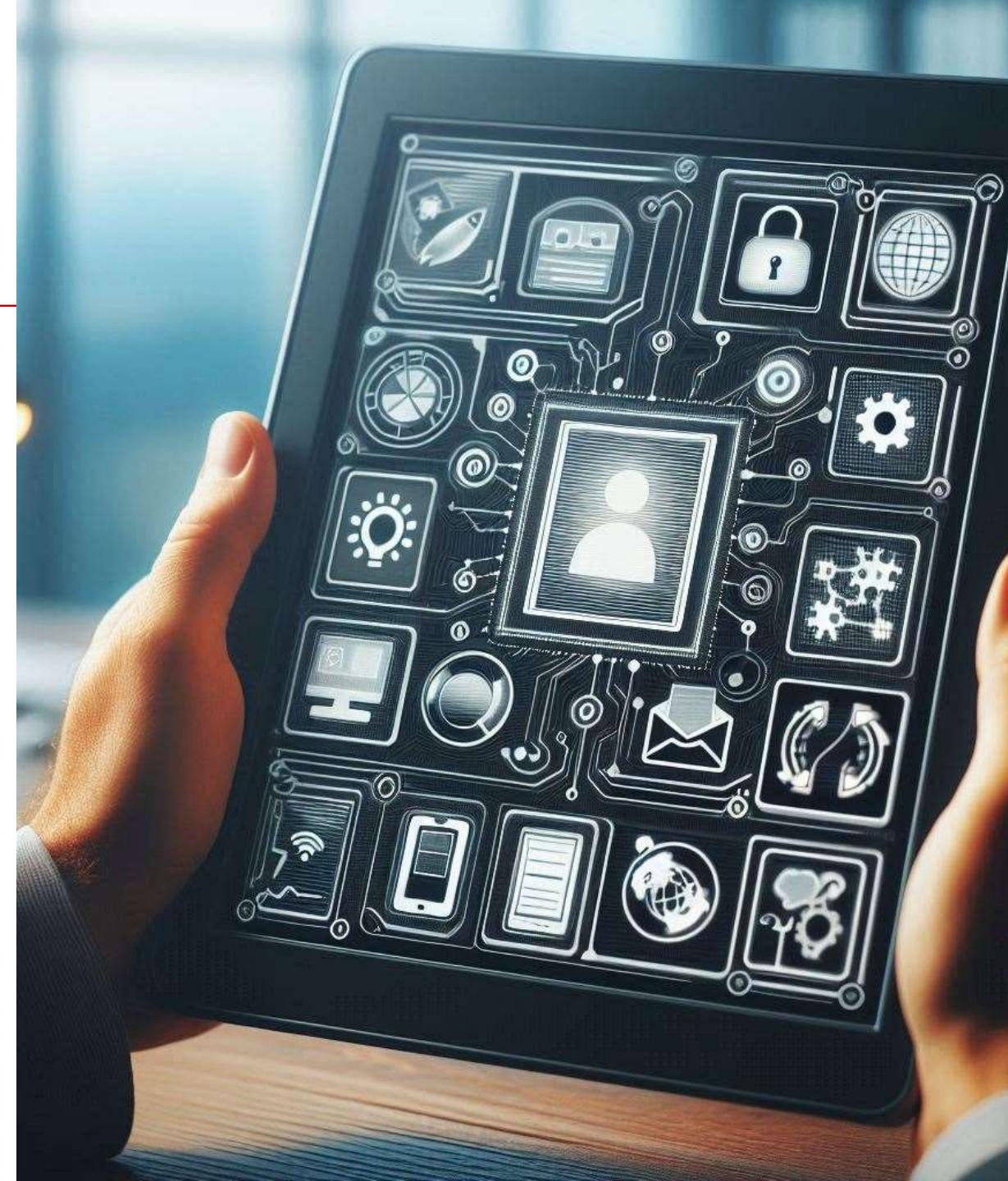
55

Responsive Web Design
(Interface Web adaptable a cada dispositivo)

Tipos de Interfaces

Interfaces Accesibles

Son las interfaces que respetan las normas del diseño universal para que puedan ser accedidas por cualquier usuario independientemente de sus condiciones físicas y mentales.



Comparación de tipos de interfaces

Característica	Interfaz de Comandos	Selección de Menú	GUI	Relleno de Formularios	Manipulación Directa	Reconocimiento de Voz	Inteligente
👉👤 Estilo de Interacción	Comandos de Texto	Elegir entre opciones presentadas	Elementos Visuales	Entrada de Datos	Interacción con Objetos	Comandos Hablados	Adaptativo
💻💡 Curva de Aprendizaje	Empinada	Moderada	Suave	Moderada	Muy Suave	Moderada	Variable
⌚️ Velocidad	Rápido	Moderada	Moderada	Moderada	Rápido	Variable	Variable
🧠 Intuición	Baja	Moderada	Alta	Moderada	Muy Alta	Moderada	Alta
💻💡 Facilidad de Uso	Moderada	Alta	Alta	Moderada	Muy Alta	Moderada	Alta



Presentación de la información en pantalla

Presentación de la información (Ux)

- » Se deben conocer los usuarios y como utilizarán el sistema.
- » ¿Información precisa o relación entre los valores?
- » ¿Es necesario presentar inmediatamente los cambios?
- » ¿El usuario realiza acciones en función de los cambios?
- » ¿Información textual o numérica?
- » ¿Información estática o dinámica?

59

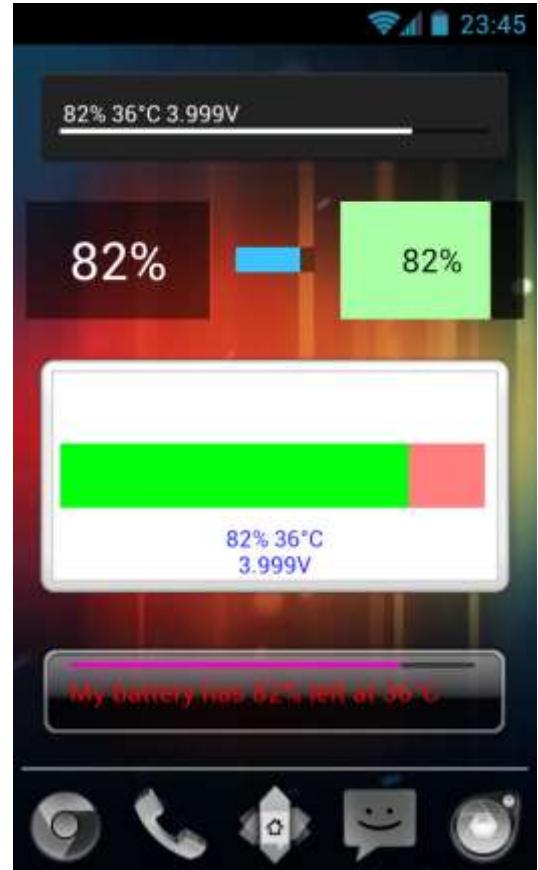
Presentación de la información

- » Mantener separada la lógica del software de la presentación y la información misma (enfoque MVC)

Presentación de la Información de manera
Directa

82 %

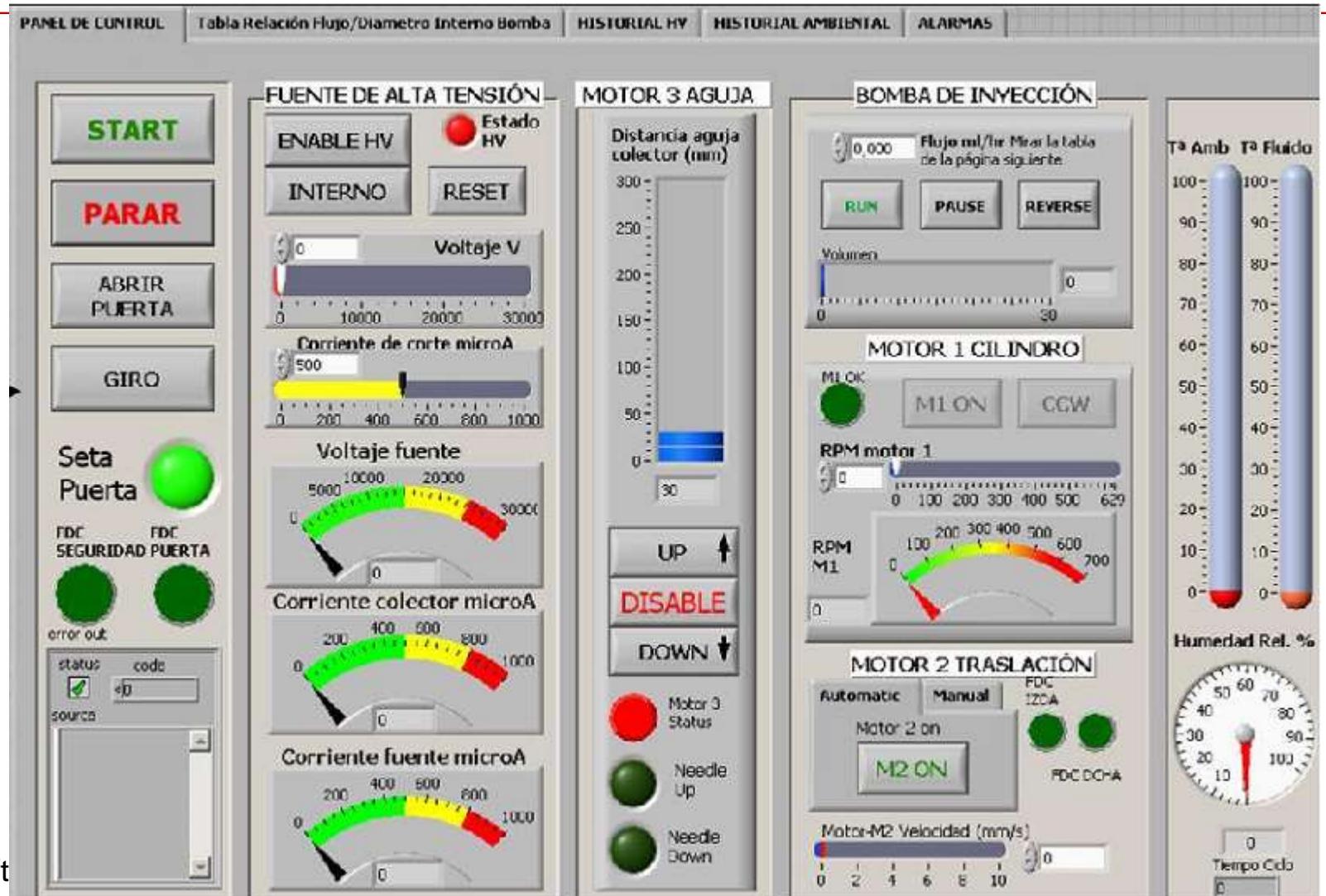
Presentación de la Información de manera
Gráfica



60

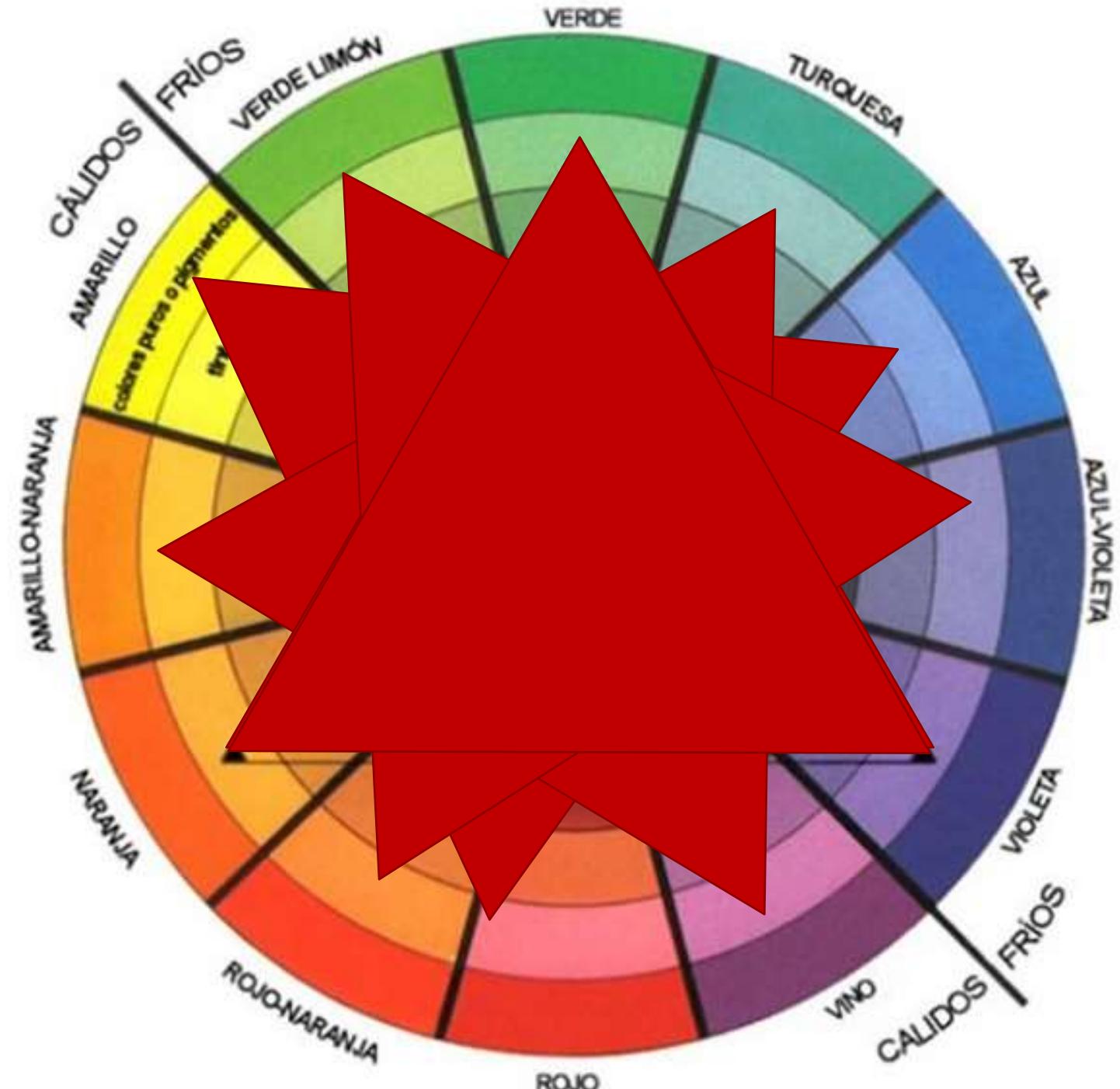
Presentación de la información

Simulador de una Central Hidroeléctrica



Presentación de la información - Colores

- No utilizar mas de 4 ó 5 colores diferentes en una ventana y no más de 7 en la interfaz total del sistema.
- Utilizar el **código de colores para apoyar la tarea** que los usuarios están tratando de llevar a cabo.
- Ser cuidadoso al utilizar **grupos de colores**.
- Si se utilizan muchos colores o sin son muy brillantes, **el despliegue puede ser confuso**



Presentación de la información - Colores

- Limitar el número de colores utilizados.
- No asociar solamente colores a significados.

10% de los humanos no perciben el color.

Acompañarlos de algún otro tipo de identificación

63

- Usar los colores consistentemente.
- Usar cambio de color para mostrar cambios en el estado del sistema.
- Combinar los colores cuidadosamente.

Soporte al usuario

- » Mensajes del sistema por acciones del usuario.
- » Ayudas en línea.
- » Documentación del sistema.



Server Error

The server encountered a temporary error and could not complete your request.

Please try again in 30 seconds.





Herramientas de prototipado

65

Ver en el curso el apartado de “Recursos para el diseño de interfaces”



Ejemplos de Interfaces

66

Ver en el curso el apartado de “Ejemplos de interfaces para explorar”

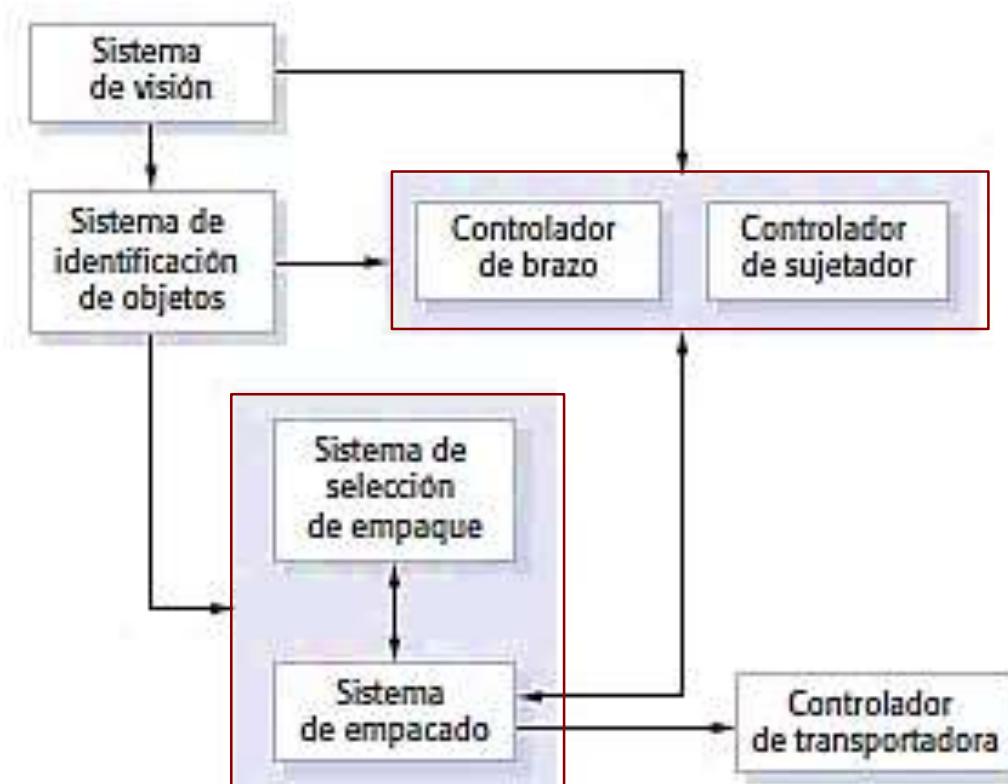


Ingeniería de software II

Diseño de Software – Diseño arquitectónico

Diseño Arquitectónico

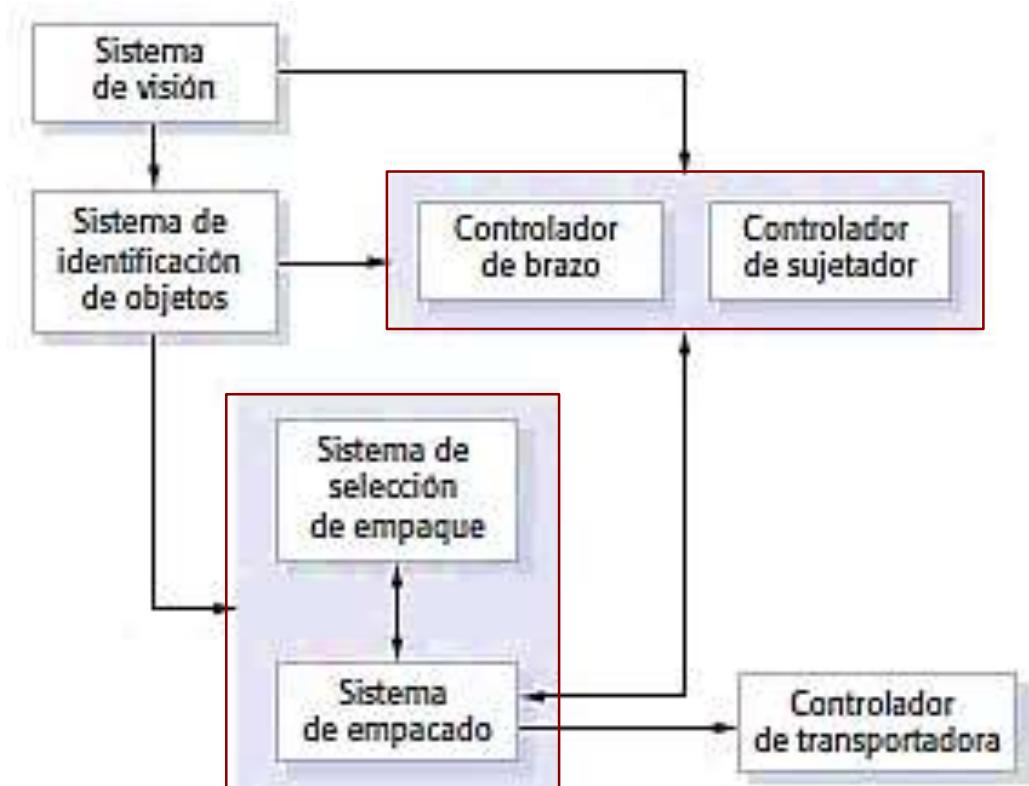
- ❖ Define la relación entre los elementos estructurales, para lograr los requisitos del sistema.
- ❖ Es el proceso de identificar los subsistemas dentro del sistema y establecer el marco de control y comunicación entre ellos.
- ❖ Los grandes sistemas se dividen en subsistemas que proporcionan algún conjunto de servicios relacionados



Diseño Arquitectónico

En la figura se presenta un modelo abstracto de la arquitectura para un sistema de robot de empaquetado, que indica los componentes que tienen que desarrollarse.

El modelo arquitectónico presenta dichos componentes y los vínculos entre ellos.



Diseño Arquitectónico

»La arquitectura afecta directamente a los requerimientos no funcionales más CRÍTICOS:

- ❖ *Rendimiento*
- ❖ *Protección*
- ❖ *Seguridad*
- ❖ *Disponibilidad*
- ❖ *Mantenibilidad*

Requerimientos No Funcionales que impactan en la Arquitectura de Software

Diseño Arquitectónico

5



- 1. Organización del sistema**
- 2. Descomposición modular**
- 3. Modelos de control**
- 4. Arquitectura de los Sistemas Distribuidos**

6

Organización del Sistema

» La organización del sistema representa la estrategia básica usada para estructurar el sistema

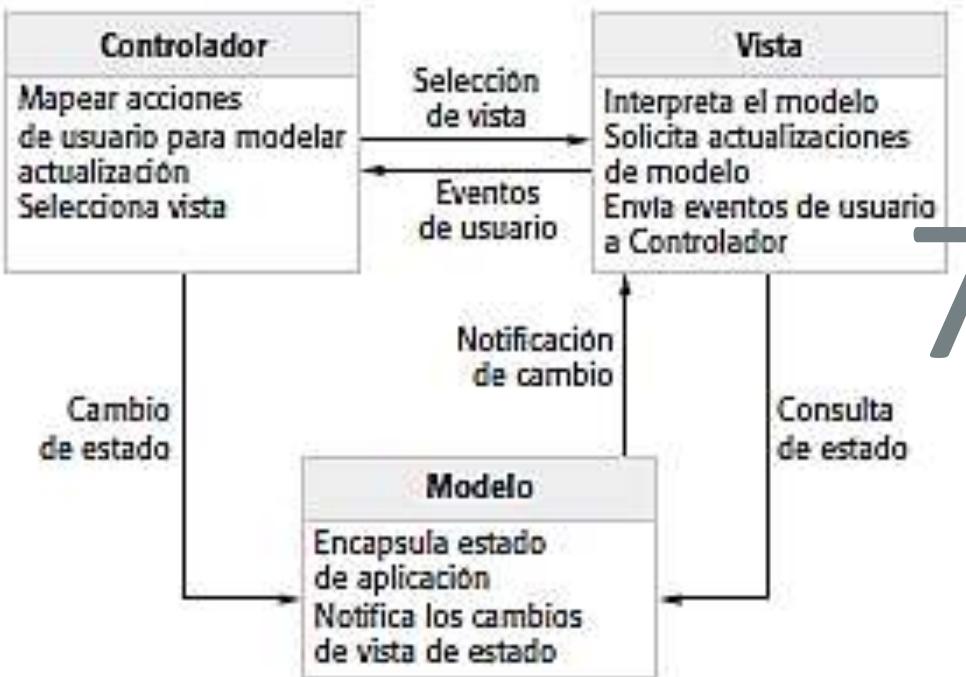
Los subsistemas de un sistema deben intercambiar información de forma efectiva

Todos los datos compartidos, se almacenan en una base de datos central

Cada subsistema mantiene su información y los intercambia entre los subsistemas

Estilos organizacionales (**Patrones arquitectónicos**)

Repositorio, cliente-servidor, capas o combinaciones entre ellos, entre otros



Organización del Sistema

Patrón de repositorio

La mayoría de los sistemas que usan grandes cantidades de datos se organizan alrededor de una base de datos compartida (repositorio)

Los datos son generados por un subsistema y utilizados por otros subsistemas

8

Ejemplo

Sistemas de gestión, Sistemas CAD, Herramientas Case, etc.

Ejemplos de patrones de repositorios

Comparación de Sistemas de Gestión, CAD y CASE

Característica	Sistema de Gestión	Sistema CAD	Herramienta CASE
 Repository Central	Base de datos con información crítica	Archivo de diseño con entidades gráficas	Repositorio con información del proyecto de software
 Componentes Independientes	Módulos que acceden y manipulan datos centrales	Herramientas que operan en el archivo de diseño central	Herramientas que interactúan con el repositorio central

Organización del Sistema

Patrón de repositorio



10

Organización del Sistema

Patrón de repositorio

Ventajas

- ❖ Forma eficiente de compartir grandes cantidades de datos, no hay necesidad de transmitir datos de un subsistema a otro
- ❖ Los subsistemas que producen datos no deben saber como se utilizan
- ❖ Las actividades de backup, protección, control de acceso están centralizadas.
- ❖ El modelo compartido es visible a través del esquema del repositorio. Las nuevas herramientas se integran de forma directa, ya que son compatibles con el modelo de datos

11

Organización del Sistema

Patrón de repositorio

Desventajas

- ❖ Los subsistemas deben estar acordes a los modelos de datos del repositorio. Esto en algunos casos puede afectar el rendimiento.
- ❖ La evolución puede ser difícil a medida que se genera un gran volumen de información de acuerdo con el modelo de datos establecido. La migración de estos modelos puede ser muy difícil, en algunos casos imposible.
- ❖ Diferentes subsistemas pueden tener distintos requerimientos de protección o políticas de seguridad y el modelo de repositorio impone las mismas para todos.
- ❖ Es difícil distribuir el repositorio en varias máquinas, existen repositorios centralizados lógicamente pero pueden ocasionar problemas de redundancia e inconsistencias.

12

Organización del Sistema

Patrón cliente-servidor

Es un modelo donde el sistema se organiza como un conjunto de servicios y servidores asociados, más unos clientes que utilizan los servicios

Componentes

Un conjunto de servidores que ofrecen servicios

Un conjunto de clientes que llaman a los servicios

Una red que permite a los clientes acceder a los servicios

Caso particular cuando los servicios y el cliente corren en la misma máquina

Los clientes conocen el nombre del servidor y el servicio que brinda, pero el servidor no necesita conocer al cliente

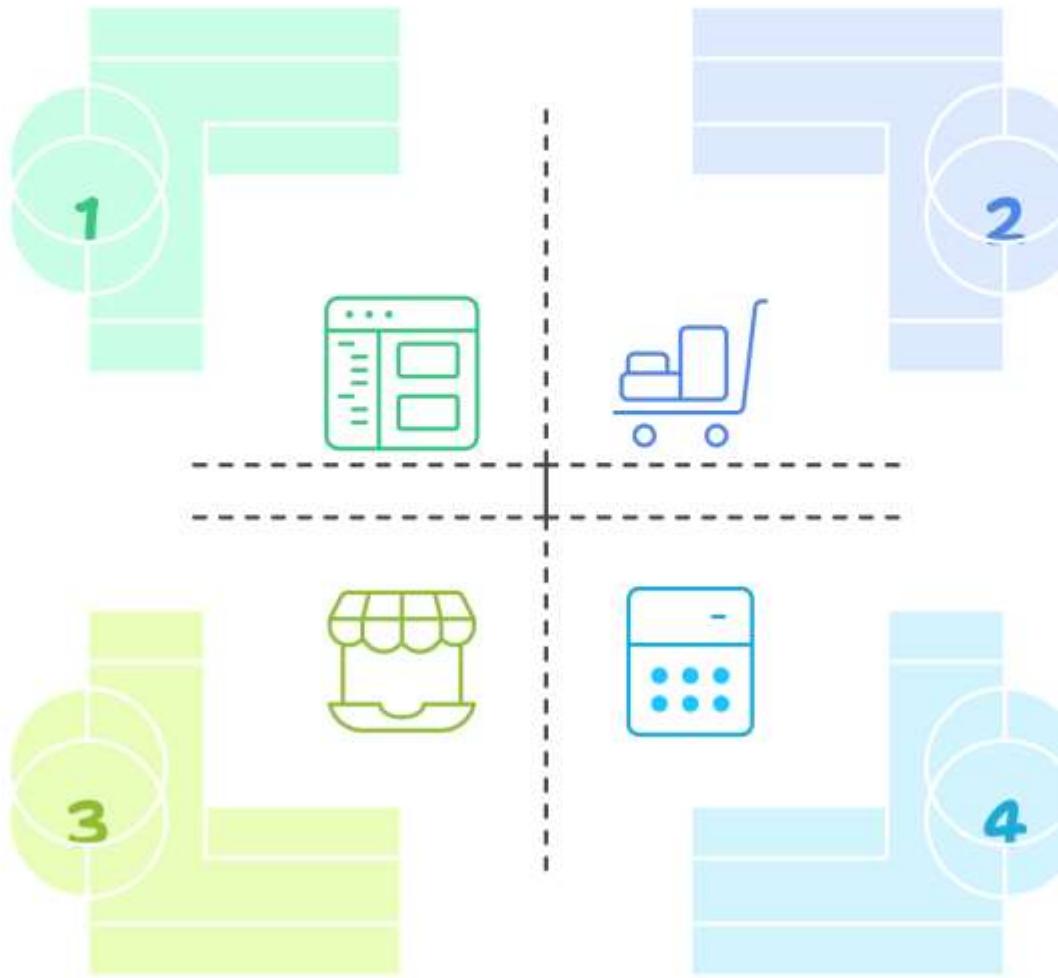
13

Ejemplos de Arquitecturas de Software en Capas

Ejemplos

Aplicaciones web con bases de datos

Aplicaciones web con bases de datos usan múltiples capas.



Plataformas de comercio electrónico

Plataformas de comercio electrónico organizan capas para pedidos.

Aplicaciones de gestión de inventario

Aplicaciones de gestión de inventario separan interfaz y lógica de datos.

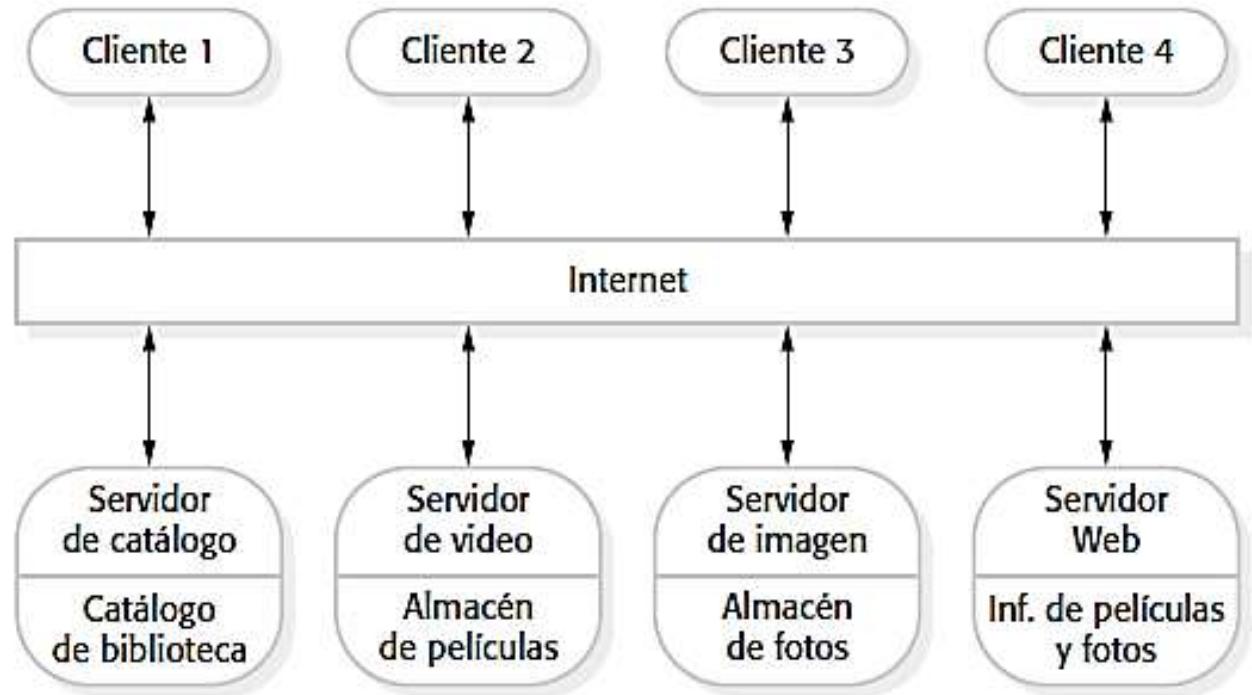
Aplicaciones de contabilidad de escritorio

Aplicaciones de contabilidad de escritorio separan interfaz y datos.

Made with Napkin

Organización del Sistema

Patrón cliente-servidor



15

Organización del Sistema

»**Patrón** de arquitectura en capas

El sistema se organiza en capas, donde cada una de ellas presenta un conjunto de servicios a sus capas adyacentes

Ventajas

- ❖ *Soporta el desarrollo incremental*
- ❖ *Es portable y resistente a cambios*
- ❖ *Una capa puede ser reemplazada siempre que se mantenga la interfaz, y si varía la interfaz se genera una capa para adaptarlas*
- ❖ *Permite generar sistemas multiplataforma, ya que solamente las capas más internas son dependientes de la plataforma (se genera una capa interna para cada plataforma)*

16

Organización del Sistema

»**Patrón** de arquitectura en capas

Desventajas

- ❖ *Difícil de estructurar*
- ❖ *Las capas internas proporcionan servicios que son requeridos por todos los niveles*
- ❖ *Los servicios requeridos por el usuario pueden estar brindados por las capas internas teniendo que atravesar varias capas adyacentes*
- ❖ *Si hay muchas capas, un servicio solicitado de la capa superior puede tener que ser interpretado varias veces en diferentes capas*

17

Arquitectura de Aplicaciones en Capas

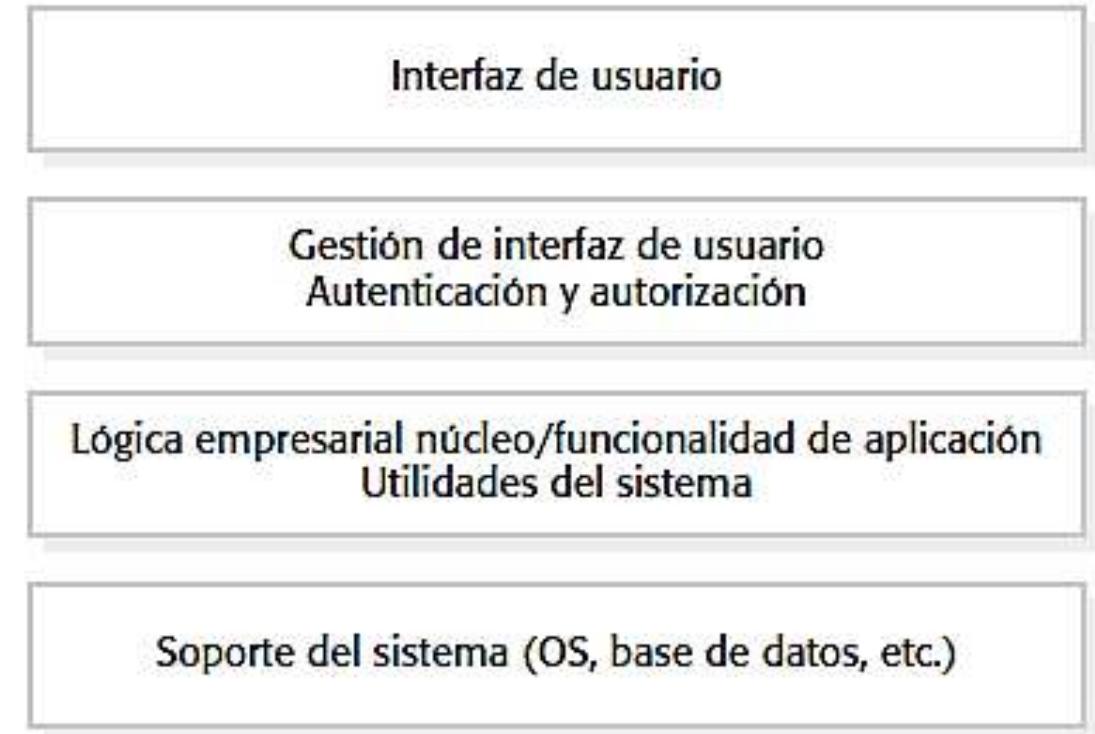
Capas típicas del software

18



Organización del Sistema

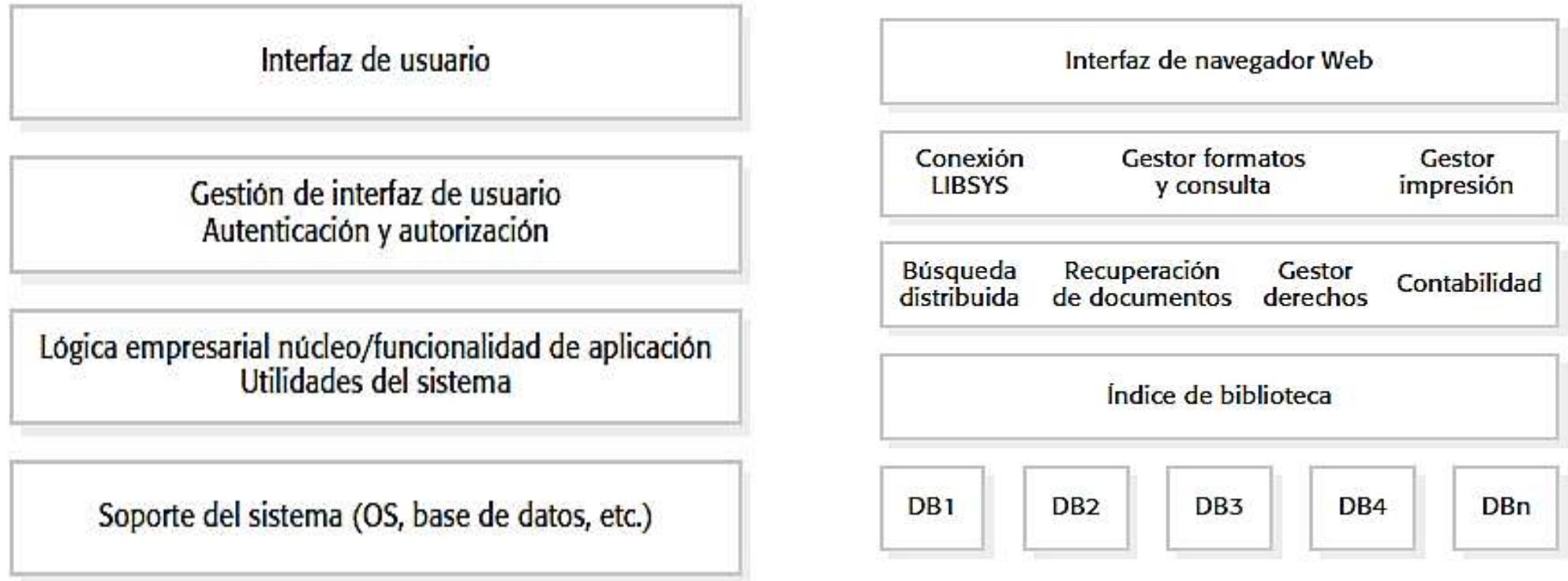
»*Patrón* de arquitectura en capas



19

Organización del Sistema

» Ejemplo de **Patrón** de arquitectura en capas:



20

1. Organización del sistema
2. Descomposición modular
3. Modelos de control
4. Arquitectura de los Sistemas Distribuidos

21

Descomposición Modular

» Una vez organizado el sistema, a los subsistemas los podemos dividir en módulos, se puede aplicar los mismos criterios que vimos en la organización, pero la descomposición modular es más pequeña y permite utilizar otros estilos alternativos.

Estrategias de Descomposición Modular



22

Descomposición Modular

» Definiciones

Subsistema

Es un sistema en sí mismo cuyo funcionamiento no depende de los servicios proporcionados por otros. Los subsistemas se componen de módulos con interfaces definidas que se utilizan para comunicarse con otro subsistemas.

Módulo

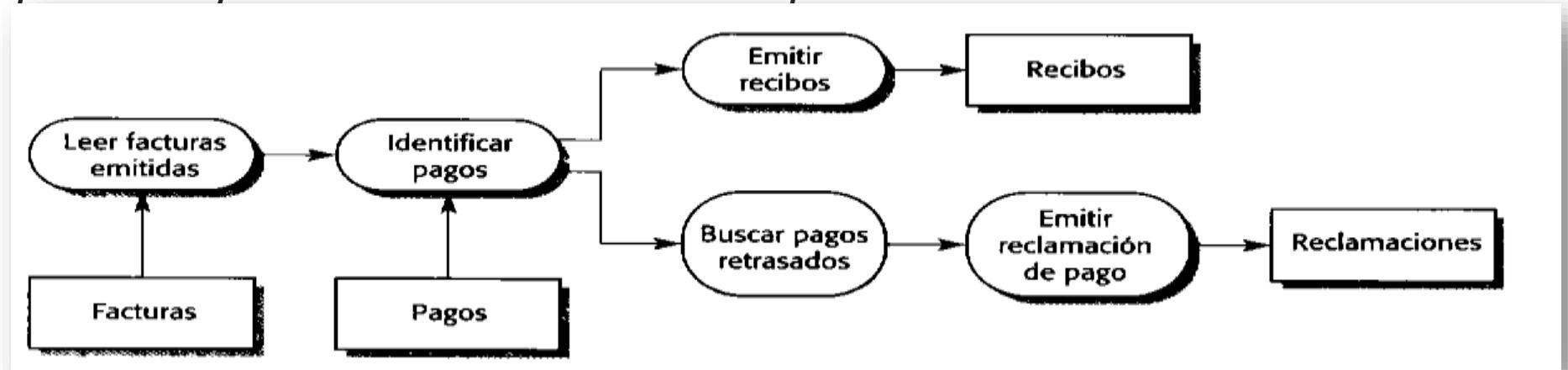
Es un componente de un subsistema que proporciona uno o más servicios a otros módulos. A su vez utiliza servicios proporcionados por otros módulos. Por lo general no se los considera un sistema independiente.

23

Descomposición Modular

» Descomposición orientada a flujo de funciones

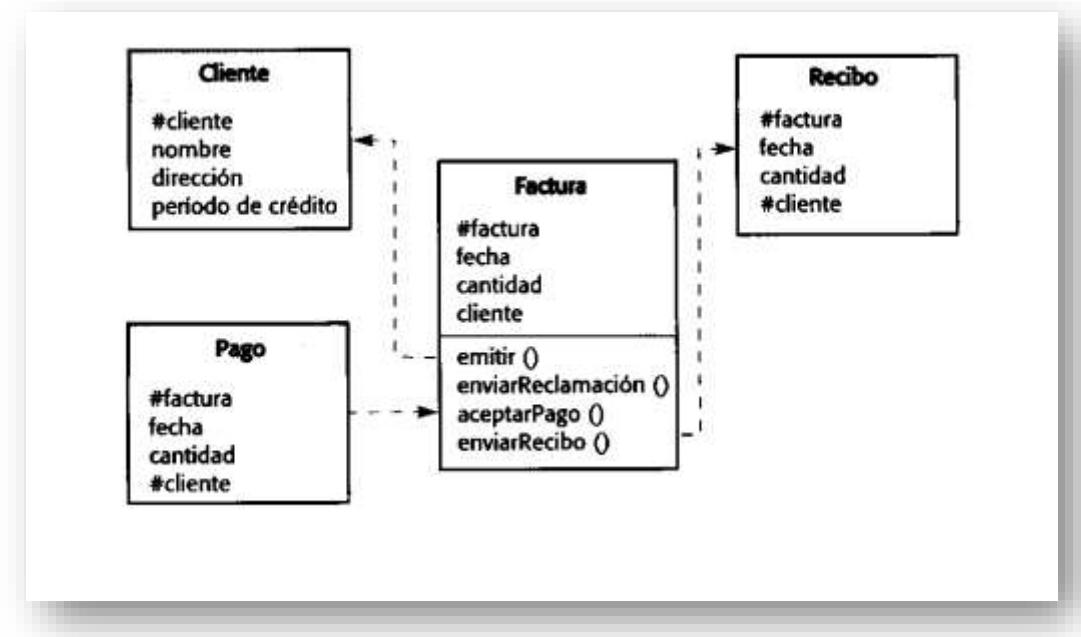
En un Modelo orientado a flujo de funciones , los datos fluyen de una función a otra y se transforman a medida que pasan por una secuencia de funciones hasta llegar a los datos de salida. Las transformaciones se pueden ejecutar en secuencial o en paralelo.



Descomposición Modular

» Descomposición orientada a objetos

Un modelo arquitectónico orientado a objetos estructura al sistema en un conjunto de objetos débilmente acoplados y con interfaces bien definidas.



25

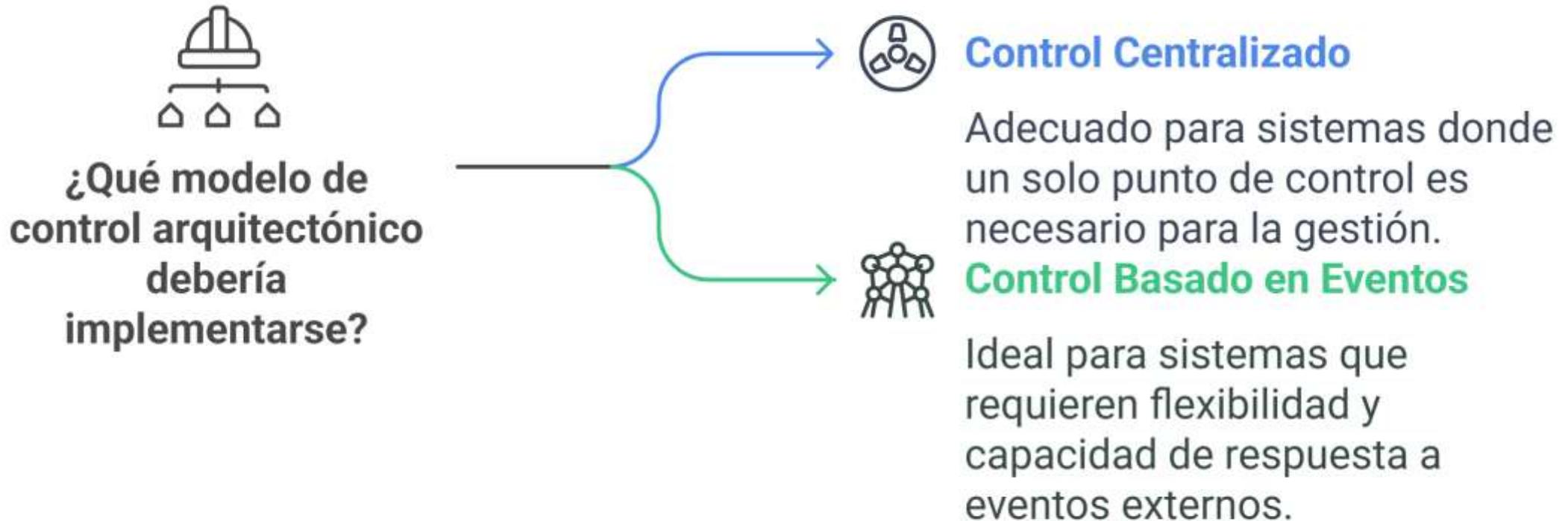
Diseño Arquitectónico

1. Organización del sistema
2. Descomposición modular
- 3. Modelos de control**
4. Arquitectura de los Sistemas Distribuidos

26

Modelos de Control

» En un sistema, los subsistemas están controlados para que sus servicios se entreguen en el lugar correcto en el momento preciso.



Modelos de Control

»*Control Centralizado*

Un subsistema se diseña como controlador y tiene la responsabilidad de gestionar la ejecución de otros subsistemas, la ejecución puede ser secuencial o en paralelo

Modelo de llamada y retorno

Modelo de subrutinas descendentes

Aplicable a modelos secuenciales

Modelo de gestor

Un gestor controla el inicio y parada coordinado con el resto de los procesos

Aplicable a modelos concurrentes

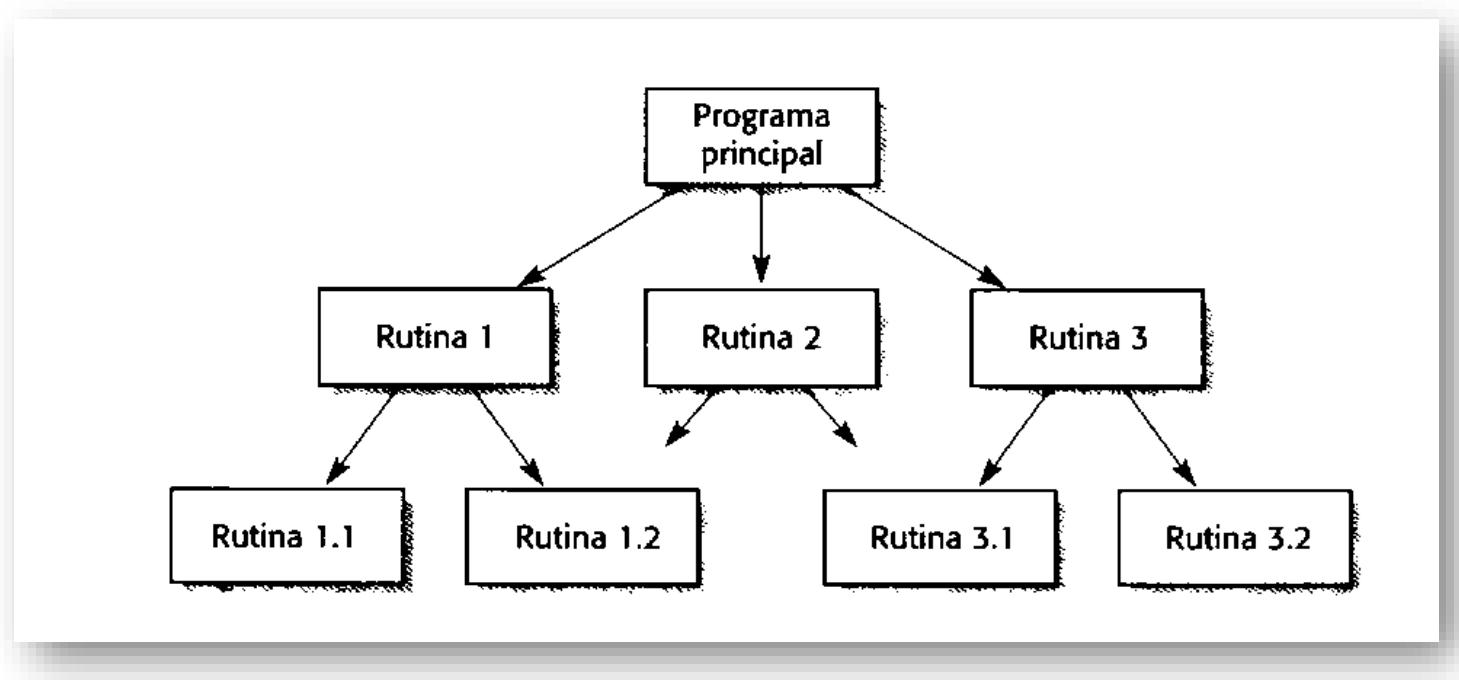
28

Modelos de Control

»*Control* Centralizado

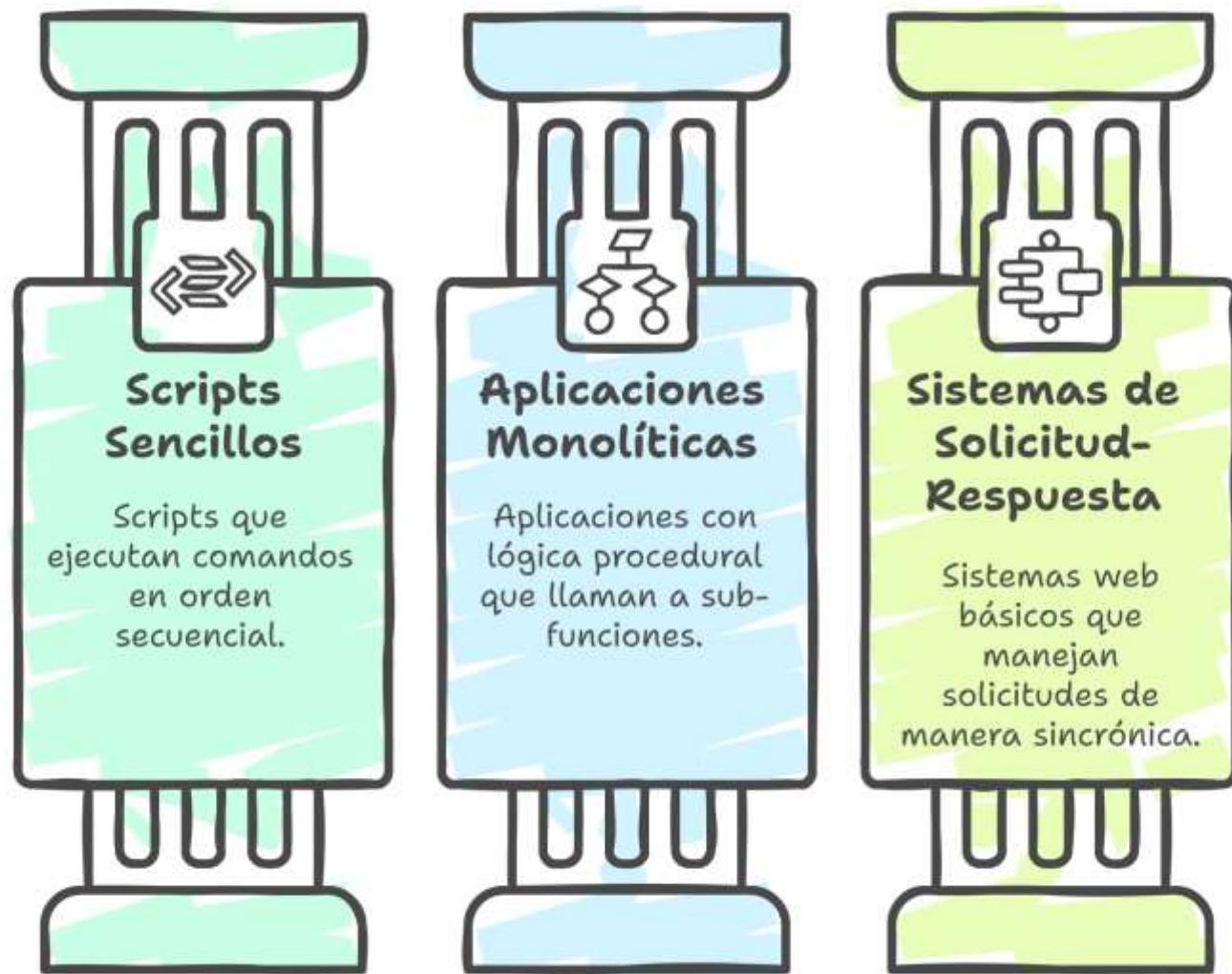
Modelo de llamada y retorno

29



Estructura de Llamada y Retorno

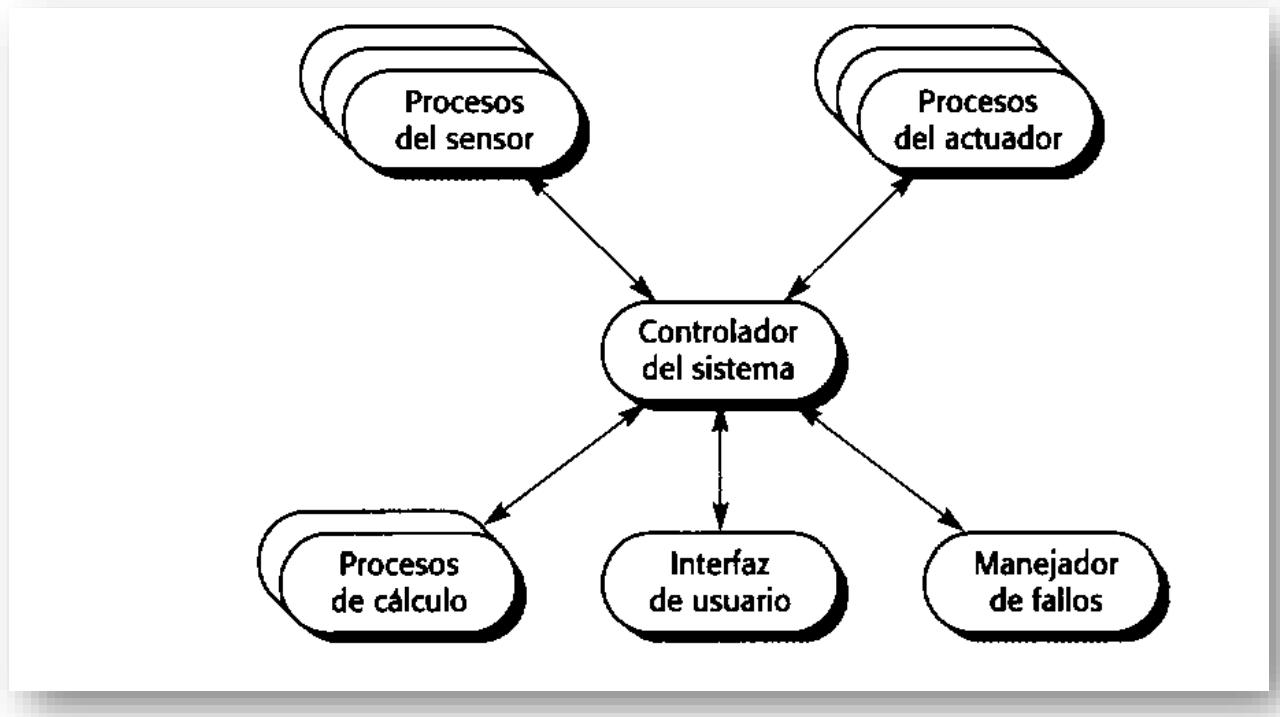
Ejemplos de control centralizado- Llamada retorno



30

Modelos de Control

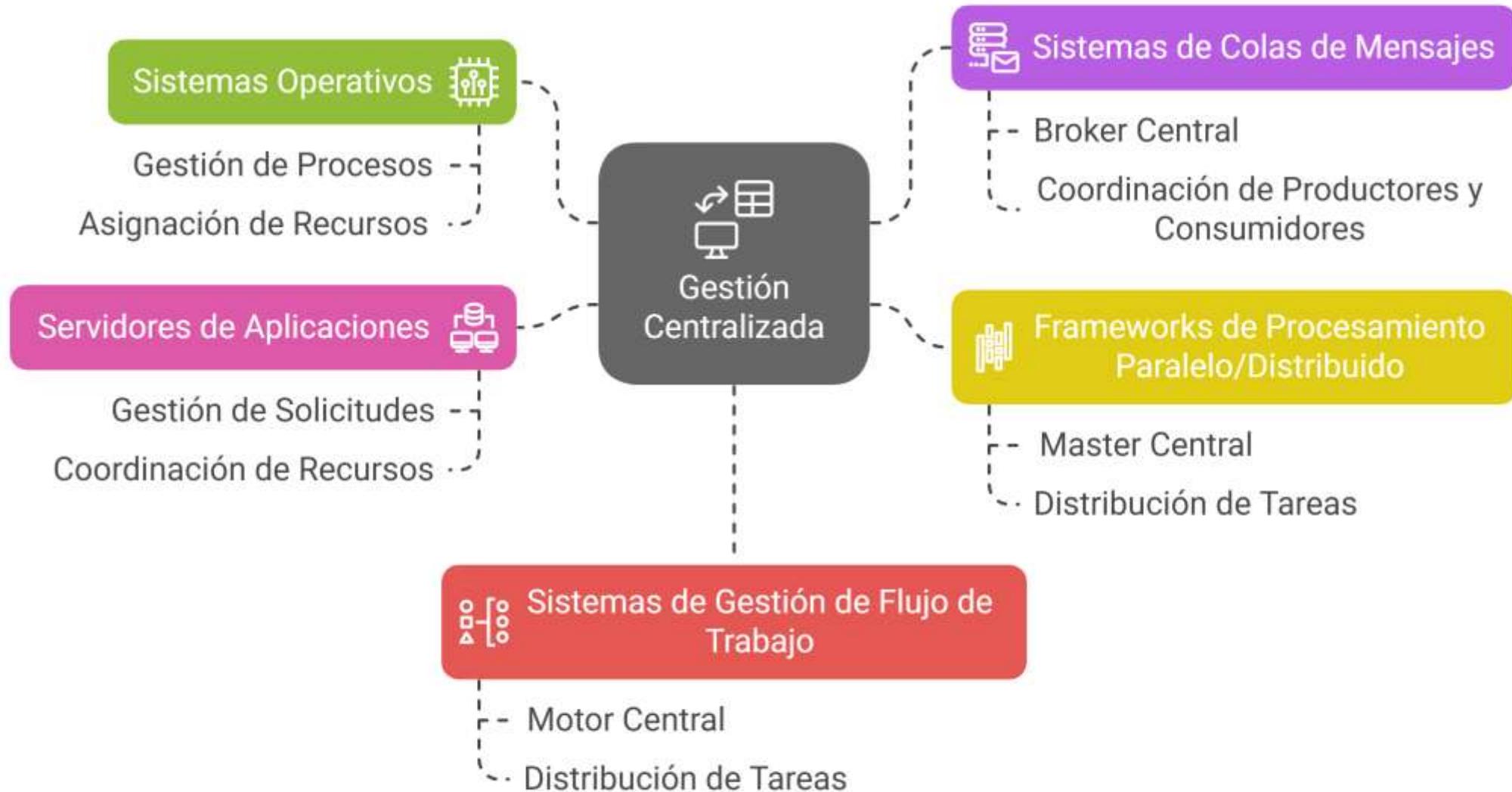
» ***Control*** Centralizado
Modelo de gestor



31

Gestión Centralizada en Sistemas de Software

Ejemplos de control centralizado – Modelo de gestor



Modelos de Control

»Sistemas Dirigidos Por Eventos

- Se rigen por eventos generados externamente al proceso
- Eventos

Señal binaria

Un valor dentro de un rango

Una entrada de un comando

Una selección del menú

- Modelos de sistemas dirigidos por eventos

Modelos de transmisión (Broadcast)

Modelo dirigido por interrupciones

33

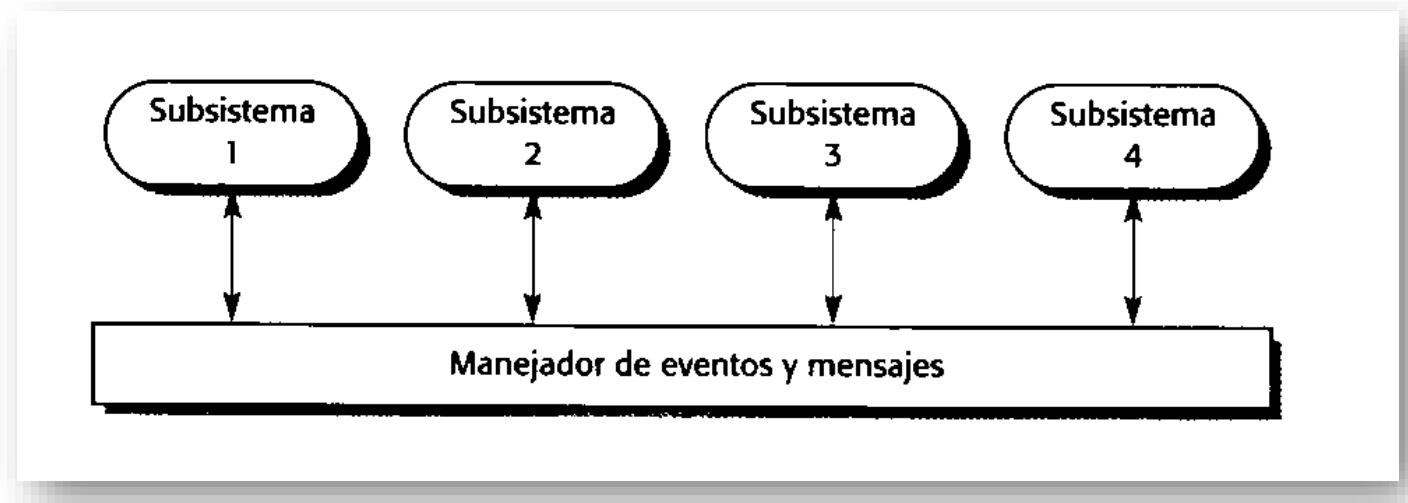
Modelos de Control

»Sistema *Dirigido* Por Eventos

Modelos de transmisión (Broadcast)

Un evento se trasmite a todos los subsistemas, cualquier subsistema programado para manejar ese evento lo atenderá

34



Ejemplo de modelo de control por eventos basado en broadcast

Ejemplos de Arquitectura Basada en Eventos por broadcast



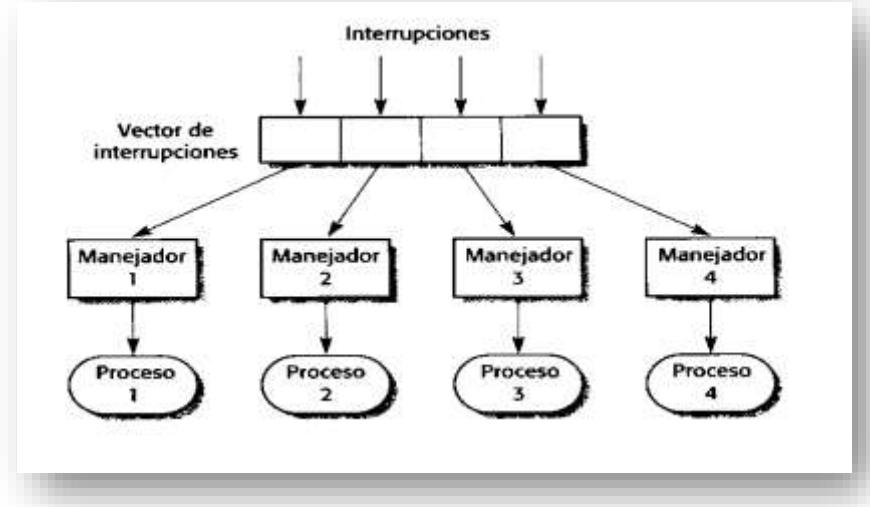
Modelos de Control

»Sistema *Dirigido* Por Eventos

Modelo *Dirigido* por interrupciones

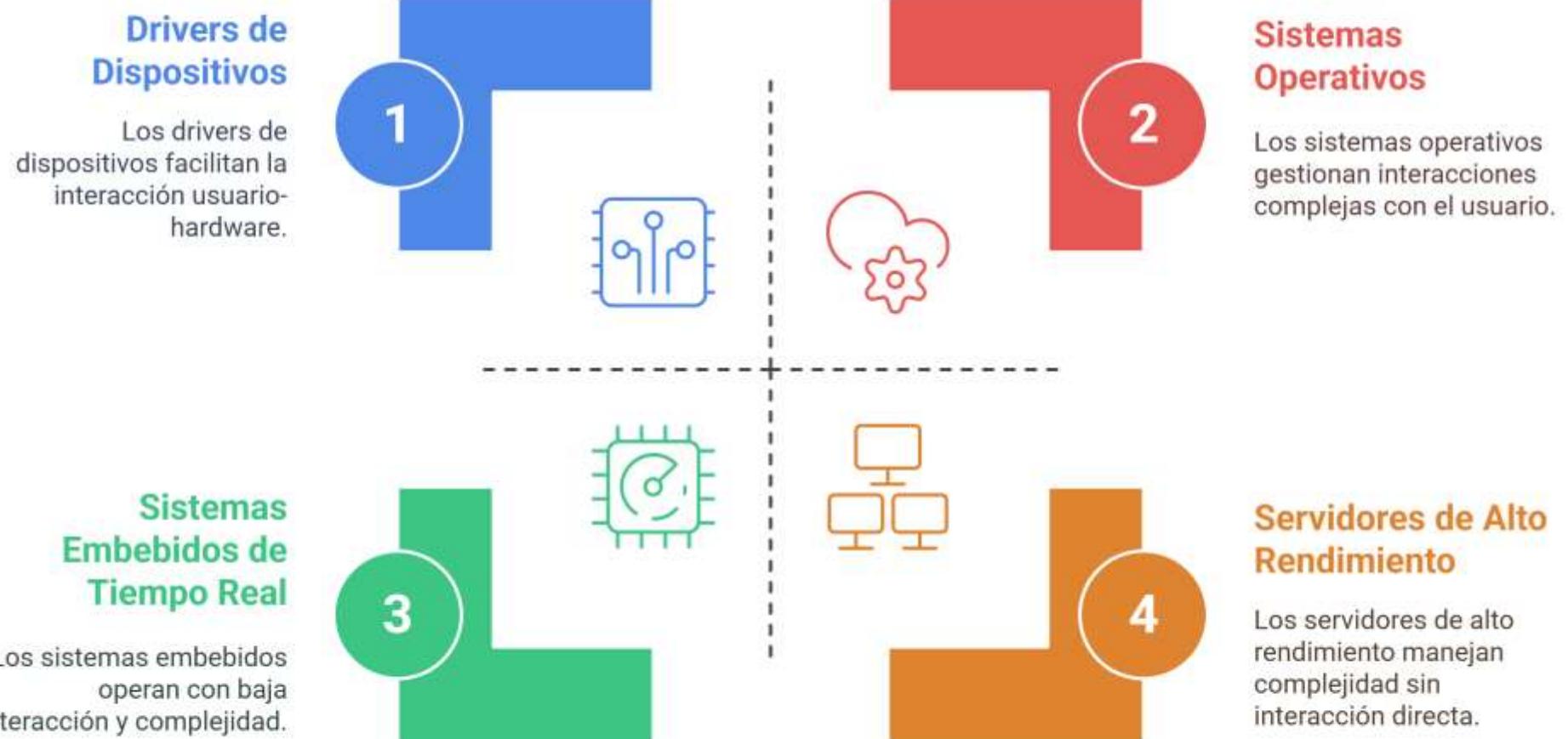
Se utilizan en sistemas de tiempo real donde las interrupciones externas son detectadas por un manejador de interrupciones y se envía a algún componente para su procesamiento

36



Ejemplos de modelos de control dirigido por interrupciones

Aplicaciones de Control Dirigido por Interrupciones



Made with Napkin

Diseño Arquitectónico

1. Organización del sistema
2. Descomposición modular
3. Modelos de control
4. Arquitectura de los Sistemas Distribuidos

38

Arquitectura de los Sistemas Distribuidos

- » Un sistema distribuido es un sistema en el que el procesamiento de información se distribuye sobre varias computadoras.

- » Tipos genéricos de sistemas distribuidos
 - Cliente-Servidor
 - Componentes distribuidos

39

Arquitectura de los Sistemas Distribuidos

»Características de los sistemas distribuidos

Compartir recursos

Un sistema distribuido permite compartir recursos

Apertura

Son sistemas abiertos y se diseñan con protocolos estándar para simplificar la combinación de los recursos

Concurrencia

Varios procesos pueden operar al mismo tiempo sobre diferentes computadoras

Escalabilidad

La capacidad puede incrementarse añadiendo nuevos recursos para cubrir nuevas demandas

Tolerancia a fallos

La disponibilidad de varias computadoras y el potencial para reproducir información hace que los sistemas distribuidos sean más tolerantes a fallos de funcionamiento de hardware y software

40

Arquitectura de los Sistemas Distribuidos

» Desventajas de los sistemas distribuidos

Complejidad

Son más complejos que los centralizados, además del procesamiento hay que tener en cuenta los problemas de la comunicación y sincronización entre los equipos

Seguridad

Se accede al sistema desde varias computadoras generando tráfico en la red que puede ser intervenido

Manejabilidad

Las computadoras del sistema pueden ser de diferentes tipos y diferentes S.O. lo que genera más dificultades para gestionar y mantener el sistema

Impredecibilidad

La respuesta depende de la carga del sistema y del estado de la red, lo que hace que el tiempo de respuesta varíe entre una petición y otra

41

Tipos de Arquitectura de los Sistemas Distribuidos (SD)

42



Arquitectura de los Sistemas Distribuidos

»Arquitectura Multiprocesador

El sistema de software está formado por varios procesos que pueden o no ejecutarse en procesadores diferentes

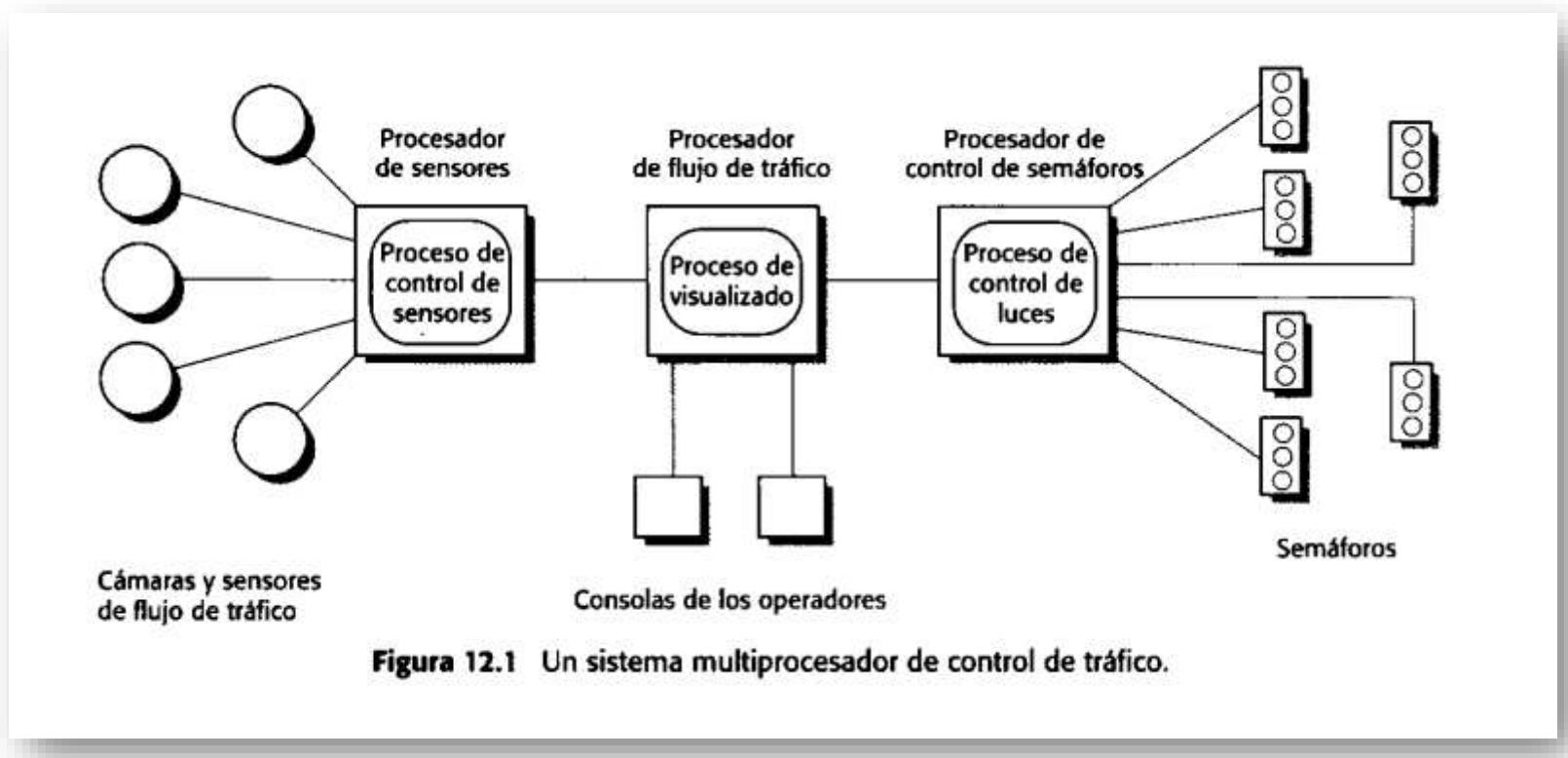
La asignación de los procesos a los procesadores puede ser predeterminada o mediante un dispatcher

Es común en sistemas grandes de tiempo real que recolectan información, toman decisiones y envían señales para modificar el entorno

43

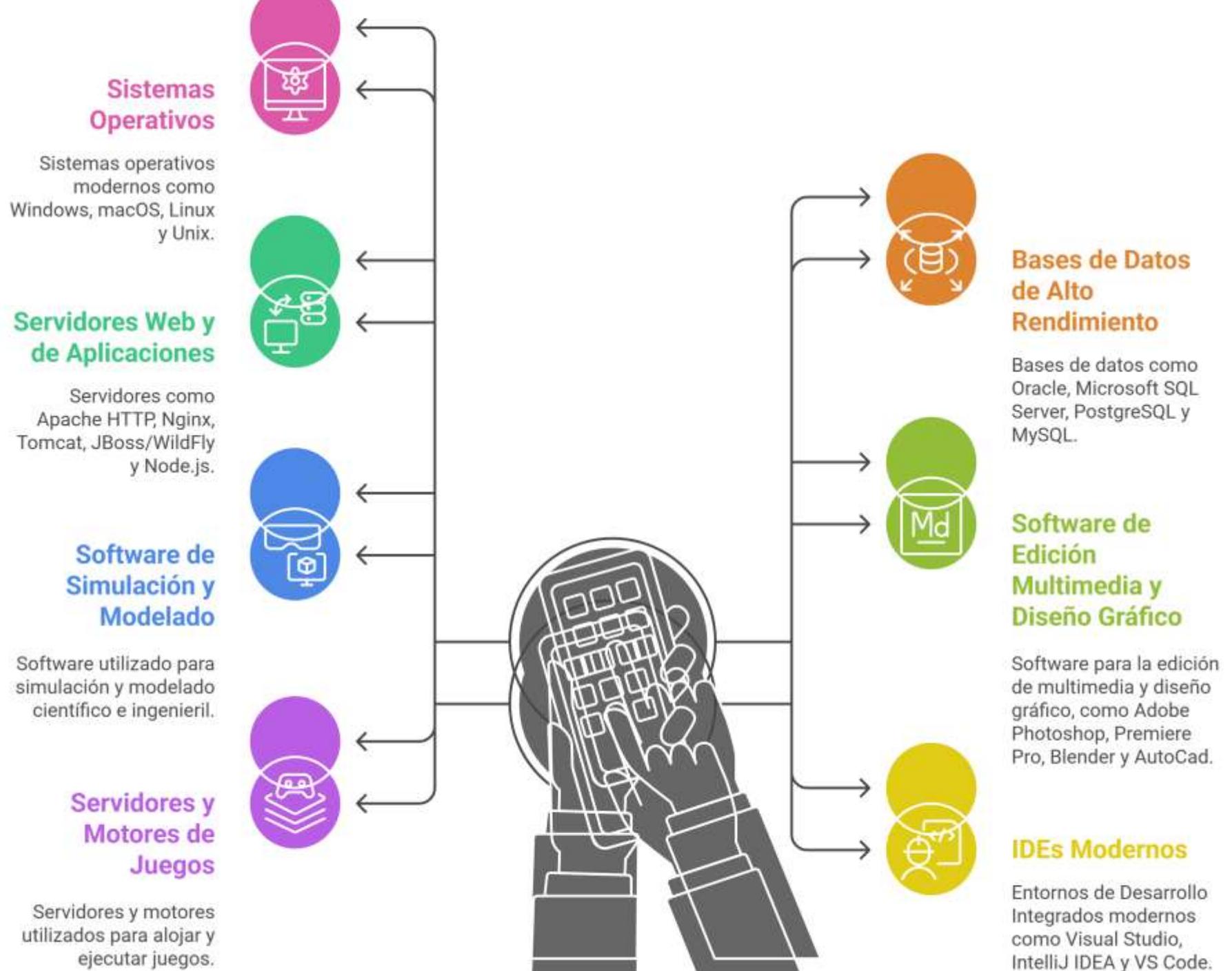
Arquitectura de los Sistemas Distribuidos

» Arquitectura Multiprocesador



44

Ejemplos arquitecturas multiprocesador



Arquitectura de los Sistemas Distribuidos

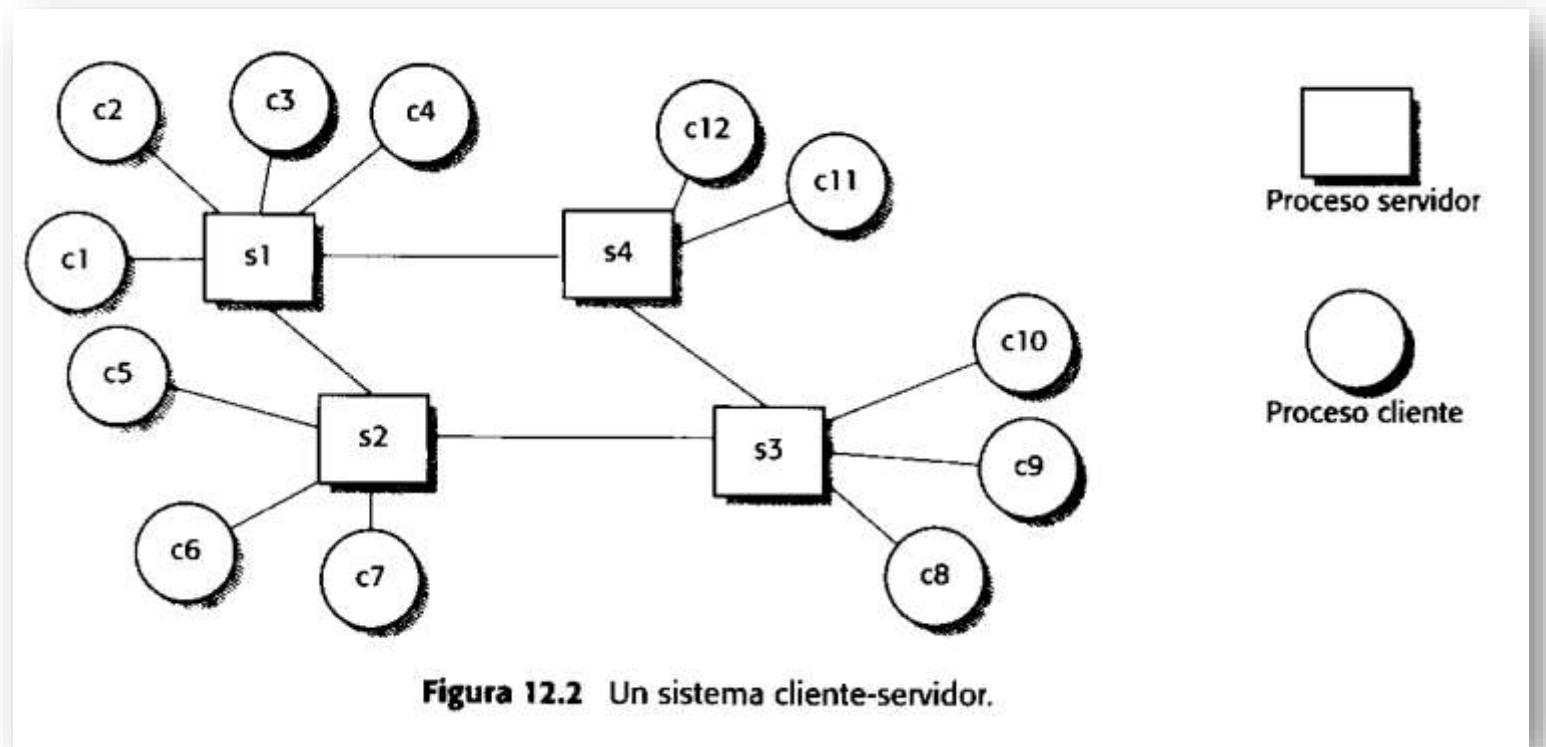
»Arquitectura Cliente-Servidor (C-S)

- ❖ Una aplicación se modela como un conjunto de servicios proporcionado por los servidores y un conjunto de clientes que usan estos servicios
- ❖ Los clientes y servidores son procesos diferentes
- ❖ Los servidores pueden atender varios clientes
- ❖ Un servidor puede brindar varios servicios
- ❖ Los clientes no se conocen entre sí

46

Arquitectura de los Sistemas Distribuidos

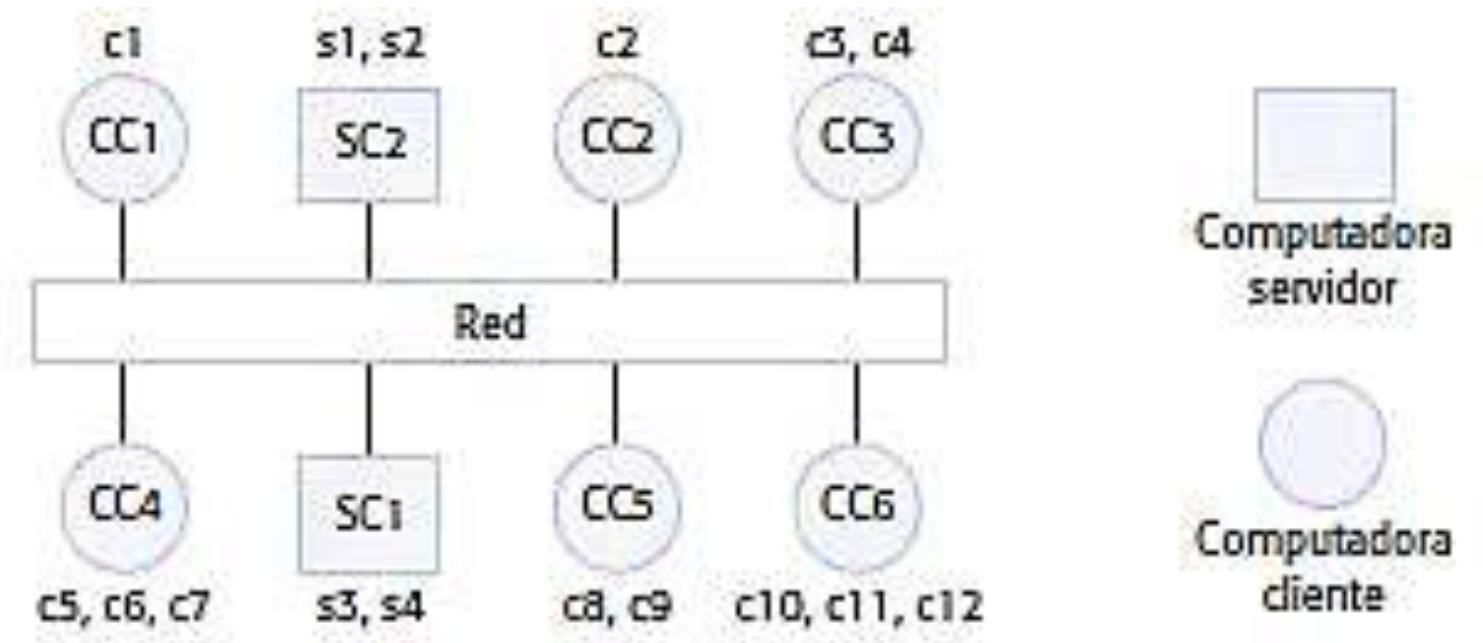
» Arquitectura Cliente-Servidor



47

Arquitectura de los Sistemas Distribuidos

» Arquitectura C-S



48

Arquitectura de los Sistemas Distribuidos

» Clasificación de Arquitectura C-S

Dos Niveles

Cliente ligero

El procesamiento y gestión de datos se lleva a cabo en el servidor

Cliente pesado

El cliente implementa la lógica de la aplicación y el servidor solo gestiona los datos

Multinivel

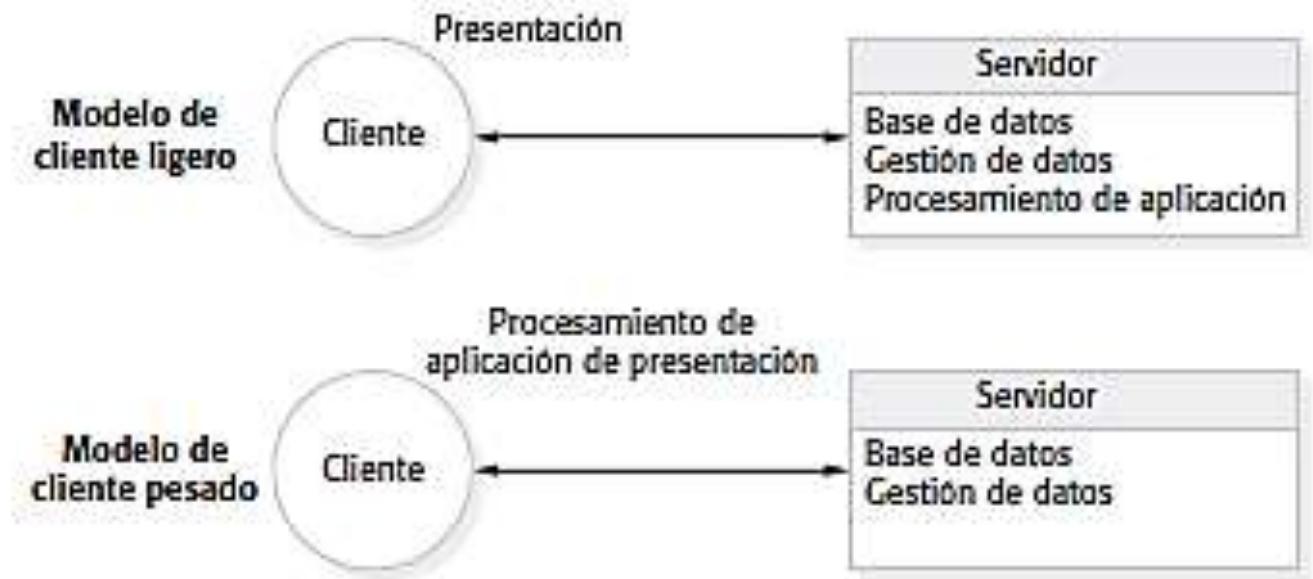
La presentación, el procesamiento y la gestión de los datos son procesos lógicamente separados y se pueden ejecutar en procesadores diferentes

49

Arquitectura de los Sistemas Distribuidos

» Arquitectura C-S

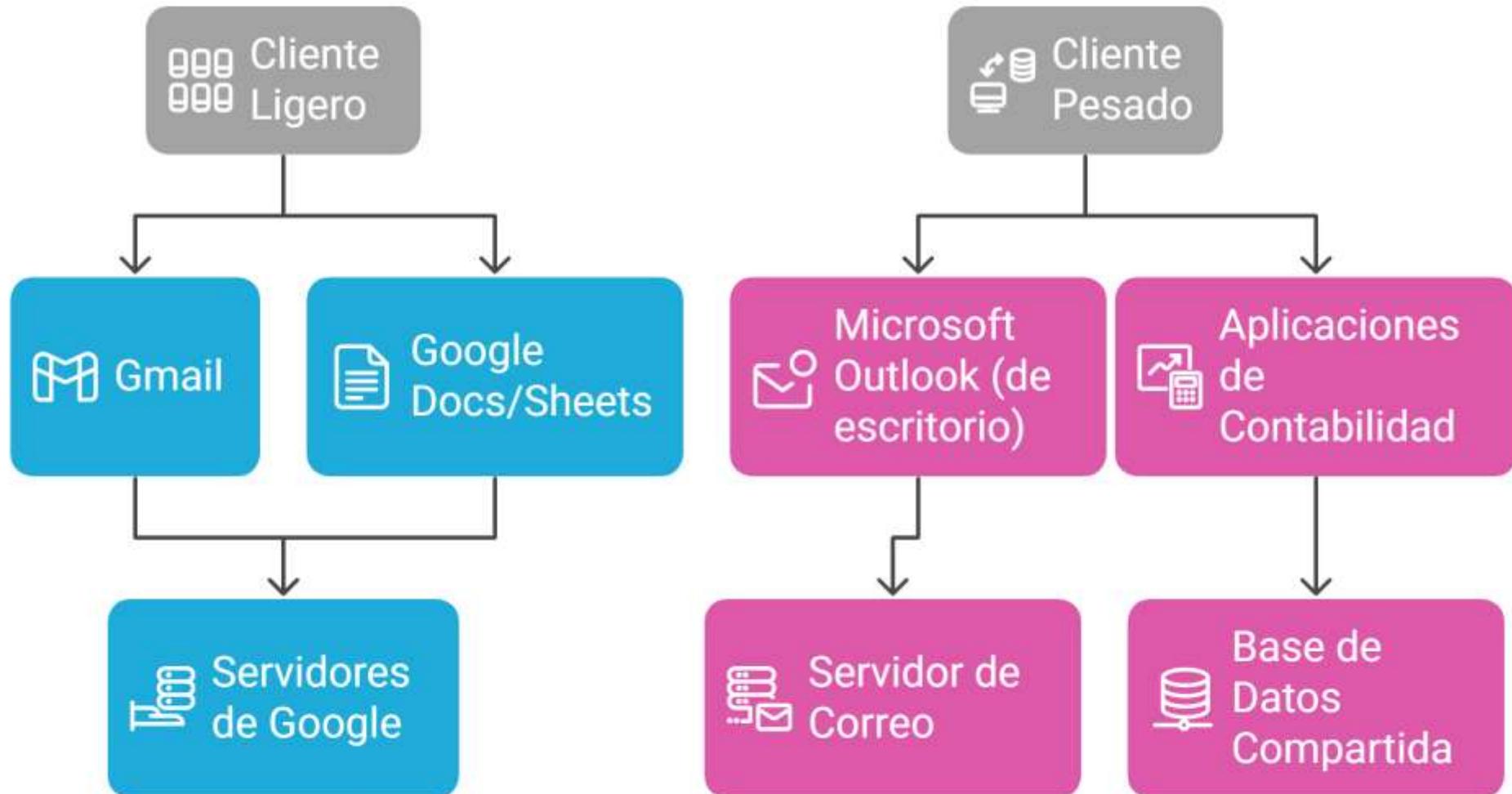
Dos Niveles



50

Arquitectura Cliente-Servidor de Dos Niveles

Ejemplos



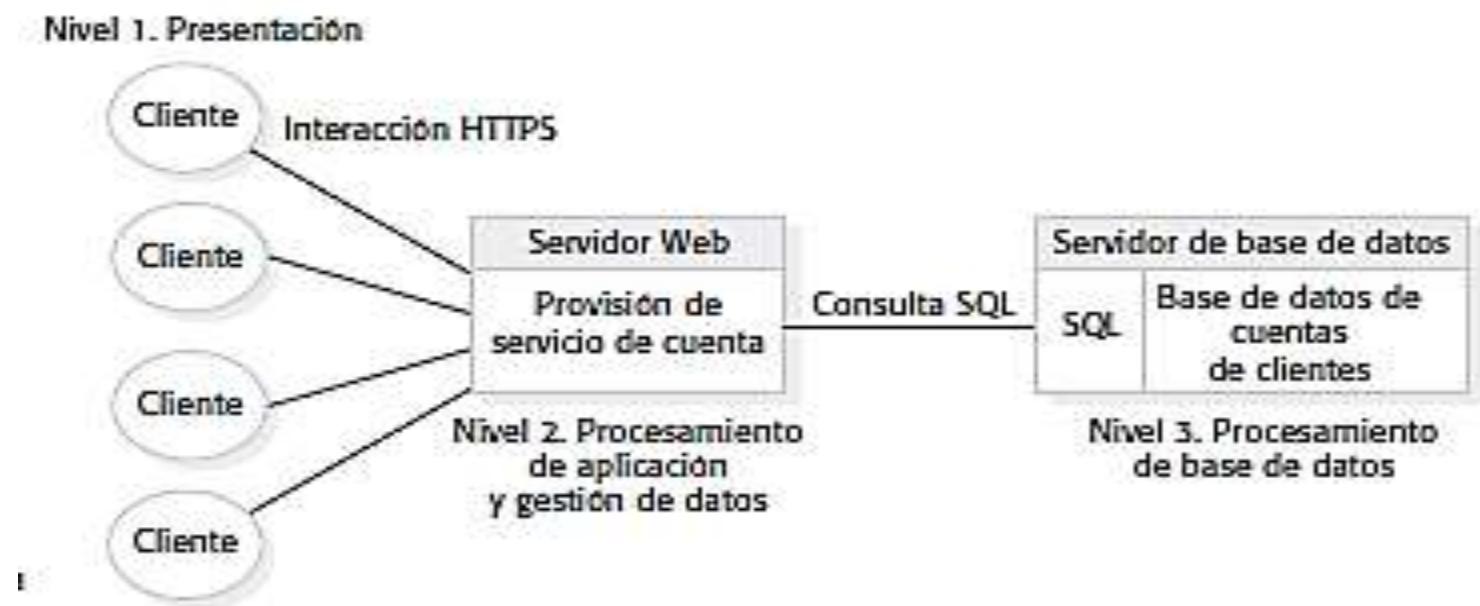
51

Arquitectura de los Sistemas Distribuidos

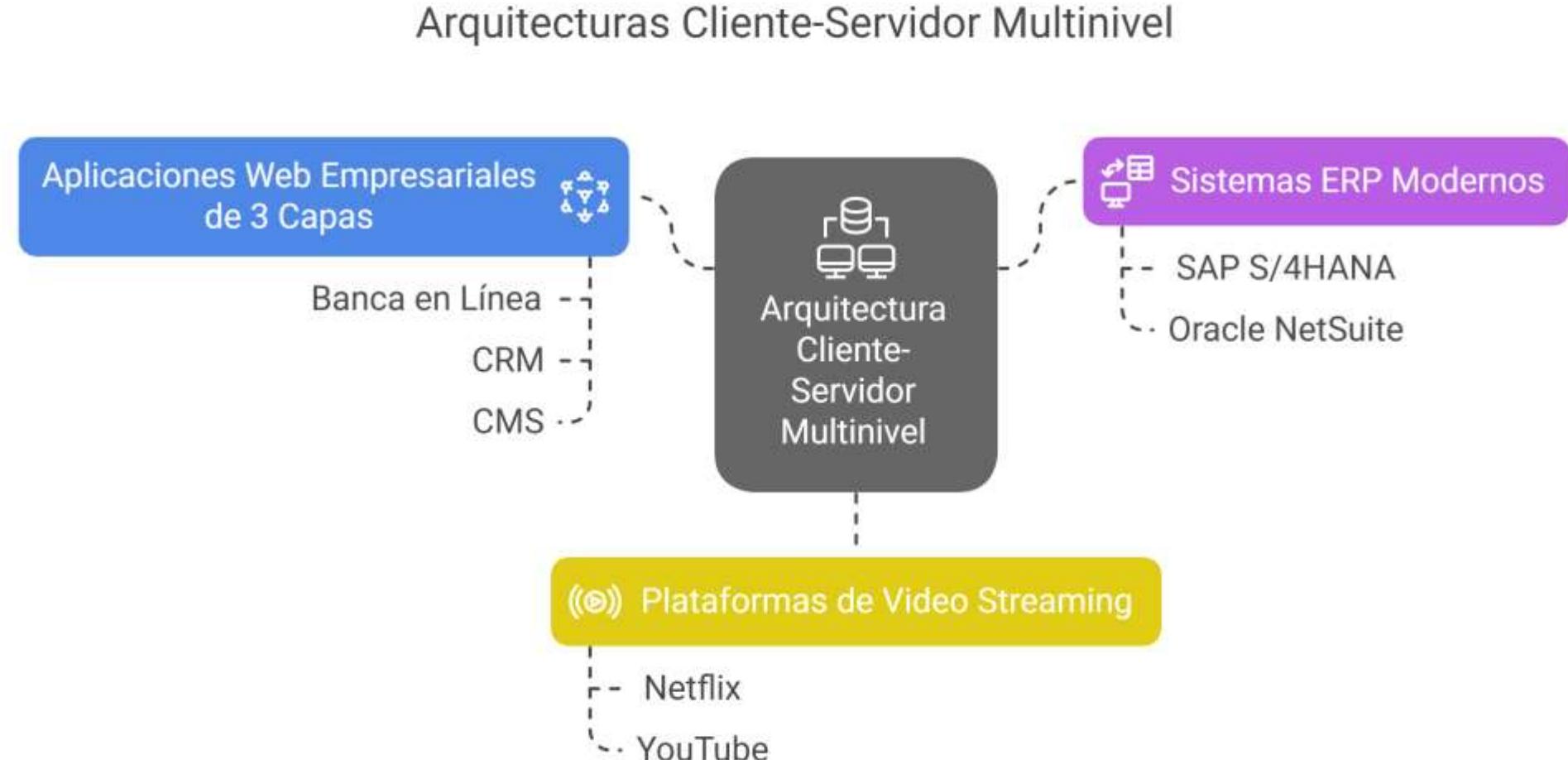
» Arquitectura C-S

Multinivel

52



Ejemplos Arquitectura Cliente – servidor multinivel



Arquitectura de los Sistemas Distribuidos

»Arquitectura de Componentes Distribuidos

Diseña al sistema como un conjunto de componentes u objetos que brindan una interfaz de un conjunto de servicios que ellos suministran. Otros componentes u objetos solicitan estos servicios. No hay distinción tajante entre clientes y servidores.

Los componentes pueden distribuirse en varias máquinas a través de la red utilizando un middleware como intermediario de peticiones

54

Arquitectura de los Sistemas Distribuidos

» Arquitectura de Objetos Distribuidos



El **middleware** es un software que actúa como una **capa intermedia** entre las aplicaciones y los sistemas operativos o las redes subyacentes. Su función principal es **facilitar la comunicación y la gestión** de los componentes distribuidos, permitiendo que diferentes partes de un sistema interactúen sin tener que conocer los detalles complejos de la infraestructura de red o del sistema operativo de cada una.

55

Ejemplos de arquitectura de objetos distribuidos

CORBA son las siglas de **Common Object Request Broker Architecture** (Arquitectura de Agente de Petición de Objetos Común). Es un estándar definido por el Object Management Group (OMG) diseñado para permitir que componentes de software escritos en diferentes lenguajes de programación y ejecutándose en diferentes máquinas (y sistemas operativos) puedan **interoperar** y trabajar juntos como si fueran objetos locales.

Sistemas Empresariales CORBA

Sistemas legados en telecomunicaciones y finanzas

Aplicaciones Java RMI

Aplicaciones Java que utilizan RMI para la comunicación

Arquitecturas de Objetos Distribuidos

RMI son las siglas de **Remote Method Invocation** (Invocación de Método Remoto).

Es una API (Interfaz de Programación de Aplicaciones) nativa del lenguaje de programación **Java** que permite que un objeto que se ejecuta en una Máquina Virtual Java (JVM) pueda **invocar métodos** de un objeto que se está ejecutando en otra **JVM diferente**. Estas JVMs pueden estar en la misma máquina física o en máquinas separadas conectadas por una red.

DCOM son las siglas de **Distributed Component Object Model** (Modelo de Objetos Componentes Distribuidos). Es una tecnología desarrollada por **Microsoft** que permite que componentes de software (objetos COM) se comuniquen directamente a través de una red. Es, esencialmente, una extensión del **Component Object Model (COM)** de Microsoft para soportar la comunicación distribuida.

Arquitectura de los Sistemas Distribuidos

»Computación Distribuida inter-organizacional

Una organización tiene varios servidores y reparte su carga computacional entre ellos.

Extender este concepto a varias organizaciones.

57

Pueden ser arquitecturas del tipo:

- Peer-to-Peer*
- Orientados a servicios*

Arquitectura de los Sistemas Distribuidos

»Computación Distribuida inter-organizacional

Arquitecturas Peer-to-Peer (P2P)

Sistemas descentralizados en los que el cálculo puede llevarse a cabo en cualquier nodo de la red

Se diseñan para aprovechar la ventaja de la potencia computacional y el almacenamiento a través de una red

Pueden utilizar una arquitectura

Descentralizada

donde cada nodo rutea los paquetes a sus vecinos hasta encontrar el destino

Semi-centralizada

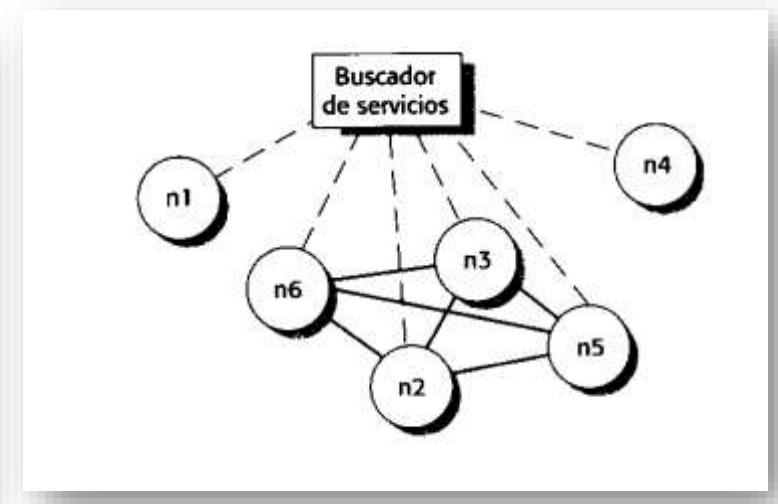
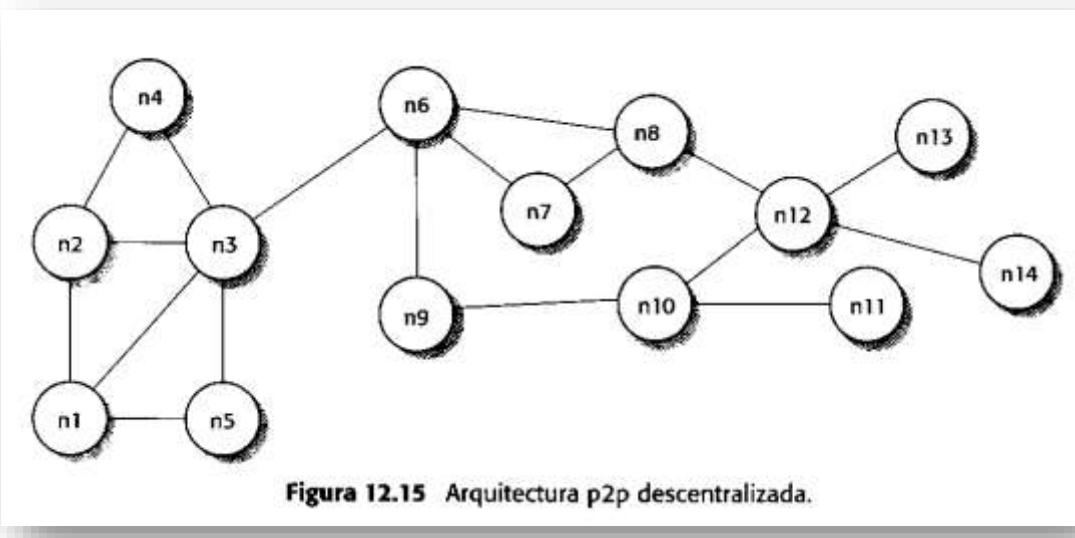
donde un servidor ayuda a conectarse a los nodos o coordinar resultados

58

Arquitectura de los Sistemas Distribuidos

» Computación Distribuida inter-organizacional
Arquitecturas Peer-to-Peer (P2P)

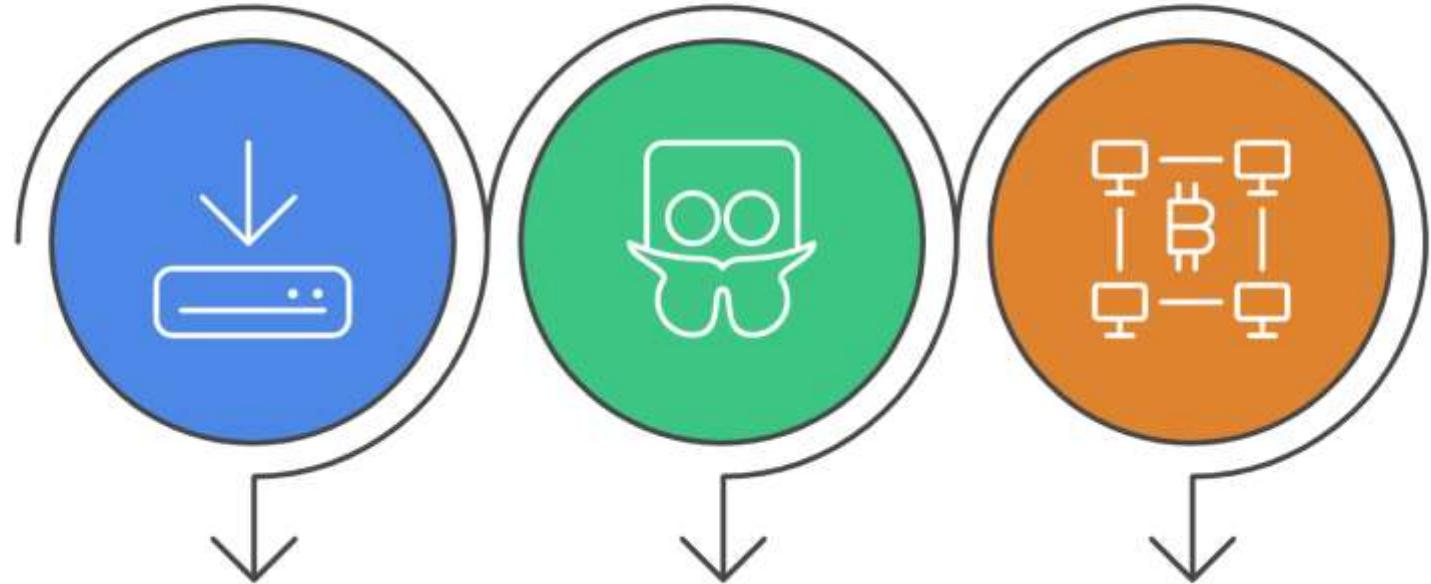
59



Ejemplos de Software P2P

Ejemplos de P2P

60



Clients de BitTorrent

Software que implementa el protocolo BitTorrent

Skype

Aplicación que utiliza P2P para llamadas directas

Sistemas de Criptomonedas

Redes que operan con software P2P

Arquitectura de los Sistemas Distribuidos

» Computación Distribuida inter-organizacional Arquitectura de sistemas orientadas a servicios

Servicio

- ❖ Representación de un recurso computacional o de información que puede ser utilizado por otros programas.
- ❖ Un servicio es independiente de la aplicación que lo utiliza
- ❖ Un servicio puede ser utilizado por varias organizaciones
- ❖ Una aplicación puede construirse enlazando servicios
- ❖ Las arquitecturas de las aplicaciones de servicios web son arquitecturas débilmente acopladas

61

Arquitectura de los Sistemas Distribuidos

» Computación Distribuida inter-organizacional

Arquitectura de sistemas orientadas a servicios

Funcionamiento

- ❖ Un proveedor de servicios oferta servicios definiendo su interfaz y su funcionalidad
- ❖ Para que el servicio sea externo, el proveedor publica el servicio en un “registro de servicio” con información del mismo
- ❖ Un solicitante enlaza este servicio a su aplicación, es decir
 - ❖ que el solicitante incluye el
 - ❖ código para invocarlo y procesa el
 - ❖ resultado del mismo

62



Ejemplos de Computación Distribuida inter-organizacional

Arquitectura de sistemas orientadas a servicios

Tipo de Integración	Ejemplos
Integración de Aplicaciones Empresariales (EAI) y Buses de Servicios Empresariales (ESB) 	IBM Integration Bus, Oracle Service Bus
Aplicaciones Empresariales Modulares y de Gran Escala 	SAP S/4HANA, Oracle Fusion Applications
Servicios Web Públicos y APIs de Terceros 	PayPal, Google Maps Platform
Plataformas de Software como Servicio (SaaS) que exponen APIs 	Dropbox, Google Drive
Sistemas de Gobierno Electrónico y Servicios Públicos Digitales 	Plataformas para la interacción ciudadano-gobierno



Ingeniería de software II

Codificación

Codificación

- » Una vez establecido el diseño, se deben escribir los programas que implementen dicho diseño.
- » Esto puede resultar una tarea compleja por distintos motivos:
 - Los diseñadores pueden no haber tenido en cuenta las particularidades de la plataforma y el ambiente de programación.
Las estructuras e interrelaciones que son fáciles de describir mediante diagramas, no siempre resultan sencillas de escribir en código.
Es indispensable escribir el código de forma que resulte comprendible para otras personas.
Se deben sacar beneficios de las características de diseño creando código que sea reutilizable.

65

Codificación: Pautas Generales

»Resultan útiles para conservar la calidad del diseño en la codificación:

Localización de entrada y salida: es deseable localizarlas en componentes separados del resto del código ya que generalmente son más difíciles de probar.

Inclusión de pseudocódigo: Es útil avanzar el diseño, realizando un pseudocódigo para adaptar el diseño al lenguaje elegido.

Revisión y reescritura, no a los remiendos: Es recomendable realizar un borrador, revisarlo y reescribirlo tantas veces como sea necesario.

Reutilización: Hay dos clases de reutilización:

Productiva: *se crean componentes destinados a ser reutilizados por otra aplicación*

Consumidora: *Se usan componentes originalmente desarrollados para otros proyectos.*

Codificación: Documentación

» Se considera como Documentación del programa al conjunto de descripciones escritas que explican al lector qué hace el programa y cómo lo hace.

» Se divide en:

Documentación interna: Es concisa, escrita en un nivel apropiado para un programador. Contiene información dirigida a quienes leerán el código fuente. Incluye información de algoritmos, estructuras de control, flujos de control.

Documentación externa: Se prepara para ser leída por quienes, tal vez, nunca verán el código real. Por ejemplo, los diseñadores, cuando evalúan modificaciones o mejoras.



Gestión de Proyectos

Métricas

Elementos clave de la gestión de proyectos

- » Métricas
 - » Estimaciones
 - » Calendario temporal
 - » Organización del personal
 - » Análisis de riesgos
 - » Seguimiento y control
- } Tema de la clase de hoy

Métricas

Clave tecnológica

Objetivos fundamentales:

- ❖
- ❖
- ❖
- ❖

**Entender
Controlar
Mejorar
Evaluar**



Métricas – Definiciones

Medida



Medición



Métrica

LOC por punto de revisión			
Desarrollador	LLOC-EPI	Desarrollador	LLOC-EPI
Diego Martínez	325	Eduardo Alvarado Gómez Túroso	64
Diego Martínez	212	José María	53
Diego Martínez	155	Manuel Cárdenas	38
Fernando	109	Edgardo Gómez	34
Fernando	108	Wilson Elizalde	32
Fernando	84	Cristina	28
Fernando	81	Cristina	27
Fernando	71	Diego	26
Fernando	61	Diego Gómez	20
Fernando	30	El Agente	13
Fernando	20	Florver Sánchez	10
Fernando	11	Hector Arce Gómez	8

Indicador

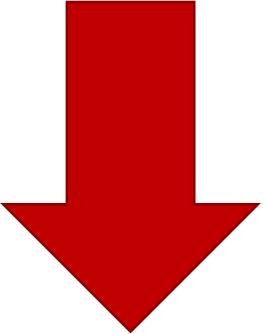


Métrica -definición

- » A diferencia de una simple medida, una métrica a menudo implica un cálculo o una fórmula que relaciona varias medidas para proporcionar una comprensión más profunda

Métricas

»Las métricas pueden ser utilizadas para que los profesionales e investigadores puedan tomar las mejores decisiones



Métricas como medio para asegurar la calidad en
Procesos/Proyectos Software/Productos

Métricas del proyecto

Propósitos tácticos

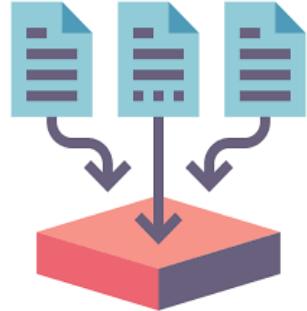
Uso

- ❖ Ajustes en el calendario y evitar demoras
- ❖ Valorar el estado de un proyecto en marcha
- ❖ Rastrear riesgos
- ❖ Descubrir áreas de problemas
- ❖ Ajustar flujo de trabajo/tareas
- ❖ Evaluar habilidad del equipo.



Métricas del proceso

» Propósitos estratégicos



Recopilación

a través de todos los proyectos y por un espacio de tiempo.



Intención

proporcionar un conjunto de indicadores para mejorar el proceso.

Métricas del proceso

Uso

- ❖ Sentido común y sensibilidad organizacional.
- ❖ Retroalimentación.
- ❖ No usar métricas para valorar a los individuos.
- ❖ Establecer metas y métricas claras.
- ❖ No excluir métricas.



Métricas del producto

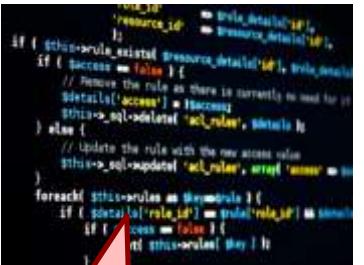
Cómo los
medimos



Dinámicas

- ❖ Hechas en tiempo real sobre un programa en ejecución.
- ❖ Ayudan a valorar la eficiencia y el rendimiento de un programa.

Se relacionan con las características de calidad del software



Estáticas

- ❖ Hechas una vez que se ha escrito el código.

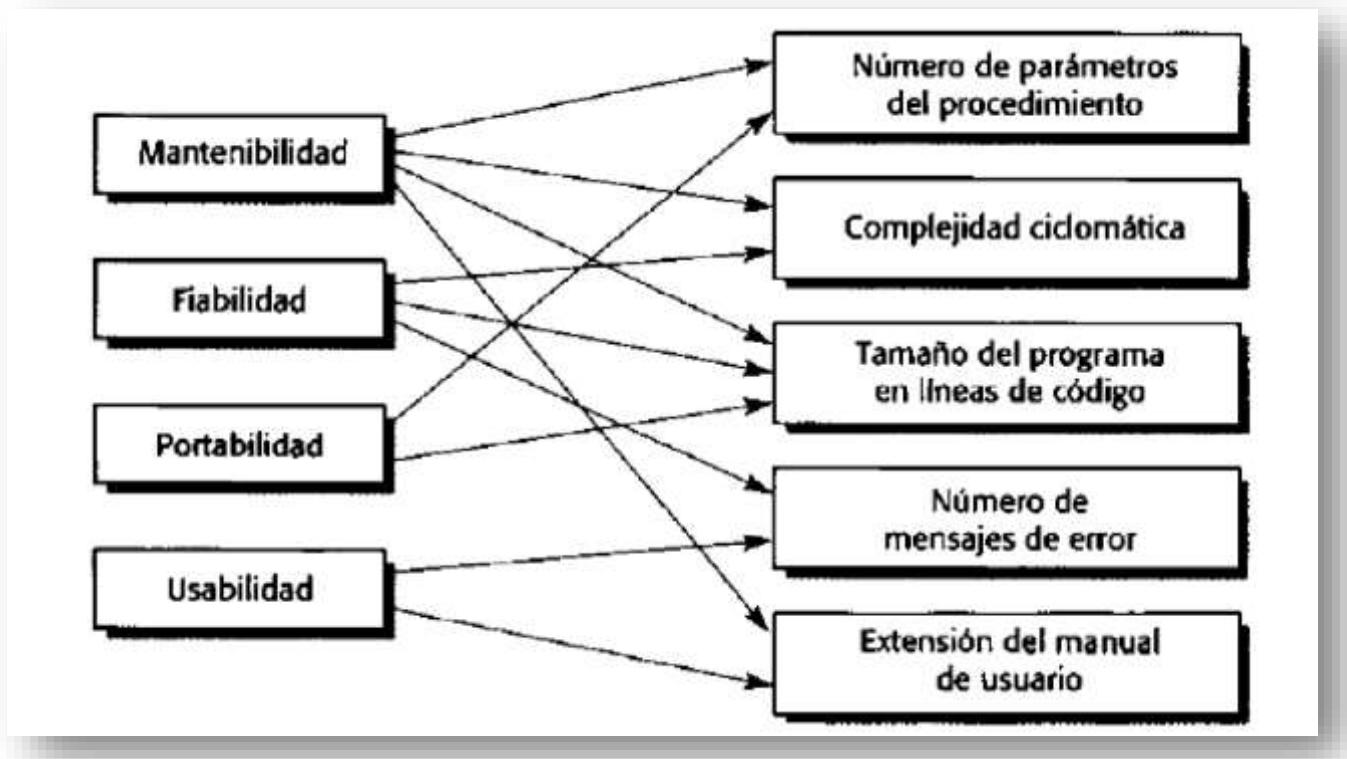
Representaciones del código fuente.

d.

Se relacionan de manera indirecta con los atributos de calidad del software

Métricas del producto

Atributos de calidad externos vs Atributos internos



Métricas estáticas del producto

Fan-in/Fan-out	Fan-in es una medida del número de funciones o métodos que llaman a otra función o método (por ejemplo, X). Fan-out es el número de funciones que son llamadas por una función X. Un valor alto de fan significa que X está fuertemente acoplada al resto del diseño y que los cambios en X tendrán muchos efectos importantes. Un valor alto de fan-out sugiere que la complejidad de X podría ser alta debido a la complejidad de la lógica de control necesaria para coordinar los componentes llamados.
Longitud del código	Ésta es una medida del tamaño del programa. Generalmente, cuanto más grande sea el tamaño del código de un componente, más complejo y susceptible de errores será el componente. La longitud del código ha mostrado ser la métrica más fiable para predecir errores en los componentes.
Complejidad ciclomática	Ésta es una medida de la complejidad del control de un programa. Esta complejidad del control está relacionada con la comprensión del programa. El cálculo de la complejidad ciclomática se trata en el Capítulo 22.
Longitud de los identificadores	Es una medida del promedio de los diferentes identificadores en un programa. Cuanto más grande sea la longitud de los identificadores, más probable será que tengan significado; por lo tanto, el programa será más comprensible.
Profundidad del anidamiento de las condicionales	Ésta es una medida de la profundidad de anidamiento de las instrucciones condicionales «if» en un programa. Muchas condiciones anidadas son difíciles de comprender y son potencialmente susceptibles de errores.
Índice de Fog	Ésta es una medida de la longitud promedio de las palabras y las frases en los documentos. Cuanto más grande sea el Índice de Fog, el documento será más difícil de comprender.

Métricas estáticas del producto

Métricas OO

Métrica orientada a objetos	Descripción
Métodos ponderados por clase (<i>weighted methods per class</i> , WMC)	Este es el número de métodos en cada clase, ponderado por la complejidad de cada método. Por lo tanto, un método simple puede tener una complejidad de 1, y un método grande y complejo tendrá un valor mucho mayor. Cuanto más grande sea el valor para esta métrica, más compleja será la clase de objeto. Es más probable que los objetos complejos sean más difíciles de entender. Tal vez no sean lógicamente cohesivos, por lo que no pueden reutilizarse de manera efectiva como superclases en un árbol de herencia.
Profundidad de árbol de herencia (<i>depth of inheritance tree</i> , DIT)	Esto representa el número de niveles discretos en el árbol de herencia en que las subclases heredan atributos y operaciones (métodos) de las superclases. Cuanto más profundo sea el árbol de herencia, más complejo será el diseño. Es posible que tengan que comprenderse muchas clases de objetos para entender las clases de objetos en las hojas del árbol.
Número de hijos (<i>number of children</i> , NOC)	Ésta es una medida del número de subclases inmediatas en una clase. Mide la amplitud de una jerarquía de clase, mientras que DIT mide su profundidad. Un valor alto de NOC puede indicar mayor reutilización. Podría significar que debe realizarse más esfuerzo para validar las clases base, debido al número de subclases que dependen de ellas.
Acoplamiento entre clases de objetos (<i>coupling between object classes</i> , CBO)	Las clases están acopladas cuando los métodos en una clase usan los métodos o variables de instancia definidos en una clase diferente. CBO es una medida de cuánto acoplamiento existe. Un valor alto para CBO significa que las clases son estrechamente dependientes y, por lo tanto, es más probable que el hecho de cambiar una clase afecte a otras clases en el programa.
Respuesta por clase (<i>response for a class</i> , RFC)	RFC es una medida del número de métodos que potencialmente podrían ejecutarse en respuesta a un mensaje recibido por un objeto de dicha clase. Nuevamente, RFC se relaciona con la complejidad. Cuanto más alto sea el valor para RFC, más compleja será una clase y, por ende, es más probable que incluya errores.
Falta de cohesión en métodos (<i>lack of cohesion in methods</i> , LCOM)	LCOM se calcula al considerar pares de métodos en una clase. LCOM es la diferencia entre el número de pares de método sin compartir atributos y el número de pares de método con atributos compartidos. El valor de esta métrica se debate ampliamente y existe en muchas variaciones. No es claro si realmente agrega alguna información útil además de la proporcionada por otras métricas.

Métricas del producto -

LDC – LÍNEAS DE CÓDIGO



La métrica más común para el tamaño de un producto

postmorten



Medida discutida !!

Resultados



¿Qué tiempo?



¿Cuántas personas?



Si una organización de software mantiene registros sencillos, se puede crear una tabla de datos orientados al tamaño

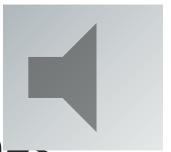
Utilidad de las métricas postmortem

- ❖ Conformar una línea base para futuras métricas
- ❖ Ayudar al mantenimiento conociendo la complejidad lógica, tamaño, flujo de información, identificando módulos críticos
- ❖ Ayudar en los procesos de reingeniería



Métricas orientadas al tamaño

- ❖ Se puede obtener :



KLDC (miles de líneas de código)
LDC - LÍNEAS DE CÓDIGO

Productividad: relación entre KLDC / Persona mes

Calidad: relación entre Errores / KLDC

Costo: relación entre \$ / KLDC



Como manejo las líneas en blanco, comentarios , etc

Propuesta Fenton/Pfleeger

- Medir : CLOC = **Cantidad de líneas de comentarios**
- Luego:
 - long total (LOC) = NCLOC + CLOC
- Surgen medidas indirectas:
 - CLOC/LOC mide la densidad de comentarios

Ejemplo

Productividad = KLDC/persona-mes
Calidad = errores/KLDC
Documentación = págs.. Doc./ KLDC
Costo = \$/KLDC

- ❖ Calcular, usando **LDC**, la productividad, calidad y costo para los cuatro proyectos de los cuales se proporcionan los datos.

Proyecto	LDC	U\$S	Errores	Personas-mes	Errores/KLDC	U\$S/KLDC	KLDC/Personas-mes
P1	25.500	15000	567	15	22,23	588,23	1,7
P2	19.100	7200	210	10	10,99	376,96	1,91
P3	10.700	6000	100	20	9,34	560,74	0,53
P4	100.000	18000	2200	30	22	180	3,33

- ¿Cuál es el proyecto de **mayor calidad** (errores/KLDC)?
- ¿Cuál es el proyecto de **mayor costo por línea** (\$/KLDC)?
- ¿Cuál es el proyecto de **menor productividad por persona** (KLDC/personas-mes)?
-

Métrica de

Punto función

Mide la cantidad de funcionalidad de un sistema descripto en una especificación

PF- Punto función (Albrecht 1978)

Factor de Ponderación, es subjetivo y está dado por la organización/equipo

$$PF = TOTAL * [0.65 + 0.01 * \sum(F_i)] \quad i=1 \text{ a } 14 \quad 0 \leq F_i \leq 5$$

Entradas

Salidas

Consultas

Almacenamientos internos

Interfaces externas

simple|medio|complejo

$$* [3 | 4 | 6] = \dots$$

$$* [4 | 5 | 7] = \dots$$

$$* [3 | 4 | 6] = \dots$$

$$* [7 | 10 | 15] = \dots$$

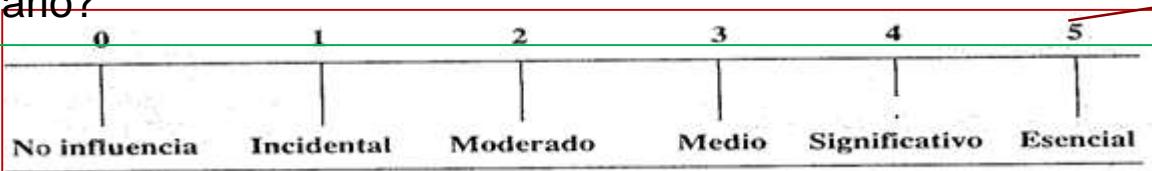
$$* [5 | 7 | 10] = \dots$$

TOTAL

Son valores de ajuste de la complejidad según las preguntas de la siguiente pantalla

Métrica de Punto función F(i)

1. ¿Requiere el sistema copias de seguridad y de recuperación fiables?
2. ¿Se requiere comunicación de datos?
3. ¿Existen funciones de procesamiento distribuido?
4. ¿Es crítico el rendimiento?
5. ¿Se ejecuta el sistema en un entorno operativo existente y fuertemente utilizado?
6. ¿Requiere el sistema entrada de datos interactiva?
7. ¿Requiere la entrada de datos interactiva que las transacciones de entrada se lleven a cabo sobre múltiples pantallas u operaciones?
8. ¿Se actualizan los archivos maestros de forma interactiva?
9. ¿Son complejas las entradas, las salidas, los archivos o las peticiones?
10. ¿Es complejo el procesamiento interno?
11. ¿Se ha diseñado el código para ser reutilizable?
12. ¿Están incluidas en el diseño la conversión y la instalación?
13. ¿Se ha diseñado el sistema para soportar múltiples instalaciones en diferentes organizaciones?
14. ¿Se ha diseñado la aplicación para facilitar los cambios y para ser fácilmente utilizada por el usuario?



Cada una de las preguntas se contesta de acuerdo a la siguiente escala de valores

Métrica de

Punto función

❖ Métricas derivadas:

Productividad: relación entre PF y Persona_mes

Calidad: relación entre Errores y PF

Costo: relación entre \$ y PF

Productividad = PF / Persona_mes

Calidad = Errores / PF

Costo = \$ / PF

Medida subjetiva ***independiente del lenguaje***, de estimación más fácil.

Métrica temprana

Desarrollo de una métrica- GQM

- ❖ Victor Basili desarrolló un método llamado **GQM** (Goal, Question, Metric) (o en castellano: OPM Objetivo, Pregunta, Métrica).
- ❖ (<ftp://ftp.cs.umd.edu/pub/sel/papers/gqm.pdf>)
- ❖ Dicho método esta orientado a lograr una métrica que “mida” cierto objetivo. El mismo nos permite mejorar la calidad de nuestro proyecto.



GQM (OPM)

Estructura GQM

Objetivo

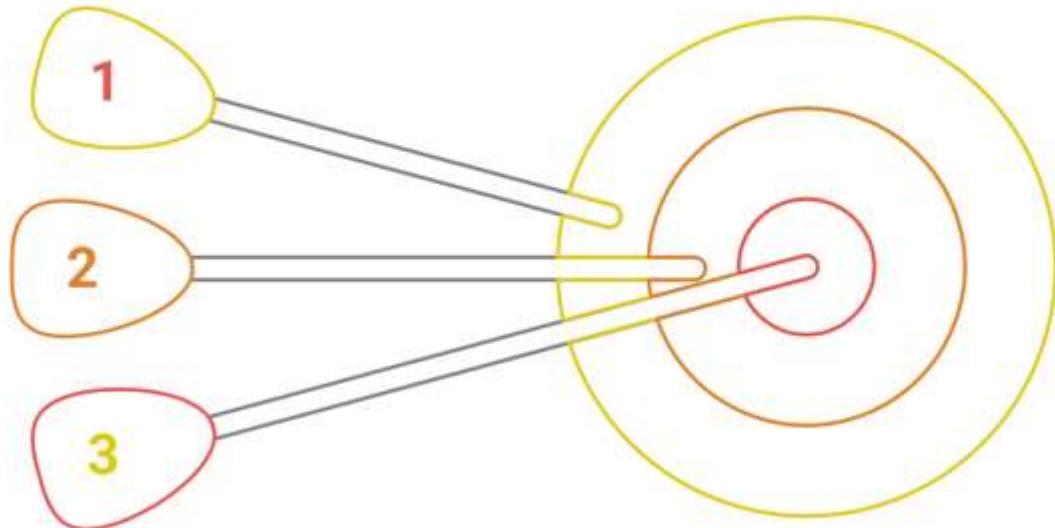
El objetivo central del proyecto

Preguntas

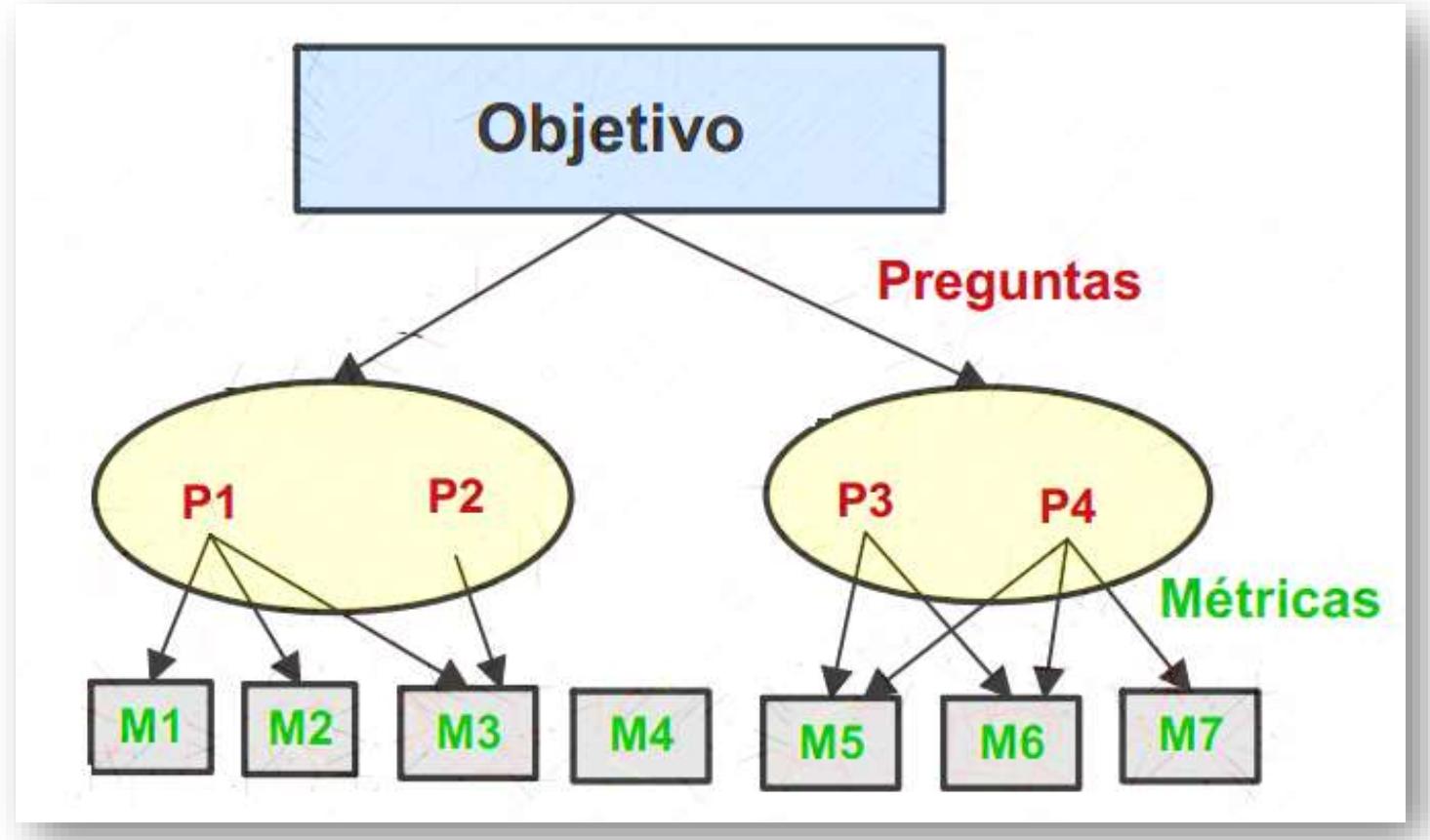
Preguntas para verificar el cumplimiento del objetivo

Métricas

Medidas cuantitativas para responder a las preguntas



GQM (OPM)



GQM Ejemplo

- ❖ Evaluamos, en la etapa de Análisis de Requerimientos, la tarea Asignación de responsabilidades.

Propósito	Evaluar	
Característica	Asignación de responsabilidades	
Punto de Vista	Gerencia de Proyecto	
Pregunta 1	¿Existe un proceso para la asignación de roles?	
	M1	Valor Binario
Pregunta 2	¿Hay un responsable de asignar roles?	
	M2	Valor Binario
Pregunta 3	¿El responsable siempre realiza su tarea?	
	M3	Valor Binario
Pregunta 4	¿Existe información anterior sobre las tareas realizadas por cada integrante?	
	M4	Valor Binario
Pregunta 5	¿Esa información está disponible?	
	M5	Valor Binario

Indicadores

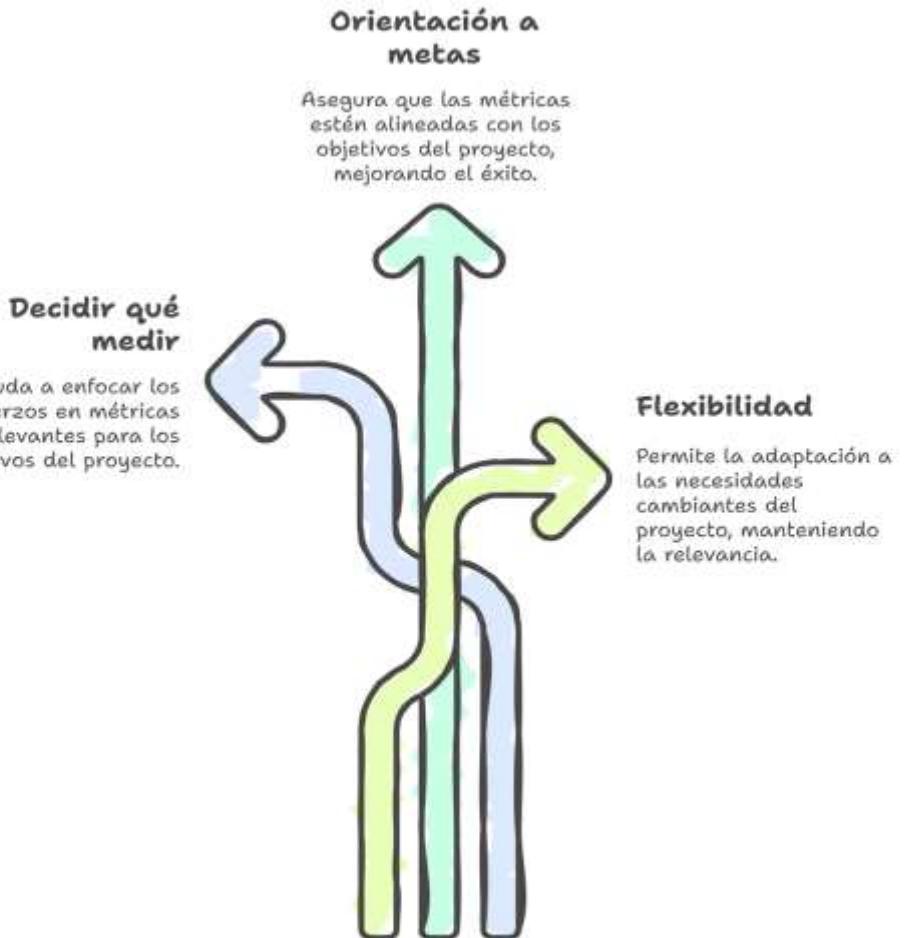
<u>Nombre</u>	<u>Descripción</u>	Fórmula
I1	Gestión de Asignación de roles	M2 & M3 & M4 & M5
I2	Proceso de Asignación de roles	M1 & M4 & M5

A partir de los indicadores definidos, se propone realizar el control de la meta a través de un tablero de control de indicadores específicos. Podemos decir que nuestra meta se cumple si los indicadores muestran los siguientes valores:

I1	Gestión de Asignación de roles	Verdadero
I2	Proceso de Asignación de roles	Verdadero

GQM

¿Cómo implementar el marco GQM?



Made with  Napkin



Gestión de Proyectos

Estimaciones

Estimaciones



¿Qué son?



¿Cómo usarlas?

La estimación del software es el proceso de predecir la cantidad más realista de esfuerzo (expresado comúnmente en horas persona o costo monetario) requerido para desarrollar o mantener software.

29

Estimaciones

Las métricas y la estimación de software se enfocan en la perspectiva temporal.



30

Made with  Napkin

Estimaciones



❖ ¿Qué podemos estimar?

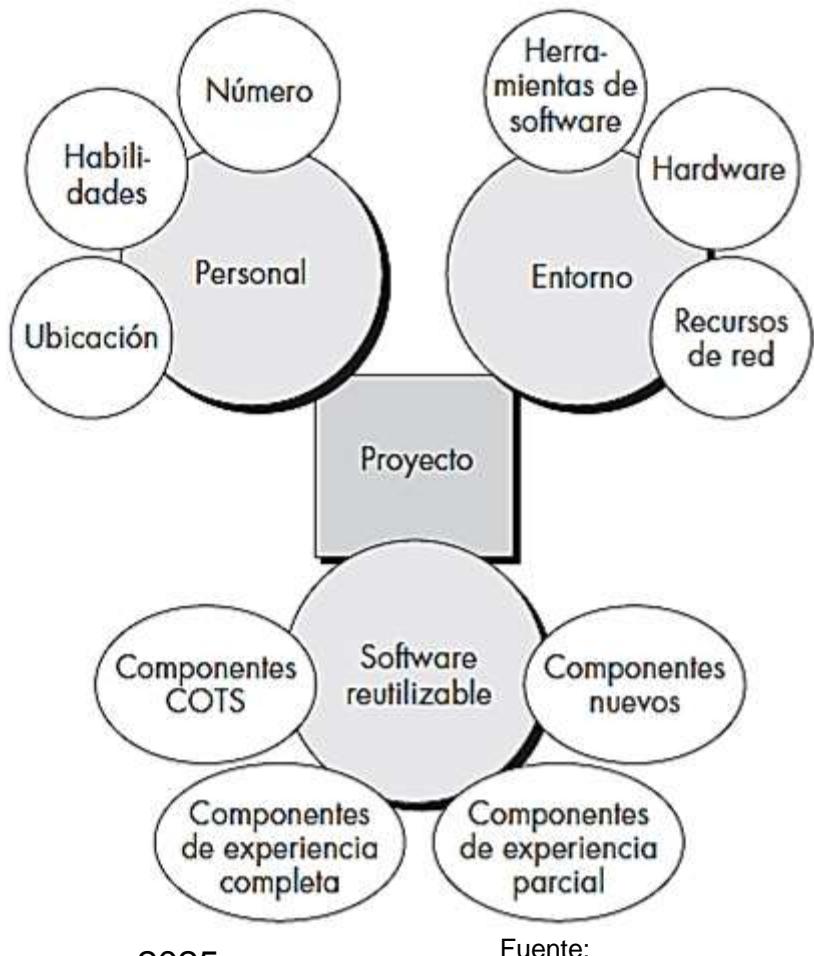


- ❖ ¿Qué tener en cuenta?
- ❖ ¿Qué factores influyen?

31

Estimaciones de recursos

32



2025

Fuente:

Estimaciones de costos

3 parámetros para calcular costo de un proyecto



33

Fijación de precio - Relación Precio Costo

❖ ¿Qué tener en cuenta ?

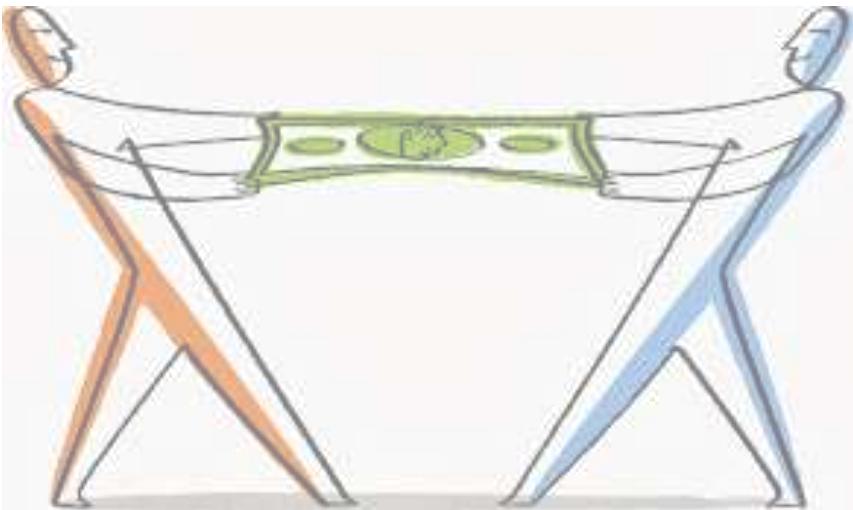
Oportunidad de mercado	Una organización de desarrollo podría ofrecer un bajo precio debido a que desea conseguir cuota de mercado. Aceptar un beneficio bajo en un proyecto podría darle la oportunidad de obtener más beneficios posteriormente. La experiencia obtenida le permite desarrollar nuevos productos.
Incertidumbre en la estimación de costes	Si una organización está insegura de su coste estimado, puede incrementar su precio por encima del beneficio normal para cubrir alguna contingencia.
Términos contractuales	Un cliente puede estar dispuesto a permitir que el desarrollador retenga la propiedad del código fuente y que reutilice el código en otros proyectos. Por lo tanto, el precio podría ser menor que si el código fuente del software se le entregara al cliente.
Volatilidad de los requerimientos	Si es probable que los requerimientos cambien, una organización puede reducir los precios para ganar un contrato. Después de que el contrato se le asigne, se cargarán precios altos a los cambios en los requerimientos.
Salud financiera	Los desarrolladores en dificultades financieras podrían bajar sus precios para obtener un contrato. Es mejor tener beneficios más bajos de los normales o incluso quebrar antes de quedar fuera de los negocios.

34

Fijación de precio

- ❖ Debe pensarse en :
- ❖ los intereses de la empresa,
- ❖ los riesgos,
- ❖ el tipo de contrato.

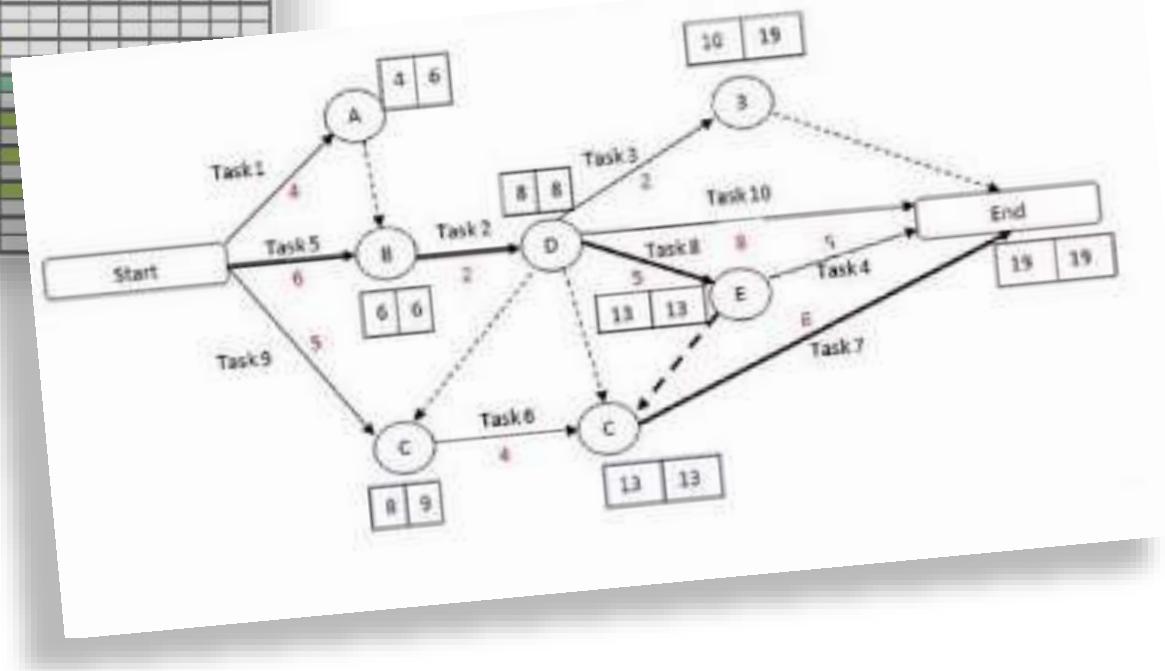
Esto puede hacer que el precio suba o baje.



35

Estimaciones de tiempo

Carta Gantt (Control de las Actividades Proyecto Servicio)														
Nombre del Equipo: Los Triunfadores		Curso: 2º Medio "X"					Año: 2008							
Nº	Meses - Semanas	Agosto	Septiembre	Octubre	Noviembre	Diciembre	1	2	3	4	1	2	3	4
1	Redacción del Proyecto													
2	Diseño del Servicio													
3	Cotización de Materiales													
4	Compra de materiales													
5	Presentación del Proyecto													
6	Promoción del Servicio													
7	Prestación del Servicio													
8	Evaluación de Proceso													
9	Control de Calidad													



36

Técnicas de estimación

Juicio experto

Consulta a varios expertos.
Estiman. Comparan.
Discuten

Técnica Delphi

Sucesivas rondas. Anónimas. Consenso

División de Trabajo

Jerárquica hacia arriba

Planning Poker

Basada en consenso,
incluye todo el equipo de
desarrollo, iterativa

Técnicas de estimación- Juicio experto

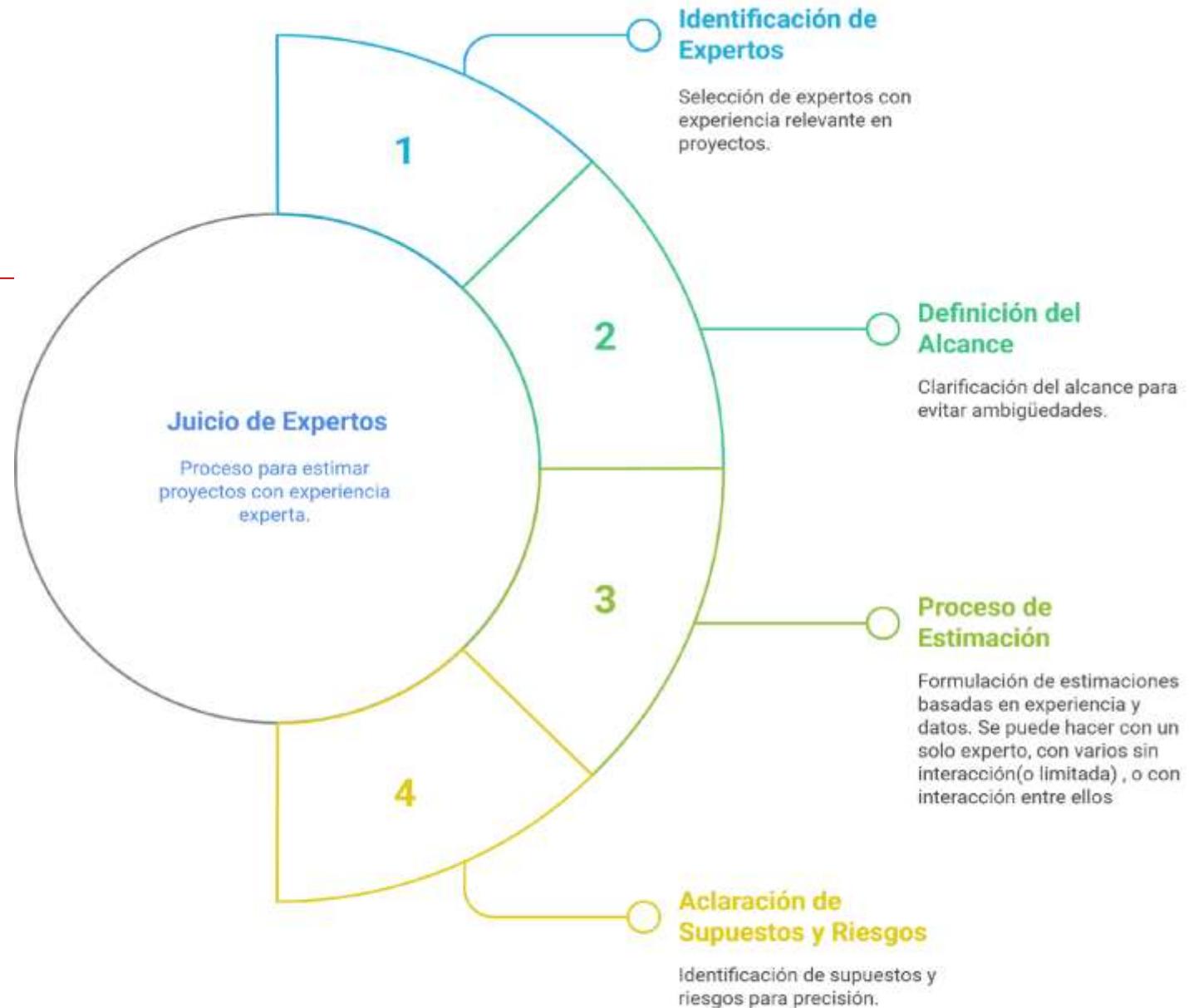
Se basa en la opinión y el conocimiento acumulado de un individuo o grupos que poseen experiencia y pericia relevante en el desarrollo de software, tecnologías específicas, dominios de negocio o gestión de proyectos.

38

Consiste en solicitar a una o varias personas consideradas "**expertas**" que proporcionen una estimación del esfuerzo, tiempo, costo o complejidad necesaria para completar una tarea, característica, módulo o proyecto de software determinado. Los expertos en general no son anónimos.

Técnicas de estimación-

Proceso de estimación por Juicio experto



Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education Limited.

Tecnica Delphi

Método estructurado de comunicación y pronóstico que se basa en la agregación de opiniones de un panel de expertos ((más de un siempre) para llegar a un consenso sobre una estimación

Se caracteriza por:

Anonimato: las identidades de los expertos que participan siempre se mantienen anónimas entre sí. Esto reduce la influencia de personalidades dominantes y permite que cada experto exprese sus opiniones libremente.

Iteración: el proceso se lleva a cabo en múltiples rondas. En cada ronda, los expertos proporcionan sus estimaciones iniciales y las justificaciones de la estimación.

Retroalimentación controlada: después de cada ronda, un facilitador recopila, resume y distribuye las respuestas a todos los expertos.

Búsqueda de consenso: basándose en la retroalimentación, los expertos tienen la oportunidad de revisar y ajustar sus estimaciones en las rondas subsiguientes. El proceso continúa hasta que se alcanza un consenso razonable de estimación.

División del trabajo (Top-down)

Consiste en descomponer un proyecto de software grande y complejo en componentes o actividades más pequeños, manejables y, por lo tanto, más fáciles de estimar de manera individual.

En esencia, el proceso implica:

- 1. Descomposición:** el proyecto de software se divide jerárquicamente en elementos de trabajo progresivamente más pequeños. La descomposición puede basarse en las funciones del software, las actividades del proceso de desarrollo (como análisis, diseño, codificación, pruebas) o una combinación de ambos.
- 2. Estimación de las Partes:** se estima el esfuerzo, el costo y/o la duración requerida para cada una de estas partes individuales. La estimación de estas partes más pequeñas tiende a ser más precisa que intentar estimar el proyecto completo de una sola vez, ya que la complejidad se reduce y es más fácil visualizar el trabajo involucrado.
- 3. Composición (Agregación):** las estimaciones de las partes individuales se suman o combinan para obtener una estimación global del proyecto completo.

Técnicas de estimación- Planning Poker

Es una técnica de estimación colaborativa y basada en el consenso utilizada en el desarrollo de software ágil, especialmente en los marcos de trabajo Scrum. Su objetivo es obtener estimaciones más precisas del esfuerzo requerido para las historias de usuario o elementos del backlog, fomentando la participación y el entendimiento compartido del equipo

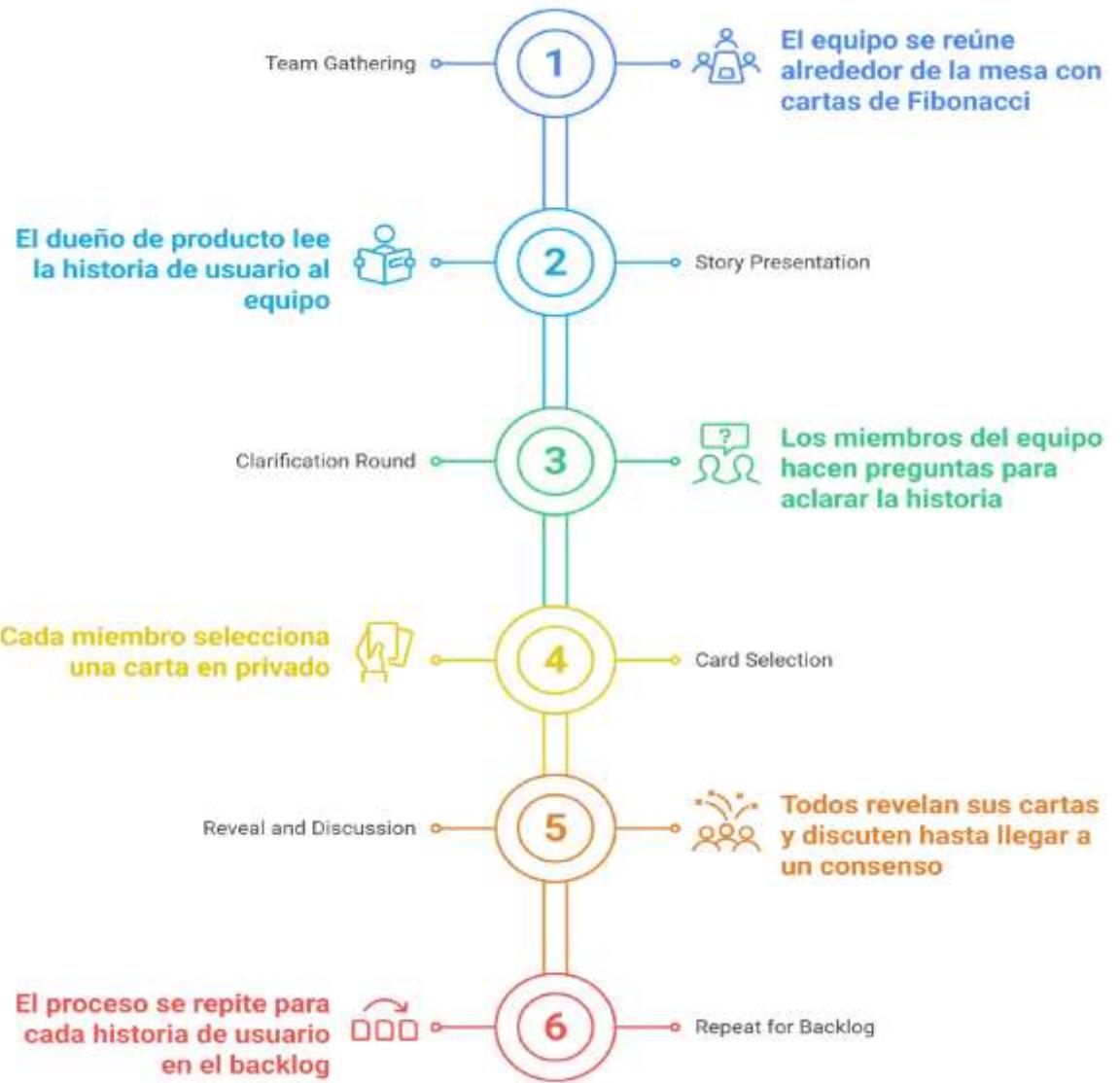
- ❖ Participan todas las personas comprometidas en el desarrollo
- ❖ Cada una de las historias de usuario de un Sprint.
- ❖ Se llama así porque se utilizan cartas numeradas. Lo normal es numerar las cartas con una serie de Fibonacci .



Técnicas de estimación- Planning Poker

Se usan los números de Fibonacci para evitar la "trampa del anclaje" (donde la primera estimación mencionada influye en las demás)

Pasos del Planning Poker



Técnicas de estimación- Ejemplo

Historia de usuario: **Como administrador, quiero generar informes mensuales sobre la actividad de los usuarios y el rendimiento de las ventas.**

Estimación: 8 puntos de historia (secuencia de Fibonacci).

44

Justificación: El equipo considera que esta historia de usuario es una tarea más compleja y compleja. Requiere diseñar e implementar funcionalidades de informes, recopilar y agregar datos de múltiples fuentes y generar información valiosa. La complejidad y el esfuerzo que esto implica resultan en una estimación más alta.



Técnicas de estimación- App

En línea:

<https://planningpokeronline.com/>

45

¿Cuando usar cada técnica de estimación?

Técnica de estimación Uso

Juicio experto	<ul style="list-style-type: none">• En las primeras etapas del proyecto, cuando la información es limitada o el alcance es aún vago.• Para proyectos pequeños o de corta duración donde la inversión en técnicas más elaboradas no se justifica.• Cuando se necesita una estimación rápida y la precisión extrema no es crítica.• Para tareas o proyectos altamente especializados donde pocas personas poseen el conocimiento necesario.• Como punto de partida para refinar con otras técnicas más adelante.
Técnica Delphi	<ul style="list-style-type: none">• Cuando se busca obtener un consenso de un grupo diverso de expertos, incluso si están geográficamente dispersos.• Para proyectos complejos o de alto riesgo donde el sesgo de una sola persona o las dinámicas grupales pueden influir negativamente. Cuando no hay datos históricos suficientes y se depende en gran medida de la experiencia subjetiva.• Para mitigar el efecto de la personalidad dominante en las discusiones grupales, ya que las respuestas son anónimas.

¿Cuando usar cada técnica de estimación?

Técnica de estimación Uso

División del trabajo	<ul style="list-style-type: none">Para proyectos grandes y complejos que necesitan ser descompuestos en componentes manejables.Cuando se requiere un alto nivel de detalle y visibilidad sobre el alcance del proyecto.Para asignar responsabilidades claras a equipos o individuos para partes específicas del proyecto. Como base para la planificación, el seguimiento y el control del proyecto.Generalmente se utiliza en conjunto con otras técnicas de estimación (como el juicio experto o la estimación análoga/paramétrica) una vez que las tareas de bajo nivel han sido definidas
Planning Poker	<ul style="list-style-type: none">Principalmente en metodologías ágiles (Scrum, Kanban) para estimar historias de usuario o elementos del backlog.Cuando se desea fomentar la colaboración y el consenso dentro del equipo de desarrollo.Para identificar y discutir suposiciones, dependencias y riesgos de las tareas de forma proactiva.Cuando se busca que el equipo de desarrollo sea dueño de sus propias estimaciones, lo que fomenta un mayor compromiso y precisión.Ideal para estimar elementos que varían en complejidad y requieren una discusión abierta para ser bien entendidos,

Modelos empíricos de estimación

COCOMO (Constructive Cost Model) es un modelo de estimación de costos para proyectos de software, desarrollado por Barry Boehm.

Este modelo utiliza fórmulas matemáticas para predecir el esfuerzo, el tiempo y el costo de un proyecto de software en función de su tamaño, complejidad y otros factores.

48

- Se basa en la experiencia y en datos de proyectos reales.
- Considera factores como el tamaño del proyecto, la complejidad del código, la experiencia del equipo y el entorno de desarrollo.
- Ofrece diferentes niveles de detalle para la estimación, desde modelos básicos hasta modelos detallados.

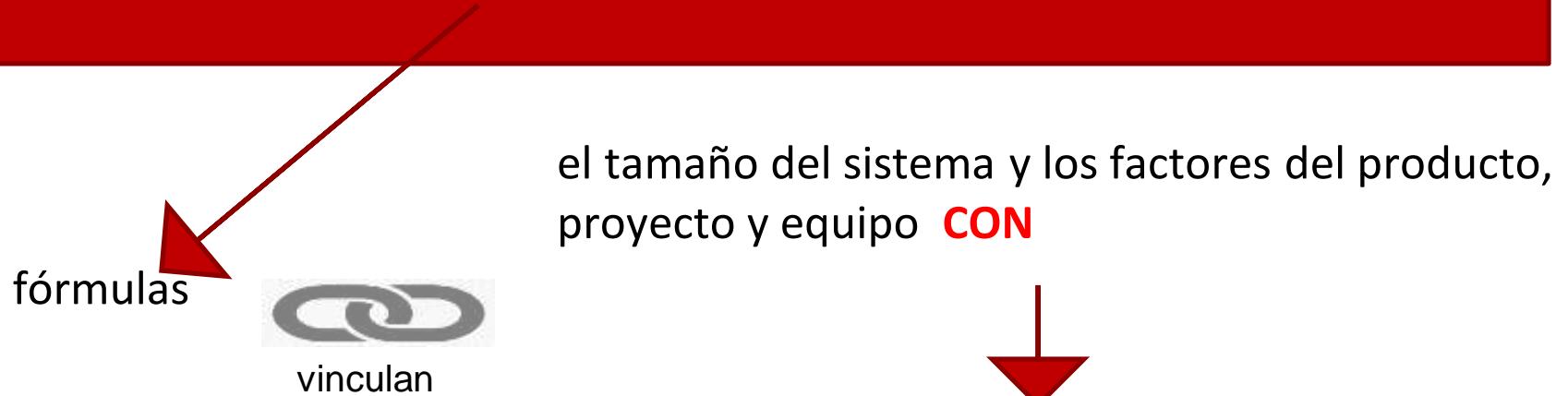
Modelos empíricos de estimación

Utilizan fórmulas derivadas empíricamente para predecir costos o esfuerzo requerido en el proyecto

COCOMO II

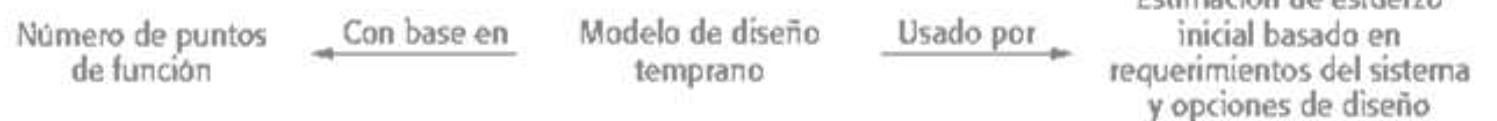
Modelo empírico que derivó de recopilar datos a partir de un gran número de proyectos de software.

49



Fuente: Somerville Cap. 26

COCOMO II - Modelos



Pressman, R. S., & Maxim, B. R. (2023). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.

50

COCOMO II

1. Modelo de composición de aplicación



Modela el **esfuerzo** requerido para desarrollar sistemas creados a partir de proyectos de creación de prototipos.



Se basa en una **estimación** de *puntos de aplicación* (o puntos objetos)

$$PM = (NAP \times (1 - \% \text{reutilización}/100))/PROD$$

de programación imperativa (como Java) y el número de líneas de lenguaje de script (*scripting language*).

Parámetros

PM = esfuerzo estimado en personas/mes

NAP = total de puntos de aplicación.

PROD = productividad medida en puntos objeto

COCOMO II

2. Modelo de diseño temprano

-  Puede usarse durante las primeras etapas de un proyecto arquitectónico detallado para el sistema.
-  Supone que se acordaron los requerimientos del usuario en el proceso de diseño del sistema
-  La meta en esta etapa debe ser elaborar una estimación

$$PM = A \times Tamaño^B \times M$$

Parámetros

PM = esfuerzo estimado en personas/mes

A = 2,94. (según Boehm)

Tamaño = KLDC (miles de líneas de código fuente).

B = esfuerzo requerido conforme aumenta el tamaño del proyecto. Puede variar de 1,1 a 1,24

M = definido por 7 atributos de proyecto y proceso que aumentan o disminuyen la estimación. EJ: fiabilidad y complejidad del producto, etc

2

COCOMO II

3. Modelo de reutilización



se emplea para estimar el esfuerzo requerido al integrar código



Para el código generado automáticamente, el modelo estima el esfuerzo para integrar este código

$$PM_{auto} = (ASLOC \times AT/100)/ATPROD.$$

Parámetros

PM = esfuerzo estimado en persona s/mes

AT = % de código adaptado que se genera automáticamente.

ATPROD = productividad de los ingenieros que integran el código

ASLOC = Nro de líneas de código en los componentes que deben ser adaptadas

53

COCOMO II

4. Modelo de post-arquitectura



Se usa cuando está disponible un diseño arquitectónico inicial (se conoce la estructura) . Entonces es posible hacer estimaciones para cada parte del sistema.

Las estimaciones están basadas en la misma fórmula básica

$$PM = A \times \text{Tamaño}^B \times M$$

54



pero se utiliza un conjunto más extenso de atributos de M



La estimación del número de líneas de código se calcula utilizando tres componentes:

- **líneas nuevas** de código a desarrollar.
- líneas de código fuente **equivalentes** (ESLOC) calculadas usando el nivel de reutilización.
- líneas de código que **tienen que modificarse** debido a cambios en los requerimientos.
- Estas estimaciones se añaden para obtener el tamaño del código (KLDC).

Modelos COCOMO

Limitaciones:

- Los modelos de COCOMO pueden ser complejos de usar, y requieren de datos históricos y experiencia para calibrar los factores de costo.
- Se miden los costes del producto, de acuerdo a su tamaño y otras características, pero no la productividad.
- La medición por líneas de código no es válida para orientación a objetos; entre otras cosas por la "reusabilidad" y la herencia, características de este paradigma (e.g., puede implicar importante aumento en productividad; pero no en líneas de código).

55

Estimaciones

- ❖ COCOMO 1:

<https://strs.grc.nasa.gov/repository/forms/cocomo-calculation/>

- ❖ COCOMO 2:

<http://softwarecost.org/tools/COCOMO/>

56



Ingeniería de Software II

Tipos de prueba

Prueba del software

- » La etapa de Prueba no es la primera instancia en que se localizan defectos.
- » Se ha visto que la revisión de requerimientos y el diseño contribuyen a descubrir los problemas (defectos) en las etapas tempranas.

2

Prueba del software

» ¿Qué significa que el software ha fallado?

El software no hace lo que especifican los requerimientos

» Posibles razones:

- *Especificación errónea.*
- *Requerimientos imposibles con las estructuras previstas.*
- *Defectos en diseño del sistema.*
- *Defectos en diseño del programa.*
- *Defectos en código.*

3

Tipos de Defecto

4

» Algorítmicos

Ej. : No inicializar variables

» De sintaxis

Ej. : Confundir un 0 por una O

» De precisión

Ej. : Fórmulas no implementadas correctamente

» De documentación

Ej. : Documentación no acorde con lo que hace el software

» De sobrecarga

Ej. : El sistema funciona bien con 100 usuarios pero no con 110.

Tipos de Defecto

» De capacidad

Ej. : El sistema funciona bien con ventas <1.000.000

» De coordinación o sincronización

Ej.: Comunicación entre procesos con fallas

» De rendimiento

Ej.: Tiempo de respuesta inadecuado.

» De recuperación

Ej. : No volver a un estado normal luego de una falla

» De relación hardware-software

Ej.: Incompatibilidad entre componentes

» De estándares

Ej. : No cumplir con la definición de estándares y procedimientos

Clasificación ortogonal de Defectos

- » Los defectos se han organizado en categorías.
- » Primeramente se debe identificar si es un:
 - Defecto por omisión** (resulta cuando algún aspecto clave del código falta).
Ej: variable no inicializada.
 - Defecto de cometido** (resulta cuando algún aspecto es incorrecto).
Ej: variable inicializada con un valor erróneo.

Clasificación ortogonal de Defectos

»Pfleeger Cap. 8

7

Tipo de defecto	Significado
Función	Afecta la capacidad, interfaces.
Interfaz	Afecta a la interacción con otros componentes.
Comprobación	Afecta la lógica del programa.
Asignación	Afecta la estructura de datos.
Sincronización	Involucra sincronización de recursos compartidos y de tiempo real.
Construcción	Ocurre debido a problemas en repositorios, gestión de cambios o control de versiones.
Documentación	Afecta a publicaciones.
Algoritmo	Involucra la eficiencia o exactitud de un algoritmo.

Prueba del software

-
- » ¿Cuál es el primer objetivo de la prueba?
 - » Diseñar pruebas que saquen a la luz diferentes clases de errores, haciéndolo en la menor cantidad de tiempo y esfuerzo.
 - » ¿Cuándo una prueba tiene éxito?

Cuándo descubre errores

8



Objetivos y Beneficios de las Pruebas del Software

9

- » Descubrir errores antes que el software salga del ambiente de desarrollo
- » Detectar un error no descubierto hasta entonces.
- » Bajar los costos de corrección de errores en la etapa de mantenimiento

Principios de la Prueba

- » A todas las pruebas se les debería poder hacer un seguimiento hasta los requisitos del cliente.
- » Las pruebas deberían planificarse mucho antes de que empiecen.
- » Es aplicable el principio de Pareto. **El mismo dice que "el 80% de los errores de un software es generado por un 20% del código de dicho software, mientras que el otro 80% genera tan sólo un 20% de los errores".**

10



Principios de la Prueba

- » Las pruebas deberían empezar por «lo pequeño» y progresar hacia «lo grande»
- » Es importante asegurarse que se han aplicado (probado) todas las condiciones a nivel de componente.
- » Para ser más eficaces, las pruebas deberían ser realizadas por un equipo independiente.

11



¿Quién realiza las pruebas?

- » Varios factores justifican un equipo independiente de pruebas, entre ellos:
 - ❖ Evitar el conflicto entre la responsabilidad por los defectos y la necesidad de descubrir defectos
 - ❖ Llevar a cabo las pruebas concurrentemente con la codificación.
 - ❖ Los desarrolladores deben colaborar y corregir los errores.

12

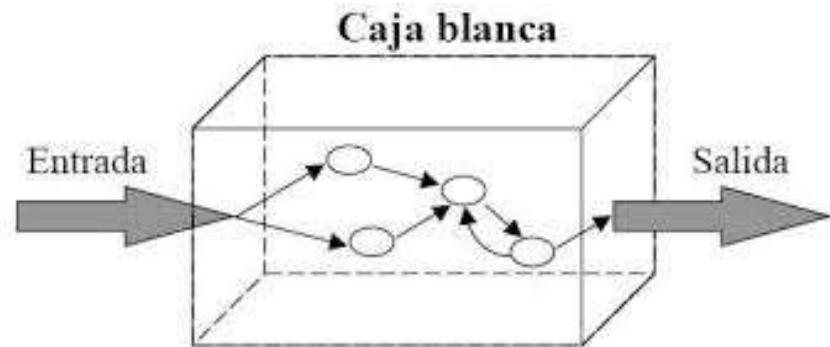
Pruebas del Software

- »Lamentablemente “La prueba **NO** puede asegurar la ausencia de defectos”
- »Se intentan buscar Casos de Prueba que permitan encontrar errores

13

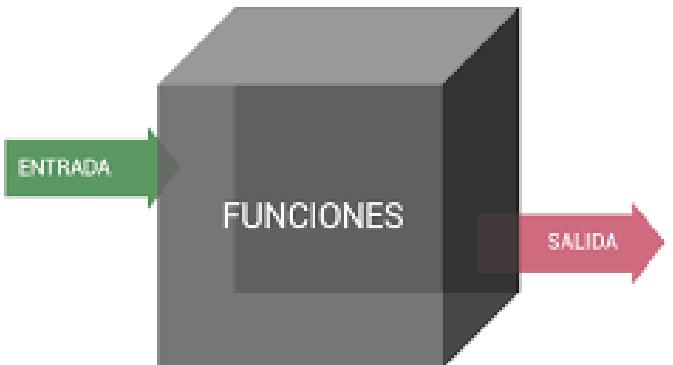
Tipos de Prueba del Software

» La prueba de **caja blanca** se basa en el minucioso examen de los detalles procedimentales. Se comprueban los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles.



14

» La prueba de **caja negra** se refiere a las pruebas que se llevan a cabo sobre la interfaz del software.



Tipos de Prueba

¿En qué momento se pueden definir los casos de prueba?

Ejemplo 1: Tengo una función del sistema que busca un número en una secuencia de números y devuelve la posición en la que se encuentra (-1 si no lo encuentra)

¿Qué valores podemos definir como casos de prueba para el ejemplo 1?
¿Por qué?

¿Cuántos valores debemos probar?

¿Cuándo lo podemos probar efectivamente?

15

Ejemplo 2:

```
Procedure eliminarValor (var pri:  
Lista; n:integer);  
Var pos,ant:lista;  
Begin  
  pos:= pri; ant:= pri; ok:= false;  
  while (pos <> nil) and (not ok)do  
    if (pos^.datos = n) then ok:=  
true  
    else begin  
      ant:=pos;  
      pos:= pos^.sig;  
    end;
```

¿Qué valores debemos probar?

¿Qué probarían en este código?
¿Por qué?

¿Cómo garantizamos que recorrimos todos los caminos posibles?

```
if (ok=true) then  
begin  
  if (pos = pri) then  
    pri:= pos^.sig  
  else begin  
    ant^.sig:= pos^.sig  
    dispose (pos);  
  end;  
End;
```



Caja Negra

2025

Tipos de Prueba Caja Negra (o Cerrada)

- » También denominada prueba de comportamiento, se centran en los requisitos funcionales del software.
- » Intenta descubrir diferentes tipos de errores que los métodos de caja blanca.

18



Tipos de Prueba Caja Negra (o Cerrada)



¿Qué errores busca las pruebas de
Caja Negra?



Funciones incorrectas

Identificación de errores en funciones que son incorrectas.



Errores de interfaz

Identificación de errores en la interfaz de usuario.



Errores en estructuras de datos

Identificación de errores en estructuras de datos o bases de datos.



Errores de rendimiento

Identificación de errores relacionados con el rendimiento.



Errores de inicialización

Identificación de errores de inicialización y de terminación.

Made with Napkin

Prueba De Partición Equivalente

- » El diseño de los casos de prueba se basa en una evaluación de las clases de equivalencia para una condición de entrada.
- » Una clase de equivalencia representa un conjunto de estados válidos o no válidos para condiciones de entrada.
- » Una condición de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

20



Prueba De Partición Equivalente - Definición

Si una condición de entrada especifica un **rango**, se define una clase de equivalencia válida y dos no válidas.

Si una condición de entrada requiere un **valor específico**, se define una clase de equivalencia válida y dos no válidas. 21

Si una condición de entrada especifica un elemento de un **conjunto**, se define una clase de equivalencia válida y una no válida.

Si una condición de entrada es **lógica**, se define una clase de equivalencia válida y una no válida.

Prueba De Partición Equivalente

Ejemplo: Dar de alta un Juguete

DAR DE ALTA JUGUETE :



Código:

Nombre:

Descripción:

Recomendaciones:

Género:

Edad:

Marca:

Stock: Stock mínimo:

Estado:

ACEPTAR CANCELAR

Dato	Tipo
Código	entero positivo
Nombre	string 20
descripción	string 256
Recomendaciones	string 512
Genero	enumerativo (masculino, femenino, no binario)
Edad	rango 0..120
Marca	string 25
Stock	entero
stock mínimo	entero positivo
Estado	enumerativo (normal, oferta, novedoso)

22

Identificación de las clases de equivalencia.

<i>Condición de entrada</i>	<i>Clases de equivalencia válidas</i>	<i>Clases de equivalencia inválidas</i>
código	$0 \leq \text{código} \leq 9999$	código < 0 código > 9999
nombre	1 a 20 caracteres	0 caracteres; mas de 20 caracteres;
descripción	0 a 256 caracteres	mas de 256 caracteres
recomendaciones	0 a 512 caracteres	mas de 512 caracteres
genero	masculino femenino	otra cadena de caracteres;
stock	Número entero	Caracteres no dígitos;

Análisis de Valores Límite (AVL)

- » Los errores tienden a darse más en los **límites** del campo de entrada que en el «centro».
- » Por ello, se ha desarrollado el análisis de valores límites (AVL) como técnica de prueba.
- » **Complementa a la partición equivalente.** En lugar de seleccionar cualquier elemento de una clase de equivalencia, el AVL selecciona los casos de prueba en los «extremos» de la clase. En lugar de centrarse solamente en las condiciones de entrada, el AVL obtiene casos de prueba también para el campo de salida

24

Caja Negra: Análisis de Valores Límite

- » Casos de prueba en los bordes de las clases:
- » Para una condición de **entrada de rango entre a y b** probar: a, b, <a y >b.
- » Para una **salida de rango entre a y b**: utilizar casos de prueba que generen valor de salida a y b
- » Probar las **estructuras de datos internas** en sus límites.

25



Caja Blanca

2025

Tipos de Prueba

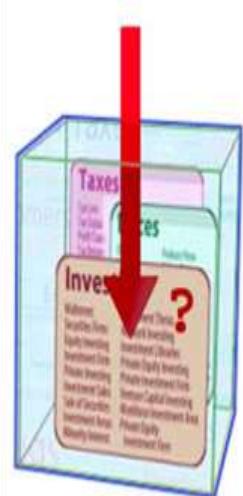
Caja Blanca (o Cristal o Abierta)

» Deriva casos de prueba de la estructura de control, para verificar detalles procedimentales. Mediante los métodos de prueba de caja blanca, el ingeniero del software puede obtener casos de prueba que garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo.

ejerciten todas las decisiones lógicas .

ejecuten todos los bucles en sus límites..

ejerciten las estructuras internas de datos
para asegurar su validez.



Tipos de Prueba

Caja Blanca (o Cristal o Abierta)

- » El flujo lógico de un programa a veces no es nada intuitivo, lo que significa que nuestras suposiciones intuitivas sobre el flujo de control y los datos nos pueden llevar a tener errores de diseño que sólo se descubren cuando comienza la prueba del camino.
- » ¿Por qué realizarlas?
- » Los casos especiales son los más factibles de error
- » Los errores tipográficos son aleatorios
- » El flujo de control intuitivo es distinto del real

28

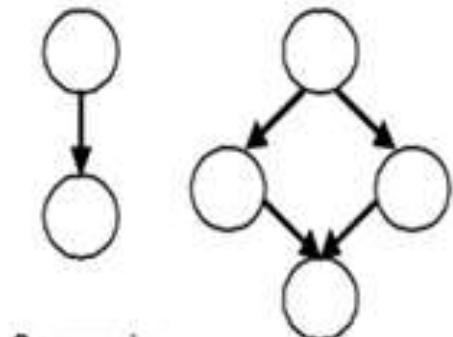
Prueba Del Camino Básico

- » Es una técnica propuesta por Tom McCabe. Permite al diseñador de casos de pruebas obtener una medida de la complejidad lógica y usarla como guía para la definición de caminos de ejecución.
- » Los casos de prueba obtenidos garantizan que se ejecuta al menos una vez cada sentencia del programa.

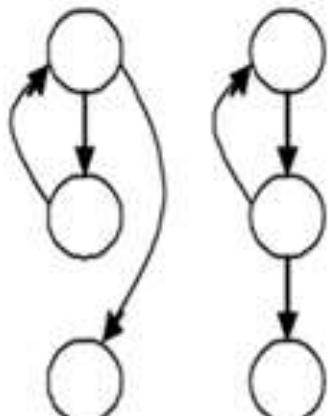
29

Prueba Del Camino Básico – Notación de grafo

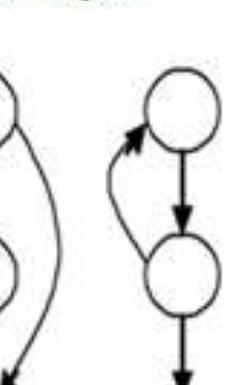
Estructuras en forma de grafo de flujo:



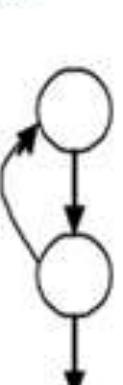
Secuencia



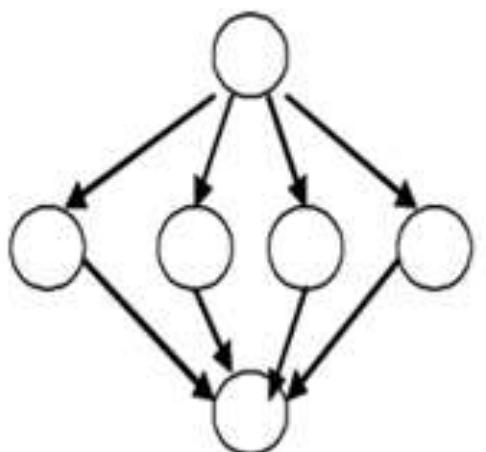
Condicional



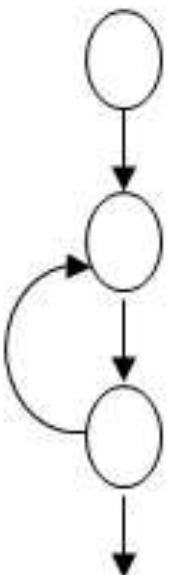
Mientras
ocurra



Repite
hasta



Case



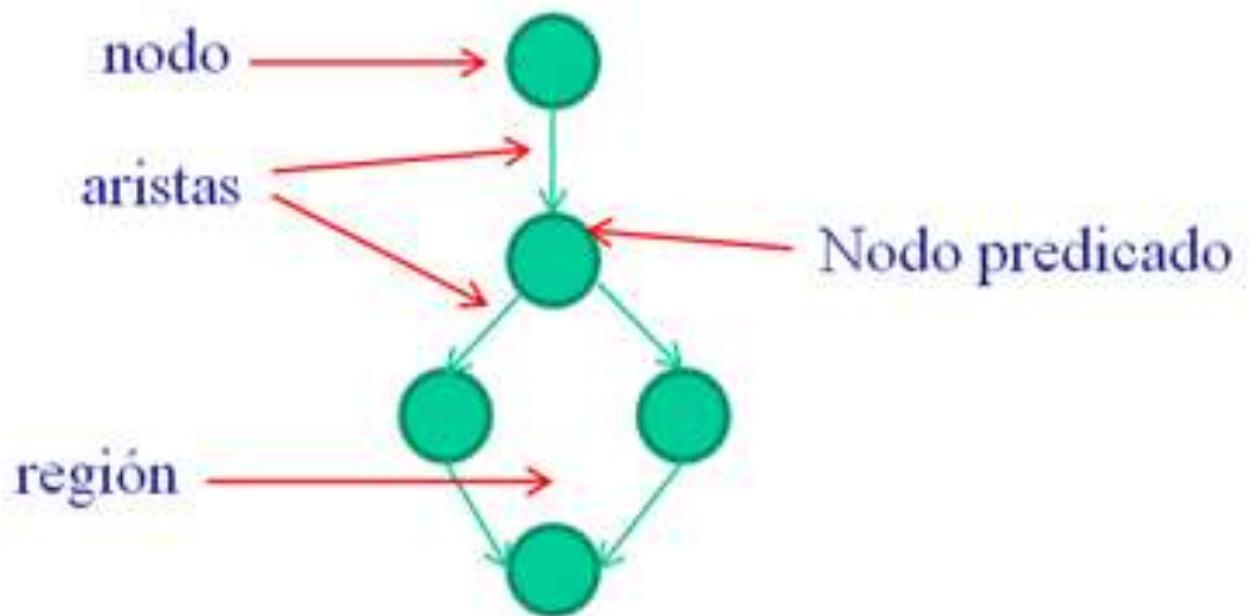
For

30

Prueba Del Camino Básico

- » Cada círculo, denominado **nodo** del grafo de flujo, representa una o más sentencias procedimentales.
- » Las flechas del grafo de flujo, denominadas **aristas** o representan flujo de control.
- » Una arista debe terminar en un nodo.

Cada nodo que contiene una condición se denomina nodo predicado y está caracterizado porque dos o más aristas emergen de él.



Las áreas delimitadas por aristas y nodos se denominan regiones. Cuando contabilizamos las regiones incluimos el área exterior del grafo, contándolo como otra región más.

Prueba Del Camino Básico – Complejidad ciclomática

- » **La complejidad ciclomática**
- » es una **métrica del software** que proporciona una medición cuantitativa de la complejidad lógica de un programa.
- » **define el número de caminos independientes** del conjunto básico de un programa y nos da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecuta cada sentencia al menos una vez.
- » **Un camino independiente** es cualquier camino del programa que introduce, por lo menos, un nuevo conjunto de sentencias de proceso o una nueva condición.

32

Prueba Del Camino Básico

»Complejidad ciclomática :

1. $V(g)$ =Cantidad de regiones del grafo ó
2. $V(g)= A - N + 2$ ó
3. $V(g)= P + 1$

33

»La complejidad ciclomática debe medirse con las tres formulas de manera de verificar su exactitud.

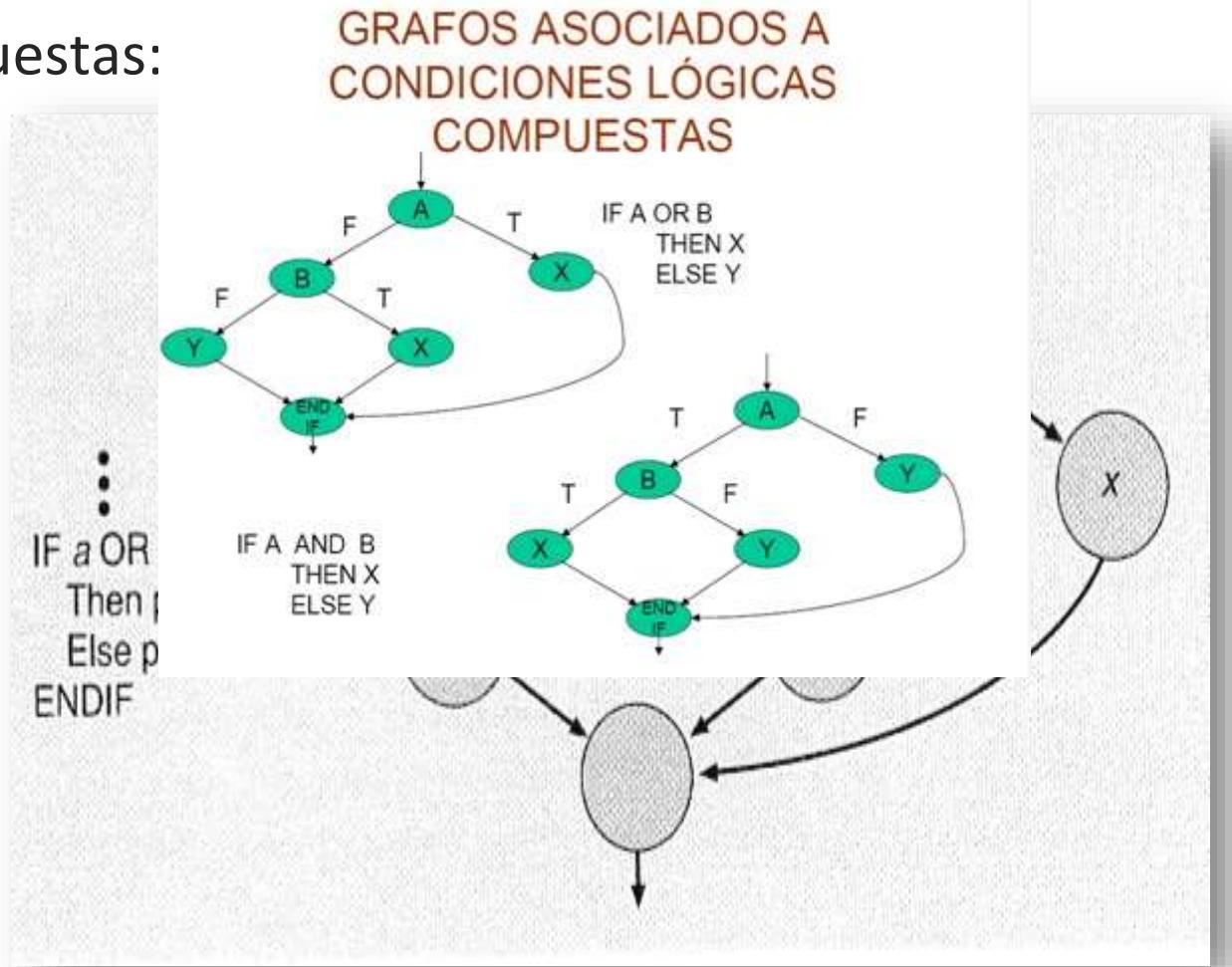
Prueba Del Camino Básico – Pasos para crear la prueba

1. Dibujar el grafo de flujo correspondiente.
2. Determinar la complejidad ciclomática.
3. Determinar un conjunto básico de caminos independientes.
4. Preparar los casos de prueba que forzarán la ejecución de cada camino del conjunto.
5. Ejecutar cada caso de prueba y comparar los resultados obtenidos con los esperados.

34

Prueba Del Camino Básico

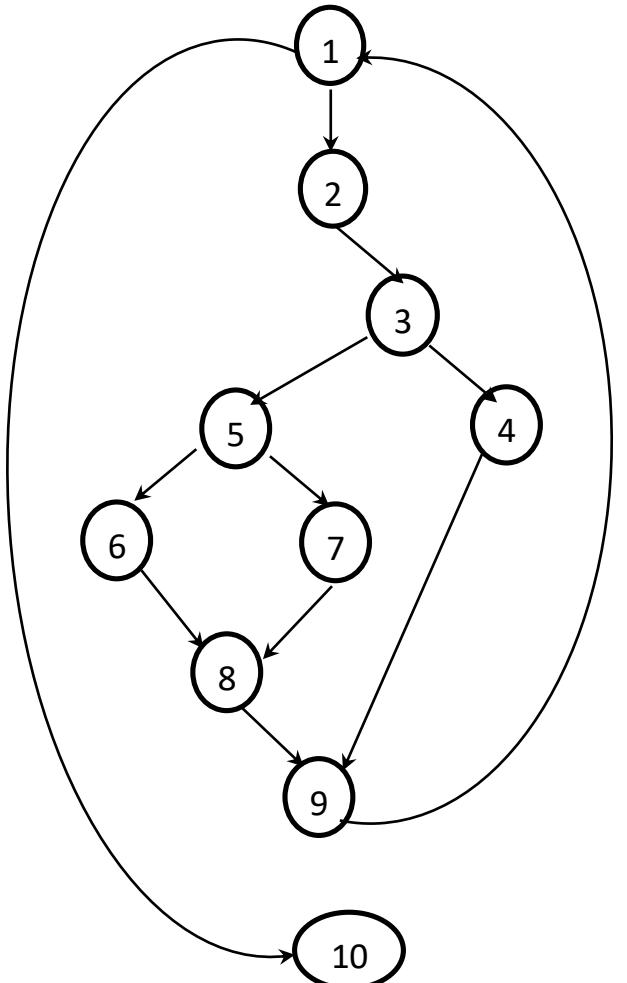
»Condiciones compuestas:



Prueba Del Camino Básico

Procedure Ordenar

```
(1) do while not eof() begin
(2)   leer registro;
(3)   if (campo 1 de registro = 0)
(4)     then procesar registro
        Guardar en buffer;
        Incrementar contador;
(5)   else if (campo 2 de registro = 0)
(6)     then reiniciar contador
(7)   else
        procesar registro;
        Guardar en archivo;
(8)   endif
(9) endif
(10) end;
```



36

Prueba Del Camino Básico

Regiones : 4

Nodos : 10

Aristas : 12

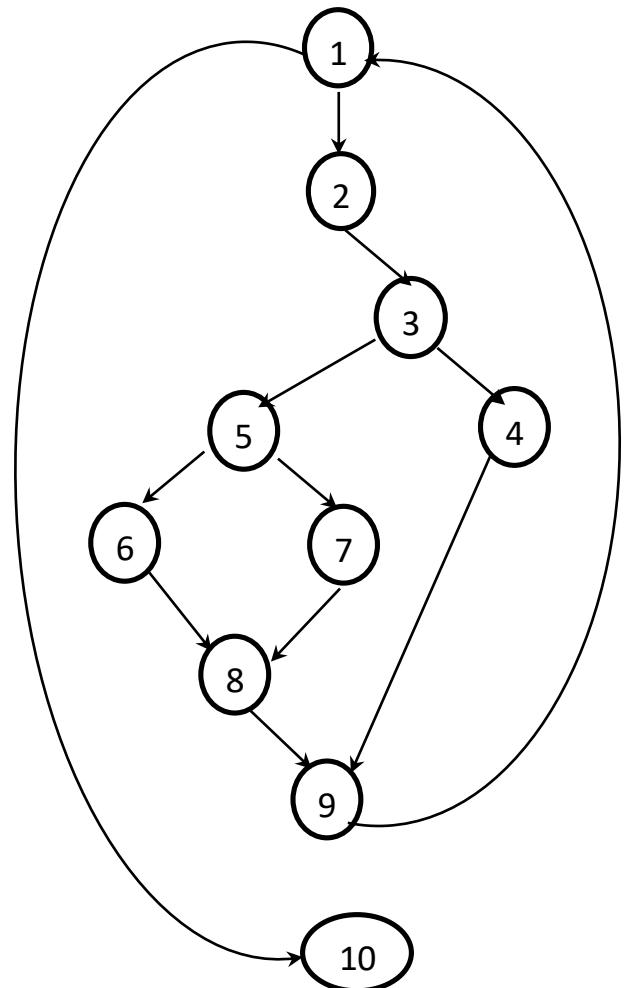
Nodos Predicados : 3

$$V(G) : A - N + 2 = 12 - 10 + 2 = 4$$

$$V(G) : NP + 1 = 3 + 1 = 4$$

$$V(G) : R = 4$$

37



programejcaracteres;

uses crt;

var

```
car: char; canta: integer; cantPal: integer;  
priCar, ultCar: char;  
cantCar: integer;
```

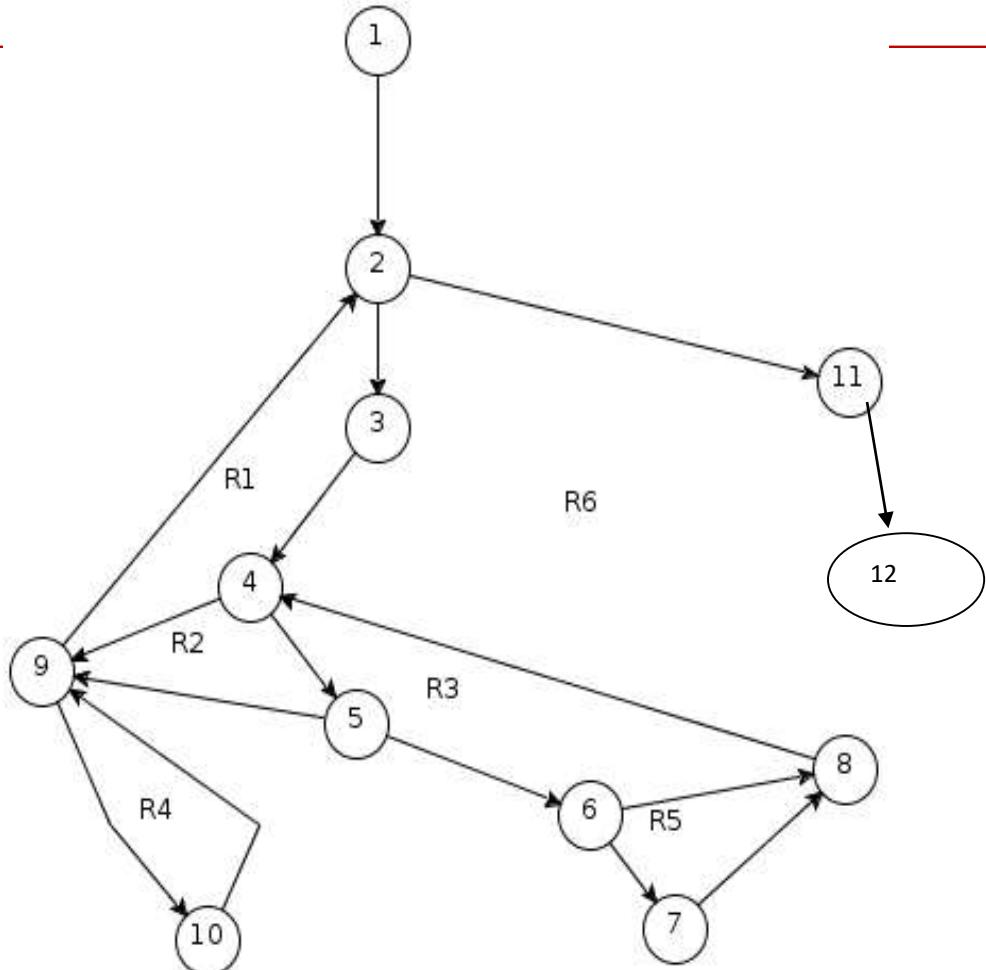
begin

```
canta:= 0; cantPal:=0;  
read(car);  
while(car <> '.') do begin  
    cantPal:= cantPal + 1;  
    priCar:=car;  
    cantCar:=0;  
    while(car <> ' ') and (car <> '.') do begin  
        if(car = 'a') then  
            canta:= canta+1;  
        cantCar:= cantCar +1;  
        ultCar:=car;  
        read(car);  
    end;  
    while(car = ' ') do read(car);  
end;  
writeln('Cantidad de letras a ', canta);  
writeln('Cantidad de palabras ',cantPal);  
end.
```

38

CO

39



```

programejcaracteres;
uses crt;
var
  car: char; canta: integer; cantPal: integer; priCar, ultCar:char;
  cantCar: integer;
begin
  canta:= 0;
  cantPal:=0;
  read(car);
  1
  while(car <> '.') do begin
    cantPal:= cantPal + 1;
    priCar:=car;
    3
    cantCar:=0;
    4
    while(car <> ' ') and (car <> ',') do begin
      5
      if(car = 'a') then
        canta:= canta+1;
      7
      cantCar:= cantCar +1;
      ultCar:=car;
      read(car);
      8
    end;
    9
    while(car = ' ') do read(car);
    10
  end;
  writeln('Cantidad de letras a ', canta);
  writeln('Cantidad de palabras ',cantPal);
  11

```

Prueba Del Camino Básico – Complejidad ciclomática

Regiones : 6

Nodos : 12

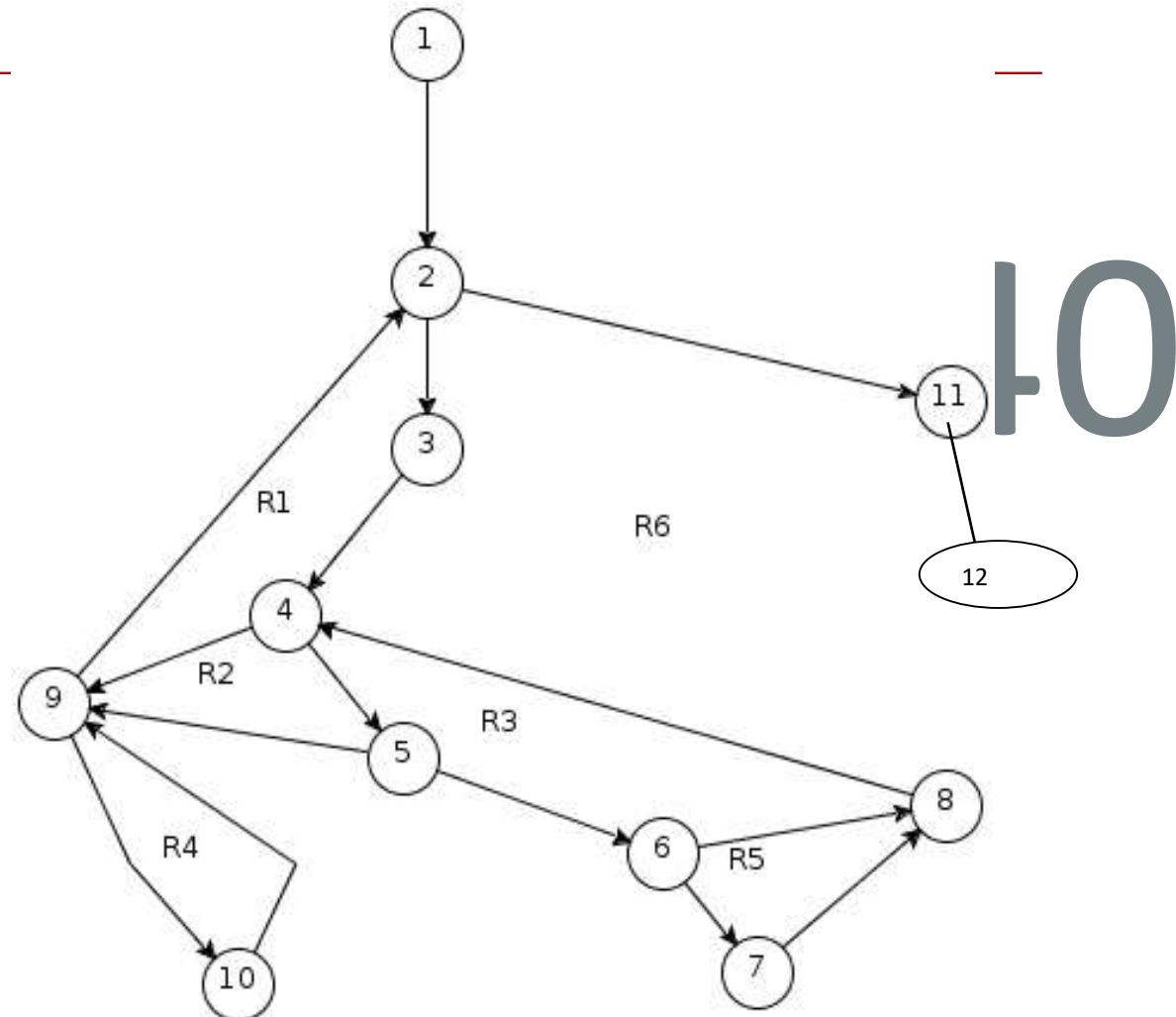
Aristas : 16

Nodos Predicados : 5

$$V(G) : A - N + 2 = 16 - 12 + 2 = 6$$

$$V(G) : NP + 1 = 5 + 1 = 6$$

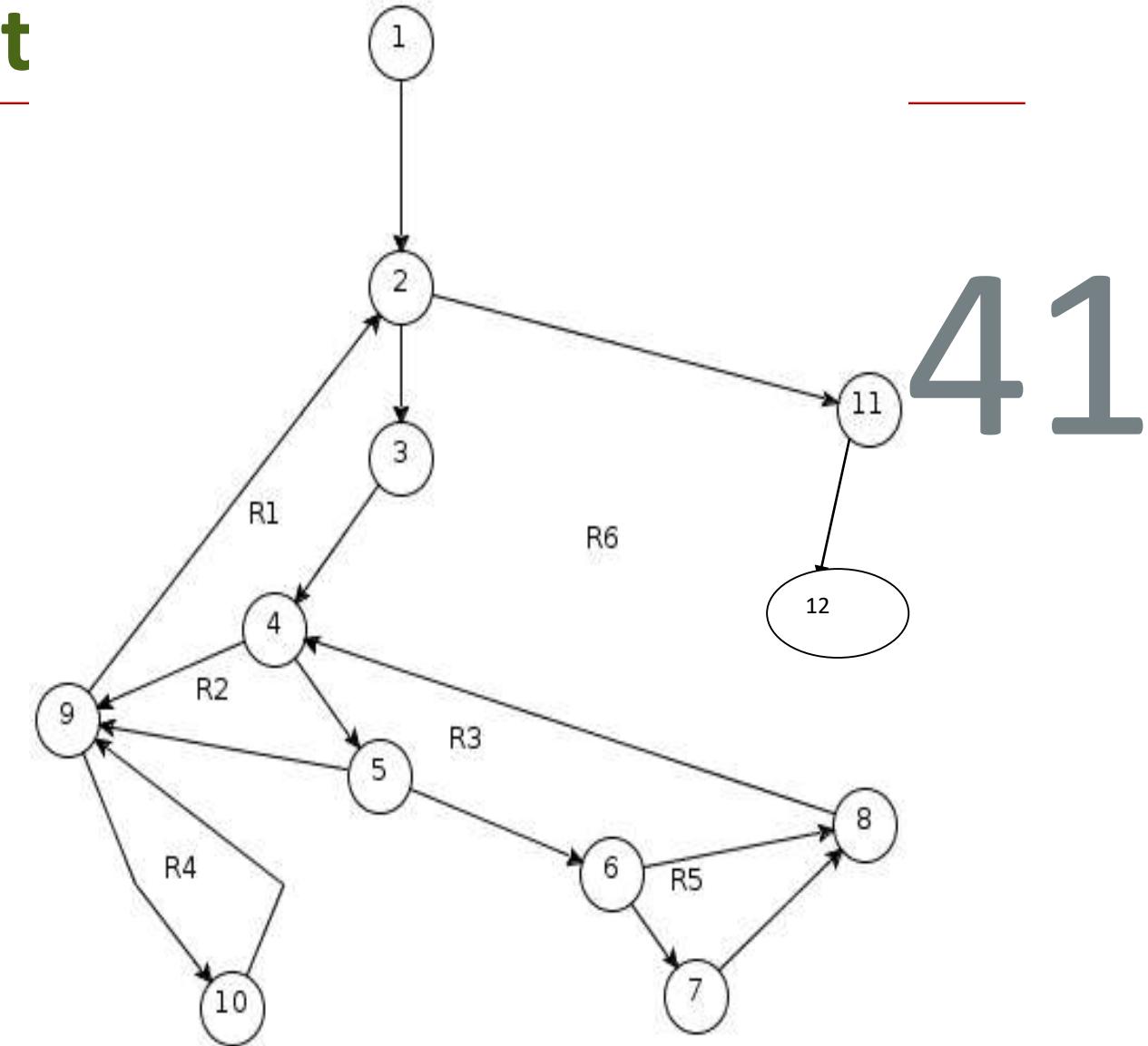
$$V(G) : R = 6$$



Prueba Del Camino Básico – Caminos linealmente independient

- »Camino 1: 1-2-11-12
- »Camino 2: 1-2-3-4-9-2-11-12
- »Camino 3: 1-2-3-4-9-10-9-2-11-12
- »Camino 4: 1-2-3-4-5-9-2-11-12
- »Camino 5: 1-2-3-4-5-6-8-4-9-2-11-12
- »Camino 6: 1-2-3-4-5-6-7-8-4-9-2-11-12

Cualquier otro camino que se quiera recorrer ya fue probado previamente en alguno de los 6 caminos



Prueba Del Camino Básico – Casos de prueba

»Caso de prueba del camino 1: 1-2-11-12

car= ‘.’

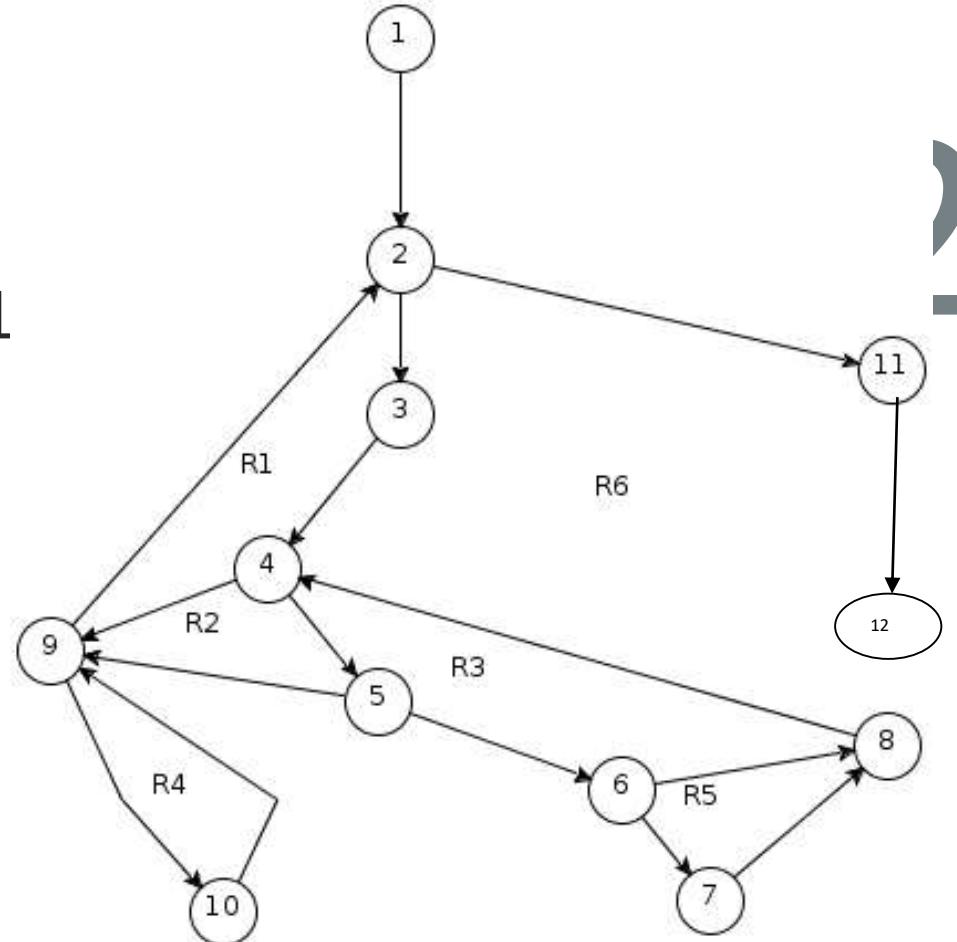
resultados esperados: canta = 0, cantPal=0

»Caso de prueba del camino 2: 1-2-3-4-9-2-11

car= ‘ ’

resultados esperados: canta = 0, cantPal=0

»Caso de prueba del camino



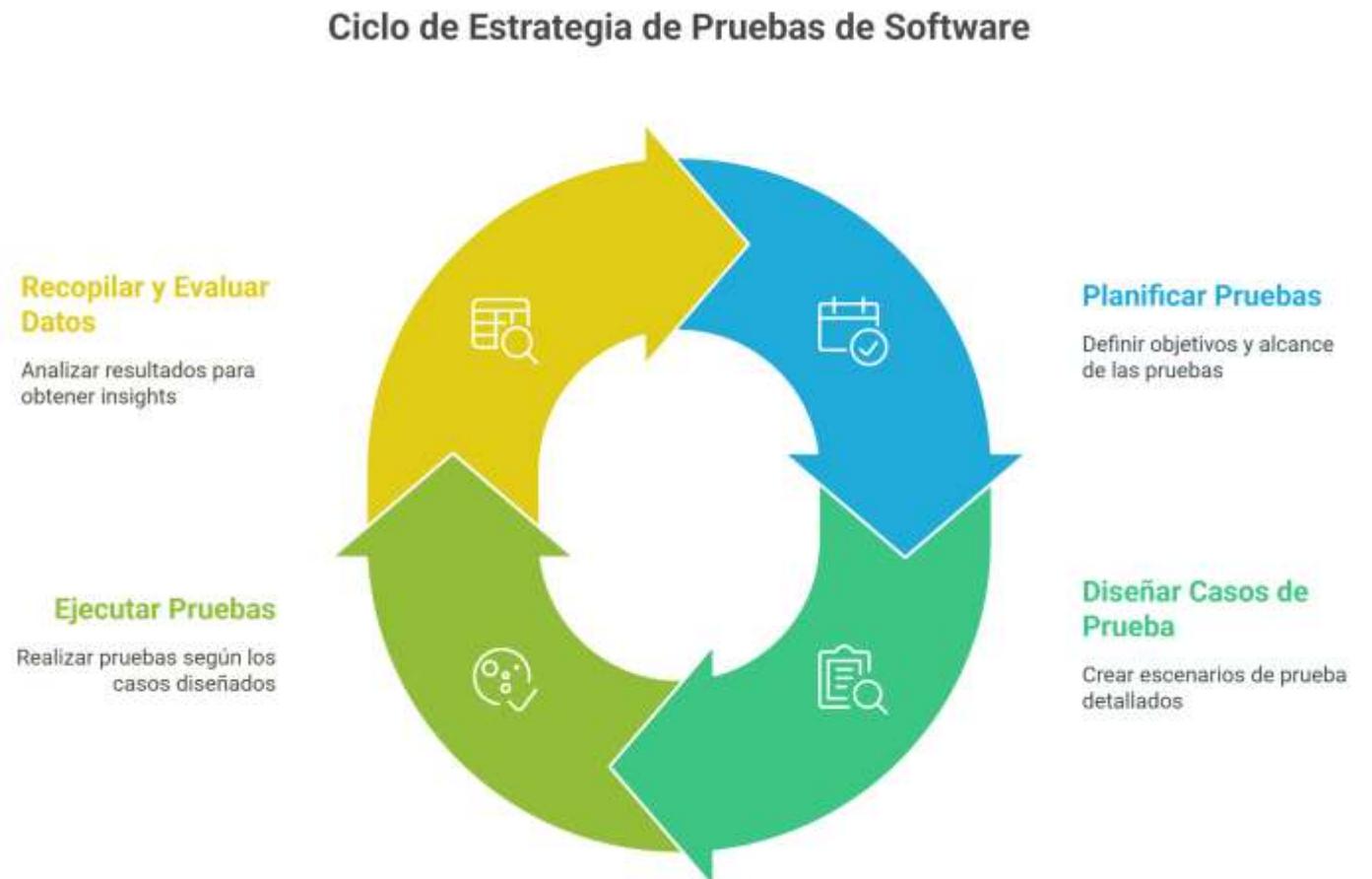


Ingeniería de Software II

Estrategias de prueba PARTE 1

Enfoque estratégico de pruebas

Una estrategia de pruebas del software proporciona una guía que describe los pasos a seguir, cuándo se planean y llevan a cabo, cuánto esfuerzo, tiempo y recurso se requerirán.



Enfoque estratégico de pruebas

- » La prueba es un **conjunto de actividades** que se planean con anticipación y se realizan de manera sistemática.
- » Conjunto de pasos en el que se incluyen técnicas y métodos específicos del diseño de casos de prueba.
- » Una estrategia de pruebas debe incluir pruebas de **bajo nivel y de alto nivel**
- » Las actividades de las estrategias de pruebas son parte de la **Verificación y Validación** incluidas en el aseguramiento de la calidad del software

3

Concepto de Verificación & Validación

»La **verificación** es el conjunto de actividades que asegura que el software implemente correctamente una función específica y **validación** es un conjunto diferente de actividades que aseguran que el software construido corresponde con los requisitos del cliente.

4

»**Verificación** : *¿Estamos construyendo el producto correctamente?*

Comprobar que el software está de acuerdo con su especificación, donde se debe comprobar que satisface tanto los requerimientos funcionales como los no funcionales.

»**Validación** : *¿Estamos construyendo el producto correcto?*

Es un proceso más general, cuyo objetivo es asegurar que el software satisface las expectativas del cliente.

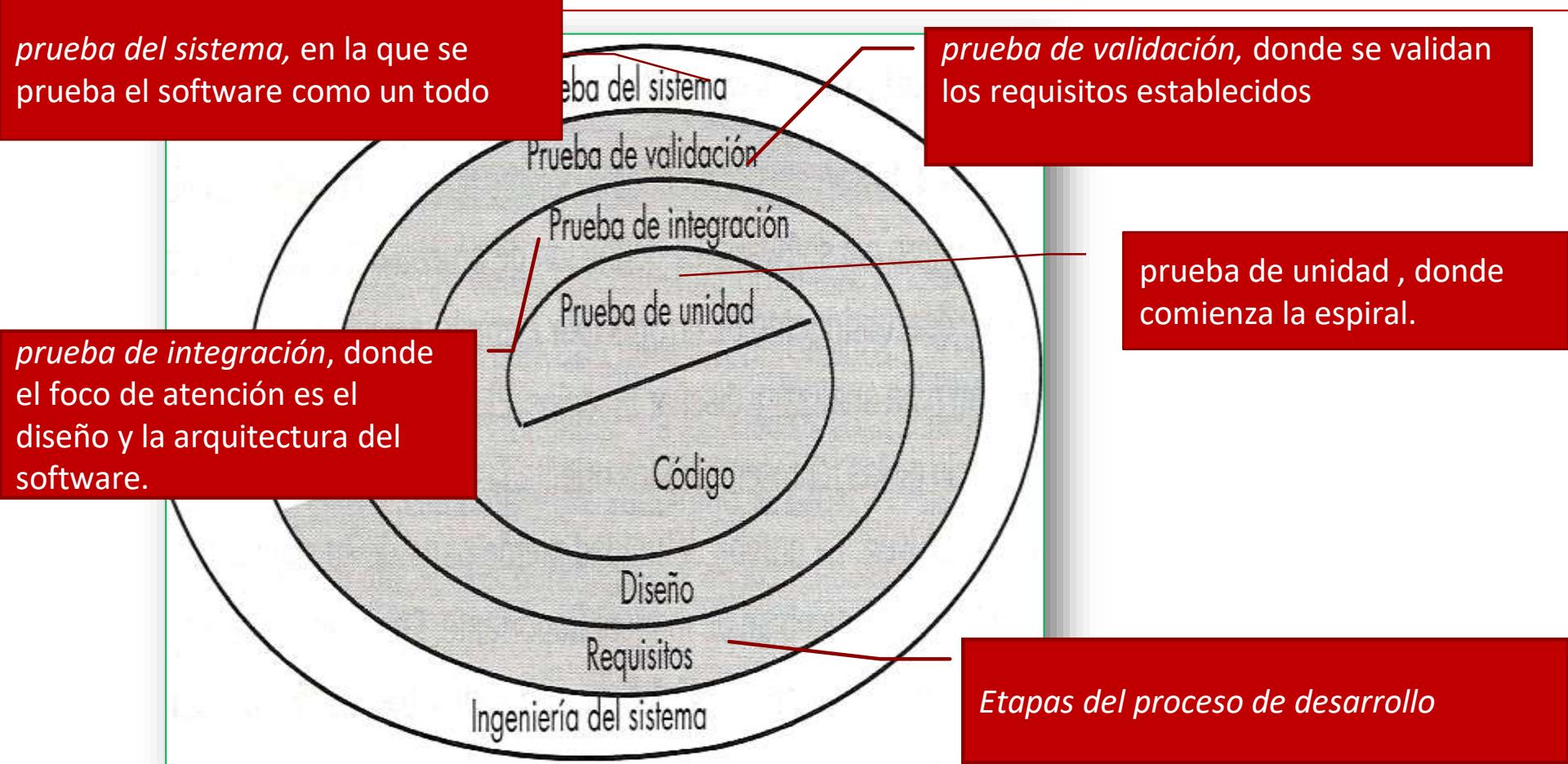
Estrategias de pruebas

prueba del sistema, en la que se prueba el software como un todo

prueba de validación, donde se validan los requisitos establecidos

prueba de integración, donde el foco de atención es el diseño y la arquitectura del software.

prueba de unidad , donde comienza la espiral.

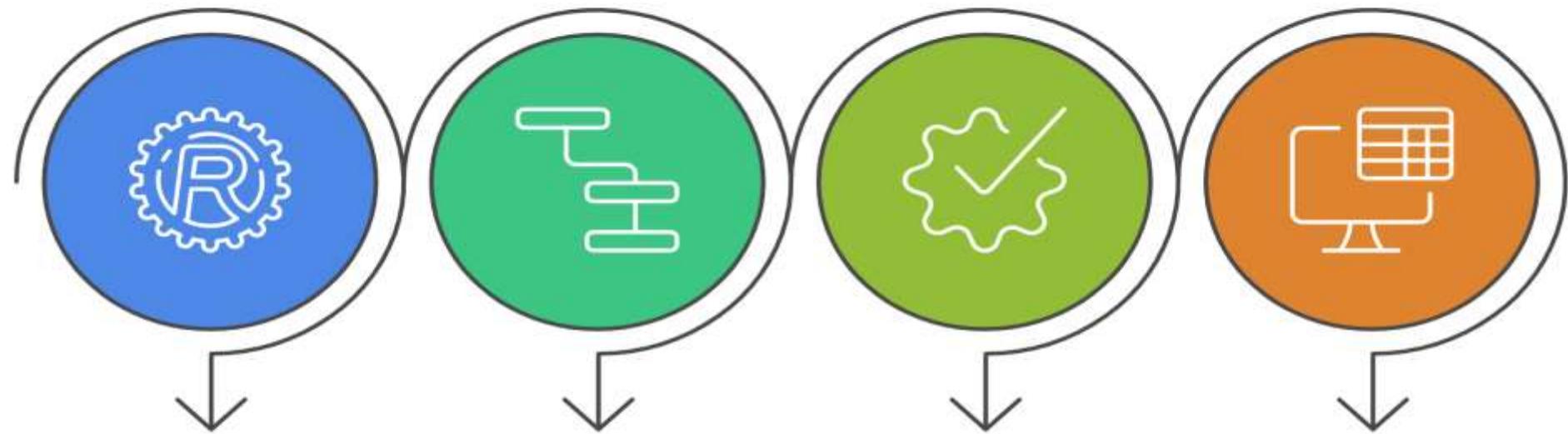


5

Tipos de Pruebas Software convencionales



Tipos de pruebas de software



Pruebas de unidad

El componente verifica el correcto funcionamiento del componente

Pruebas de integración

Verificación de que los componentes trabajan juntos correctamente

Pruebas de validación

El software satisface los requisitos funcionales y no funcionales

Prueba del sistema

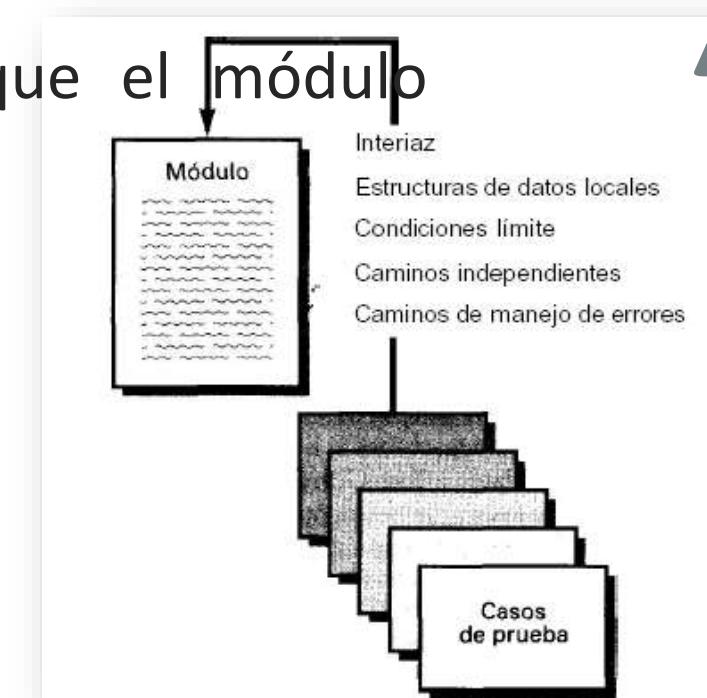
Verificación de que cada elemento encaja de forma adecuada

6

Tipos de Pruebas. Pruebas de Unidad

- » Se prueba la interfaz del módulo para asegurar que la información fluye de forma adecuada.
- » Se examinan las estructuras de datos locales.
- » Se prueban las condiciones límite para asegurar que el módulo funciona correctamente
- » Se ejercitan todos los caminos independientes.

7



Tipos de Pruebas. Pruebas de Unidad

» Los errores más comunes detectados por la pruebas de unidad:
Cálculos incorrectos

- ❖ *Aplicación incorrecta de predecesores aritméticos*
- ❖ *Operaciones mezcladas*
- ❖ *Inicialización incorrecta*
- ❖ *Falta de precisión*
- ❖ *Representación simbólica incorrecta*

Comparaciones erróneas

Flujos de control inapropiados

Tipos de Pruebas. Pruebas de Unidad

» Los Casos de prueba deben descubrir errores como:

- Comparaciones entre diferentes tipos de datos
- Operadores lógicos aplicados incorrectamente
- Expectativas de igualdad con grado de precisión
- Comparación incorrecta de variables
- Terminación inapropiada o inexistente de bucles
- Falla en la salida cuando se encuentre una iteración divergente
- Variables de bucle modificadas inapropiadamente

Tipos de Pruebas. Pruebas de Unidad - Procedimiento

- » Como un componente no es un programa independiente, se debe desarrollar para cada prueba de unidad un software que controle y/o resguarde.
- » Un controlador es un «programa principal» que acepta los datos del caso de prueba, pasa estos datos al módulo (a ser probado) y muestra los resultados.

10



Tipos de Pruebas.

Pruebas de Unidad - Procedimiento

- » Un resguardo sirve para reemplazar a módulos subordinados al componente que hay que probar.
- » Los controladores y resguardos son una sobrecarga de trabajo.
- » Si los controladores y resguardos son sencillos, el trabajo adicional es relativamente pequeño.
- » La prueba de unidad se simplifica cuando se diseña un módulo con un alto grado de cohesión.

11

Tipos de Pruebas. Pruebas de Integración

- » Se toman los componentes que han pasado las pruebas de unidad y se los combina según el diseño establecido.
- » En esta combinación es posible que:
 - Los datos se pueden perder en una interfaz.
 - Un módulo puede tener un efecto adverso e inadvertido sobre otro.
 - La combinación de subfunciones no produzca el resultado esperado.

12

Tipos de Pruebas. Pruebas de Integración

- » El programa se construye y se prueba en pequeños segmentos en los que los errores son más fáciles de aislar y de corregir
- » La Integración puede ser :

Descendente

Ascendente

13

Tipos de Pruebas.

Pruebas de Integración - *Descendente*

- » Los módulos se integran al descender por la jerarquía de control, iniciando por el programa principal

- » Se puede realizar:

- En profundidad

Primero-en-profundidad integra todos los módulos de un camino de control principal de la estructura.

- En anchura

Primero-en-anchura incorpora todos los módulos directamente subordinados a cada nivel

14

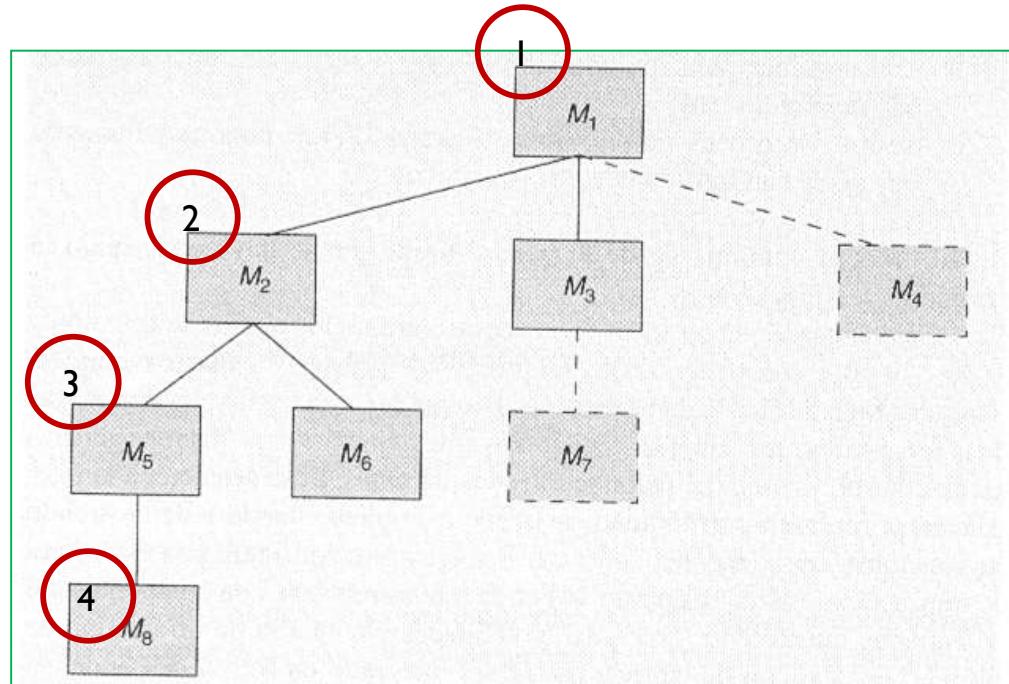
Tipos de Pruebas.

Pruebas de Integración - *Descendente*

» En profundidad: primero-en-profundidad integra todos los módulos de un camino de control principal de la estructura.

Pasos:

1. Conductor: Módulo principal
Resguardos: Para los módulos subordinados
2. Sustituir resguardos por módulos I a I
3. Probar
4. Reemplazar otro resguardo
5. Pruebas de regresión



15

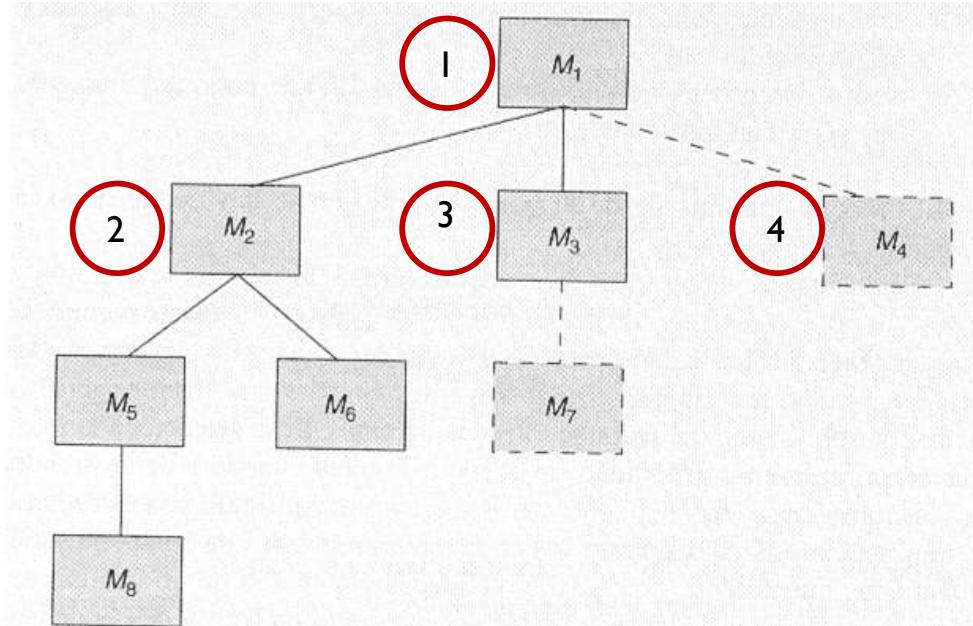
Tipos de Pruebas.

Pruebas de Integración - *Descendente*

»En anchura: primero-en-anchura incorpora todos los módulos directamente subordinados a cada nivel.

Pasos:

1. Conductor: Módulo principal
Resguardos: Para los módulos subordinados
2. Sustituir resguardos por módulos I a I
3. Probar
4. Reemplazar otro resguardo
5. Pruebas de regresión



16

Tipos de Pruebas.

Pruebas de Integración - Ascendente



Se empieza la prueba con los módulos atómicos (es decir, módulos de los niveles más bajos de la estructura del programa).

17



Dado que los módulos se integran de abajo hacia arriba, el proceso requerido de los módulos subordinados siempre está disponible y se elimina la necesidad de resguardos pero no así, los conductores.

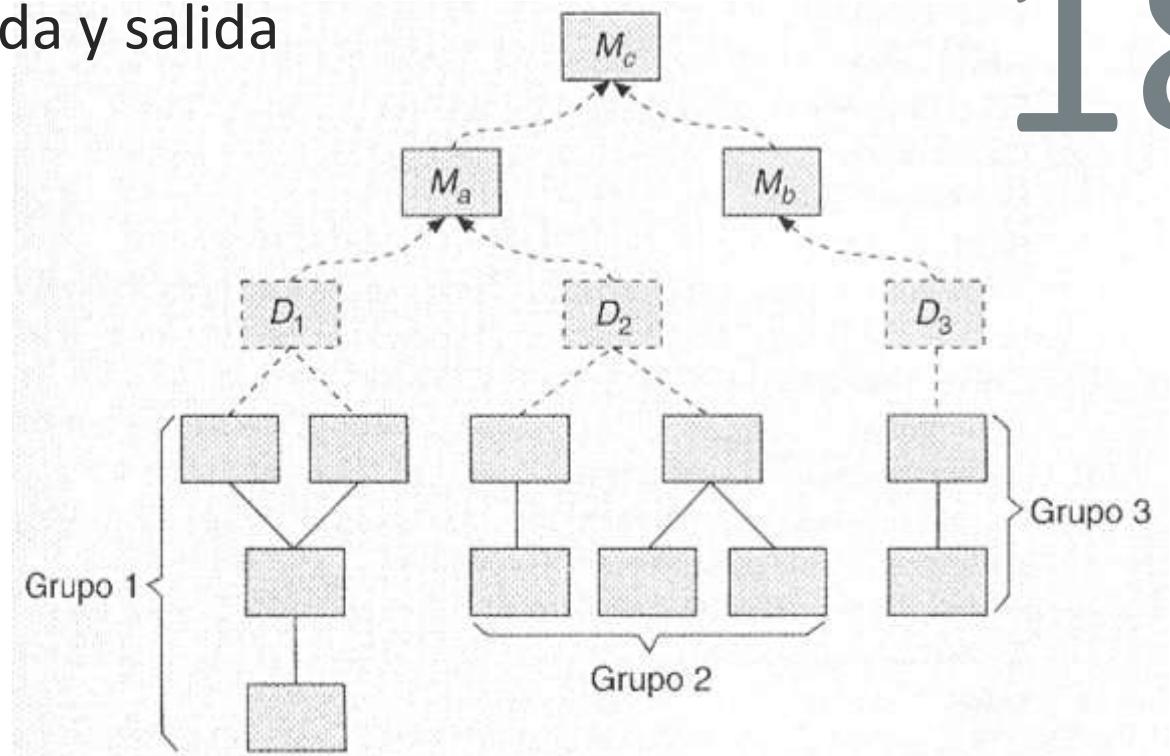
Tipos de Pruebas.

Pruebas de Integración - Ascendente

Pasos :

- » 1. Combinar módulos de bajo nivel
- » 2. Hacer conductor para coordinar entrada y salida
- » 3. Probar el grupo
- » 4. Eliminar conductores

18



Fuente:

2025

Tipos de Pruebas.

Pruebas de integración - Selección

Hay discusión respecto a las ventajas y desventajas de los enfoques Ascendente con Descendente.

La principal desventaja del enfoque Descendente es la necesidad de los resguardos.

La principal desventaja de la opción Ascendente es que “el programa como entidad no existe hasta que se agrega el último módulo”.

La elección depende de las características del software, Un enfoque combinado (*prueba sándwich*) puede ser el mejor.

19

Tipos de Pruebas.

Pruebas de integración - *Pruebas de regresión*

- » Cada vez que se añade un nuevo módulo como parte de una Prueba de integración, el software cambia.
- » Se establecen nuevos caminos, pueden ocurrir nuevas E/S y se invoca una nueva lógica de control.
- » Estos cambios pueden causar problemas con funciones que antes trabajaban perfectamente.
- » En el contexto de una estrategia de Prueba de integración, la Prueba de regresión es volver a ejecutar un subconjunto de pruebas que se han llevado a cabo anteriormente para asegurarse de que los cambios no han propagado efectos colaterales no deseados.

20

Tipos de Pruebas.

Pruebas de integración - *Pruebas de regresión*

- » Estas pruebas se pueden hacer manualmente, volviendo a realizar un subconjunto de todos los casos de prueba o utilizando herramientas automáticas.
- » El conjunto de pruebas de regresión contiene tres clases diferentes de casos de prueba:
 - una muestra representativa de pruebas que ejercite todas las funciones del software.
 - pruebas adicionales que se centren en las funciones del software que son probablemente afectadas por el cambio.
 - pruebas que se centren en los componentes del software que han cambiado.

21

Tipos de Pruebas.

Pruebas de integración - *Criticidad*

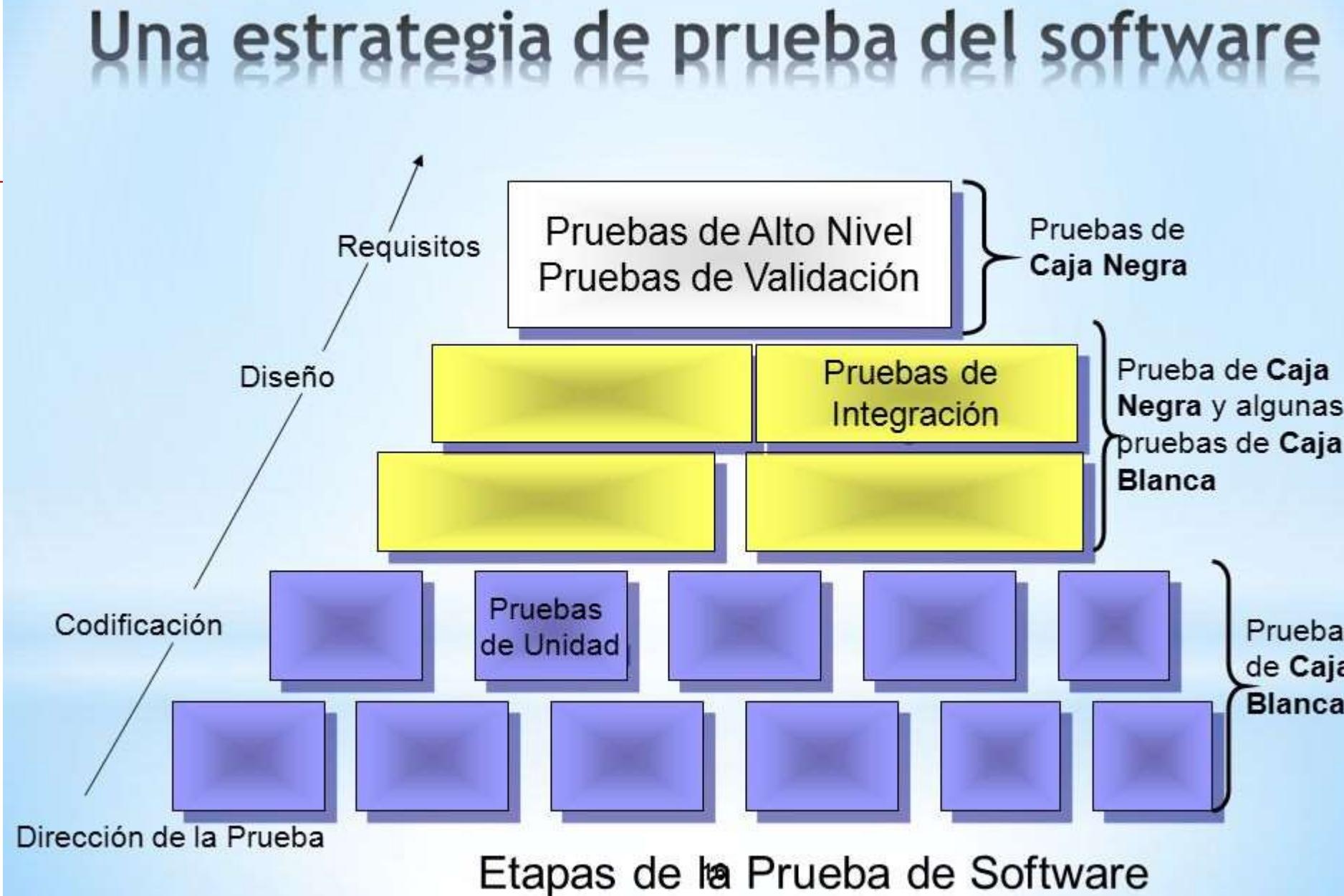
Se deben identificar los módulos críticos, que pueden ser los que:

1. *Abordan muchos requerimientos de software*
2. *Tienen alto nivel de control*
3. *Es complejo o proclive a error*
4. *Tiene requerimientos de rendimientos definidos.*

22

Deben probarse lo antes posible. Las pruebas de regresión deben hacer foco en ellos.

Possible estrategia de pruebas de integración



Tipos de Pruebas. Pruebas de Unidad e Integración para software OO

» Prueba de Unidad :

Por lo general una clase encapsulada es el foco de la prueba de unidad.

Los métodos son las unidades comprobables más pequeñas.

La prueba de clase es el equivalente en este caso, la cual debe ser dirigida a las operaciones encapsuladas por la clase y el comportamiento de estado de ésta.

24

» Prueba de integración:

El software OO no tiene una estructura de control jerárquico obvia.

La prueba basada en hebra integra el conjunto de clases requeridas para responder a una entrada o evento.

La prueba basada en uso comienza con las clases independientes, luego las dependientes.

Tipos de Pruebas. Pruebas del Sistema

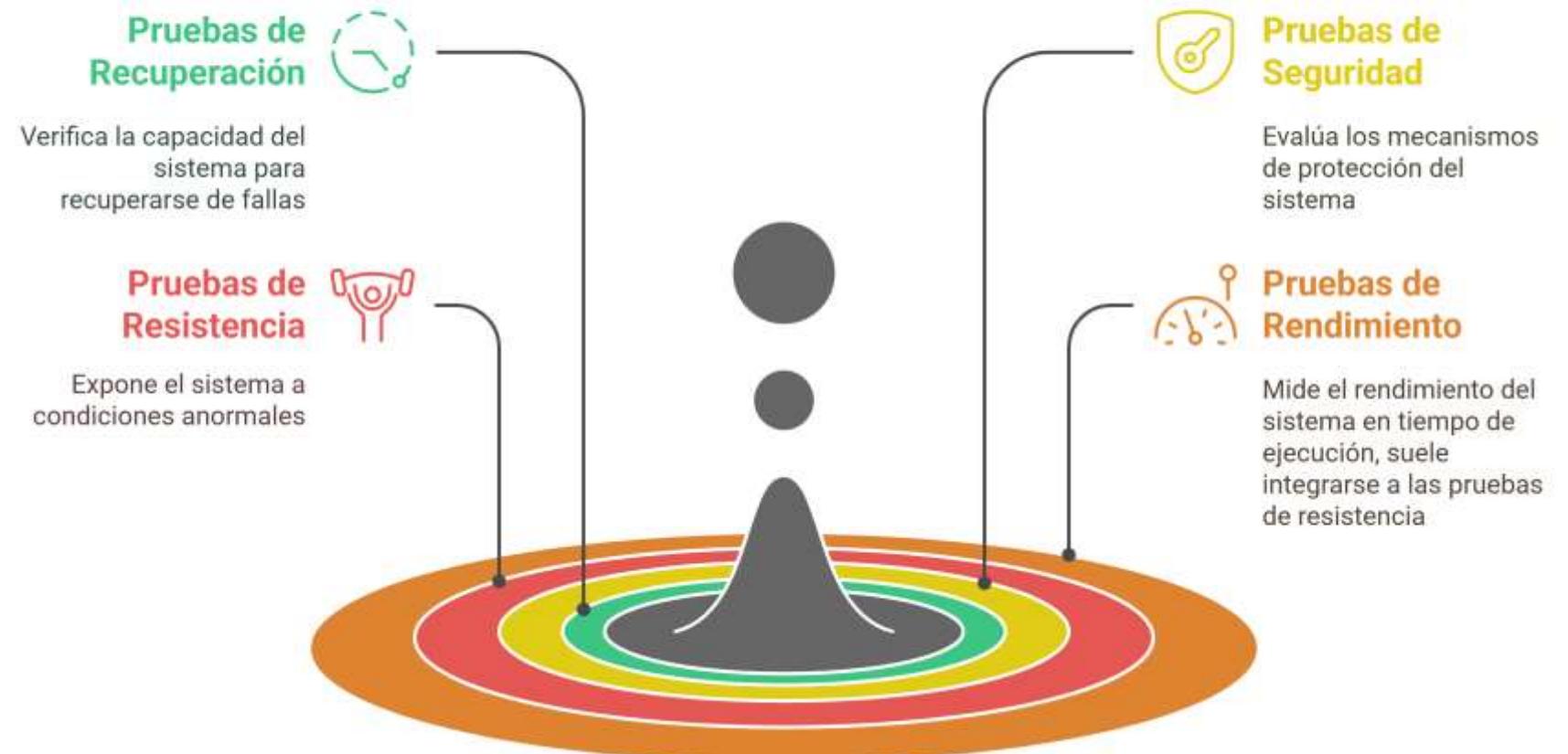
- » La prueba del sistema, está constituida por una serie de pruebas diferentes.
- » Aunque cada prueba tiene un propósito diferente, todas trabajan para verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las funciones apropiadas.

25

Tipos de Pruebas. Pruebas del Sistema



Tipos de Pruebas de Sistema



Tipos de Pruebas. Pruebas de Validación

- » La validación del software se consigue mediante una serie de pruebas que demuestren la conformidad con los requisitos.
- » Una vez que se procede con cada caso de prueba de validación, puede darse una de las dos condiciones:

27

Las características de funcionamiento o de rendimiento están de acuerdo con las especificaciones y son aceptables;

O

Se descubre una desviación de las especificaciones y se crea una lista de deficiencias.

Tipos de Pruebas. Pruebas de Validación

- » Comienzan cuando finalizan las pruebas de integración.
- » Revisión de la configuración
 - Asegurar que todos los elementos de la configuración del software se hayan desarrollado apropiadamente, estén catalogados y contengan detalle suficiente para reforzar la fase de soporte.

28

Tipos de Pruebas. Pruebas de Validación

» Pruebas de aceptación (ALFA y BETA)

Las realiza el usuario final en lugar del responsable del desarrollo del sistema, una prueba de aceptación puede ir desde algo informal, hasta la ejecución sistemática de una serie de pruebas bien planificadas.

Dentro de las Pruebas de aceptación se pueden encontrar:

Pruebas ALFA: desarrolladores con clientes antes de liberar el producto.

Pruebas BETA: seleccionando los clientes que efectuarán la prueba. El desarrollador no se encuentra presente.

29

Tipos de Pruebas. Pruebas de Validación – aceptación ALFA

- » Se llevan a cabo, por un cliente, en el lugar de desarrollo.
- » Se usa el software de forma natural con el desarrollador como observador del usuario y registrando los errores y problemas de uso.
- » Las pruebas alfa se hacen en un entorno controlado.
- » Se realizan después de que todos los procedimientos de prueba básicos, como las pruebas unitarias y pruebas de integración se han completado, y se produce después de las pruebas del sistema.
- » Esta no es la versión final de software y cierta funcionalidad puede ser añadido al software incluso después de la prueba alfa.

30

Tipos de Pruebas.

Pruebas de Validación – aceptación BETA

- » Se llevan a cabo por los usuarios finales del software en los lugares de trabajo de los clientes.
- » El desarrollador no está presente normalmente. Así, la prueba beta es una aplicación en vivo del software en un entorno que no puede ser controlado por el desarrollador.
- » El cliente registra todos los problemas que encuentra durante la prueba beta e informa a intervalos regulares al desarrollador.
- » Las pruebas beta es la última fase de las fases de prueba y se hace utilizando técnicas de caja negra.
- » A veces la versión beta también es liberada en el mercado, y en base a las modificaciones que se hacen comentarios de los usuarios o si no hay cambios en el software se libera.

31

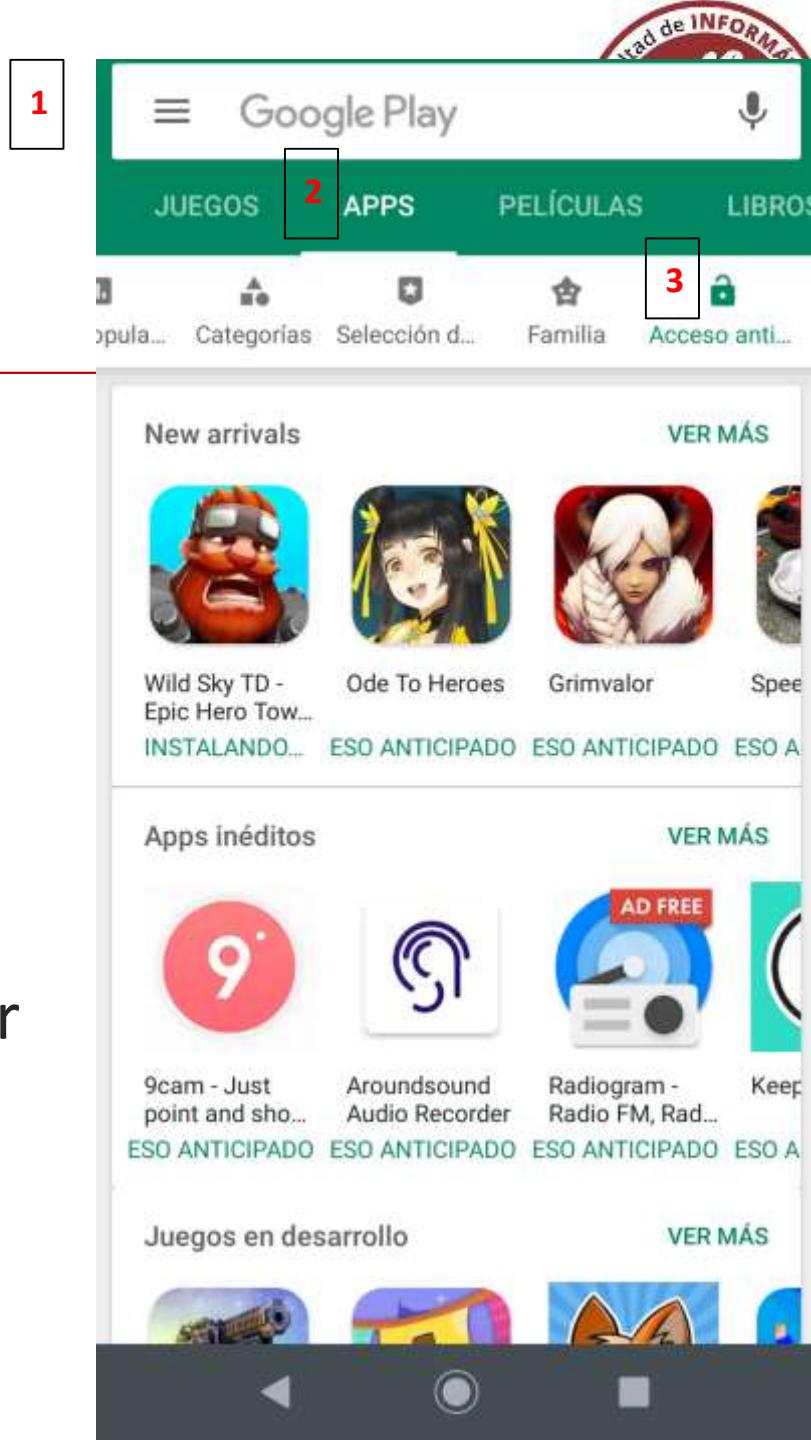
Tipos de Pruebas.

Pruebas BETA

Beta testers de App de Android

¿Cómo hacer?

1. Ir a Play Store
2. Buscar pestaña de Apps
3. Buscar opción Acceso Anticipado, (está al final)
4. Descargar la app que se quiera probar
5. Se habilitan mensajes privados con el desarrollador



Tipos de Pruebas.

Pruebas de Validación – aceptación BETA

» Ejemplo de prueba beta:

<https://developer.apple.com/testflight/>

» La explicación de cómo desarrollar una prueba beta de pre lanzamiento de una App para Apple Store se puede ver en:

https://developer.apple.com/library/content/documentation/LanguagesUtilities/Conceptual/iTunesConnect_Guide/ES/Chapters/BetaTestingTheApp.html

33

Tipos de Pruebas.

Pruebas de Validación – aceptación BETA

»Lugares donde pueden ser beta tester:

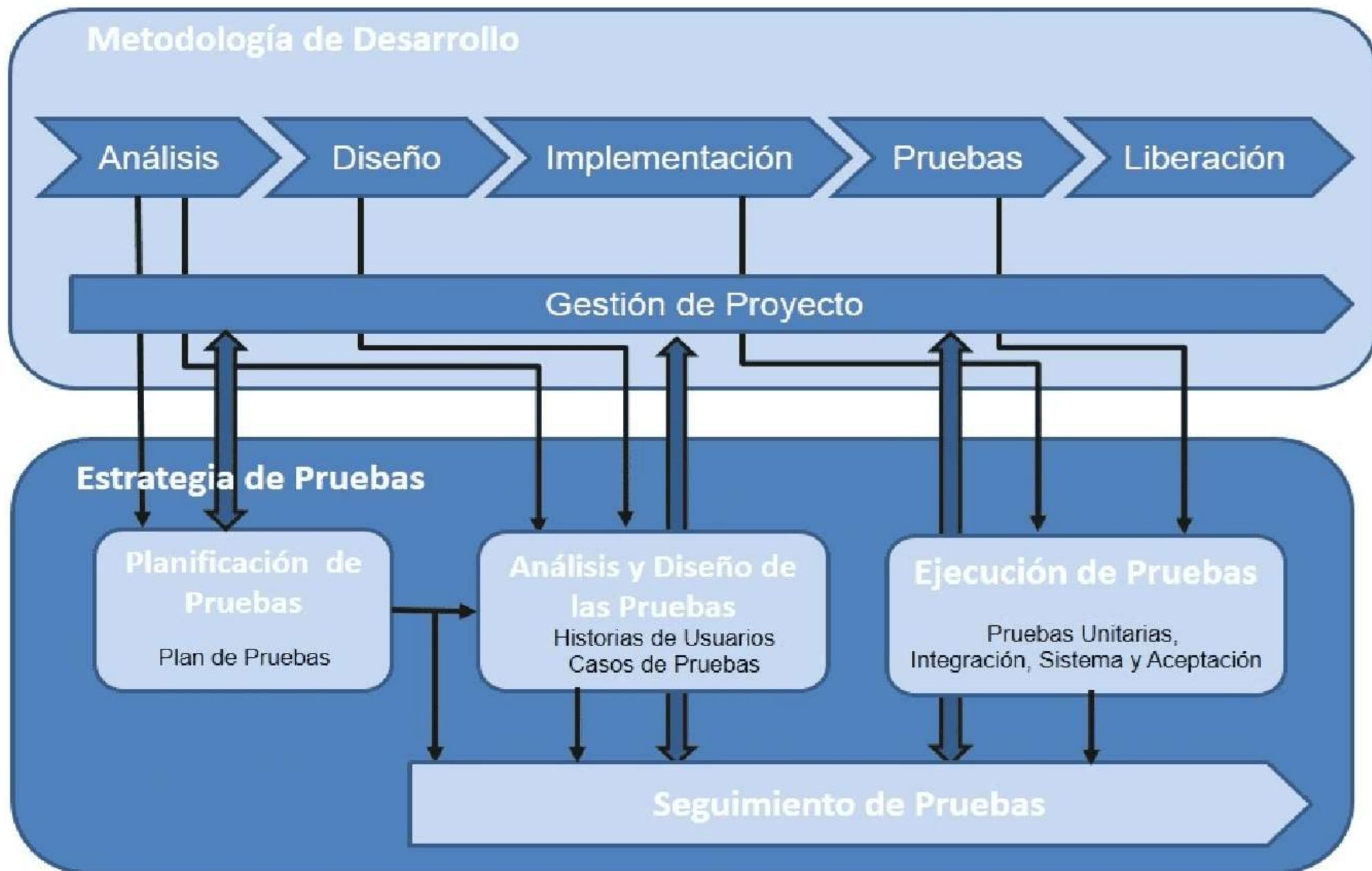
<https://www.usertesting.com/>

34

<https://www.utest.com/>

[Beta Testers: Únete a la comunidad de BetaTesting hoy](#)

Una estrategia posible de pruebas



Prueba de entornos especializados

- » A medida que el software se hace más complejo, crece también la necesidad de enfoques de pruebas especializados.
- » Pruebas de interfaces gráficas
- » Pruebas de arquitecturas cliente-servidor
- » Pruebas de la documentación y ayuda
- » Pruebas de sistema en tiempo real

36

Prueba de entornos especializados

Prueba de arquitectura cliente-servidor

- » Pruebas de funcionalidad de la aplicación
- » Prueba de servidor
 - Probar las funciones de coordinación y manejo de datos del servidor.
 - Desempeño del servidor (tiempo de respuesta y procesamiento total de los datos)
- » Prueba de base de datos
 - Probar la exactitud e integridad de los datos, examinar transacciones, asegurar que se almacenan, actualizan y recuperan los datos.
- » Pruebas de transacciones
 - Se crea una serie de pruebas para asegurar que cada transacción se procese de acuerdo a los requisitos.
- » Pruebas de comunicación de red
 - Verificar comunicación entre los nodos, que el paso de mensajes, transacciones y tráfico de la red se realice sin errores.

37

Prueba de entornos especializados

Prueba de la documentación y funciones de ayuda

- » Es importante para la aceptación del programa.
- » Revisar la guía del usuario o funciones de ayuda en línea.
- » Prueba de documentación es en dos fases:

Revisar e inspeccionar

examinar la claridad editorial del documento.

Prueba en vivo

usar la documentación junto con el programa real.

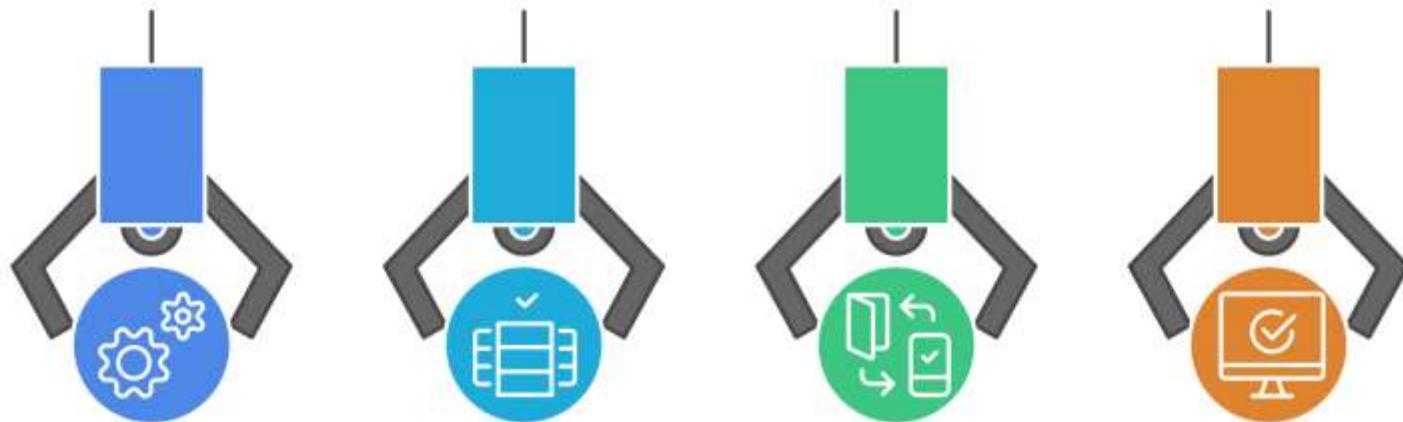
38

Pruebas de sistemas de tiempo real



El diseño de los casos de prueba, además de los convencionales deben incluir manejo de eventos (interrupciones), temporización de los datos, el paralelismo entre las tareas, etc.

Pruebas de sistemas de tiempo real



Pruebas de tareas

Probar las tareas de forma independiente, en búsqueda de errores lógicos

Pruebas de comportamiento

Simular el comportamiento del sistema de tiempo real y analizarlo como consecuencia de eventos.

Pruebas inter-tareas

Se prueban las tareas asincrónicas entre las cuales se sabe que hay comunicación

Pruebas de sistemas

Se prueba el software y hardware integrados.



DEPURACIÓN de Programas

Depuración



La depuración de programas, es el proceso de identificar y corregir errores en programas informáticos.



La depuración no es una prueba, pero siempre ocurre como consecuencia de la prueba efectiva.

Es decir, se descubre un error, la depuración elimina dicho error.

41

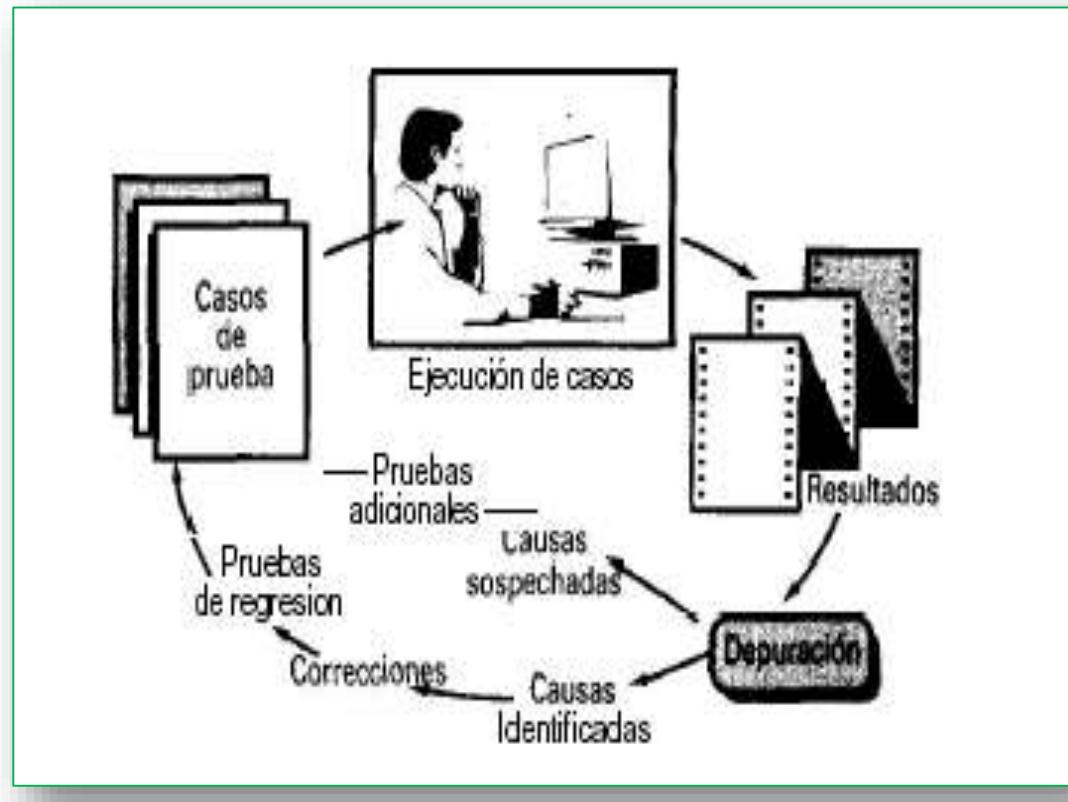
El Proceso de Depuración

» El proceso de depuración siempre tiene uno de los dos resultados:

Se encuentra la causa, se corrige y se elimina.

o

No se encuentra la causa. La persona que realiza la depuración debe sospechar la causa, diseñar un caso de prueba que ayude a confirmar sus sospechas y el trabajo vuelve hacia atrás a la corrección del error de una forma iterativa.



El Proceso de Depuración

Características de los errores

1. Síntoma lejano (geográficamente) de la causa
2. Síntoma desaparece temporalmente al corregir otro error
3. Síntoma producido por error
4. Síntoma causado por error humano
5. Síntoma causado por problemas de tiempo
6. Condiciones de entrada difíciles de reproducir
7. Síntoma intermitente (especialmente en desarrollos hardware-software)
8. El síntoma se debe a causas distribuidas entre varias tareas que se ejecutan en diferentes procesadores

43

Enfoques de la Depuración

Proceso de Depuración de Programas





Ingeniería de software II

Mantenimiento

¿Porqué surge el mantenimiento?

2

- » Éste comienza casi de inmediato. El software se libera a los usuarios finales y, en cuestión de días, los reportes de errores se filtran de vuelta hacia la organización de ingeniería de software.
- » En semanas, una clase de usuarios indica que el software debe cambiarse de modo que pueda ajustarse a las necesidades especiales de su entorno.
- » Y en meses, otro grupo corporativo, que no quería saber nada del software cuando se liberó, ahora reconoce que puede ofrecerle beneficios inesperados. Pero...necesitará algunas mejoras para hacer que funcione en su mundo.

Una hipótesis

“Mucho del software del que dependemos en la actualidad tiene en promedio una antigüedad de 10 a 15 años. Aun cuando dichos programas se crearon usando las mejores técnicas de diseño y codificación conocidas en la época [y muchas no lo fueron], se produjeron cuando el tamaño del programa y el espacio de almacenamiento eran las preocupaciones principales.

Luego migraron a nuevas plataformas, se ajustaron para cambios en máquina y tecnología de sistema operativo, y aumentaron para satisfacer las necesidades de los nuevos usuarios, todo sin suficiente preocupación por la arquitectura global. El resultado es estructuras pobremente diseñadas, pobre codificación, pobre lógica y pobre documentación de los sistemas de software que ahora debemos seguir usando...”

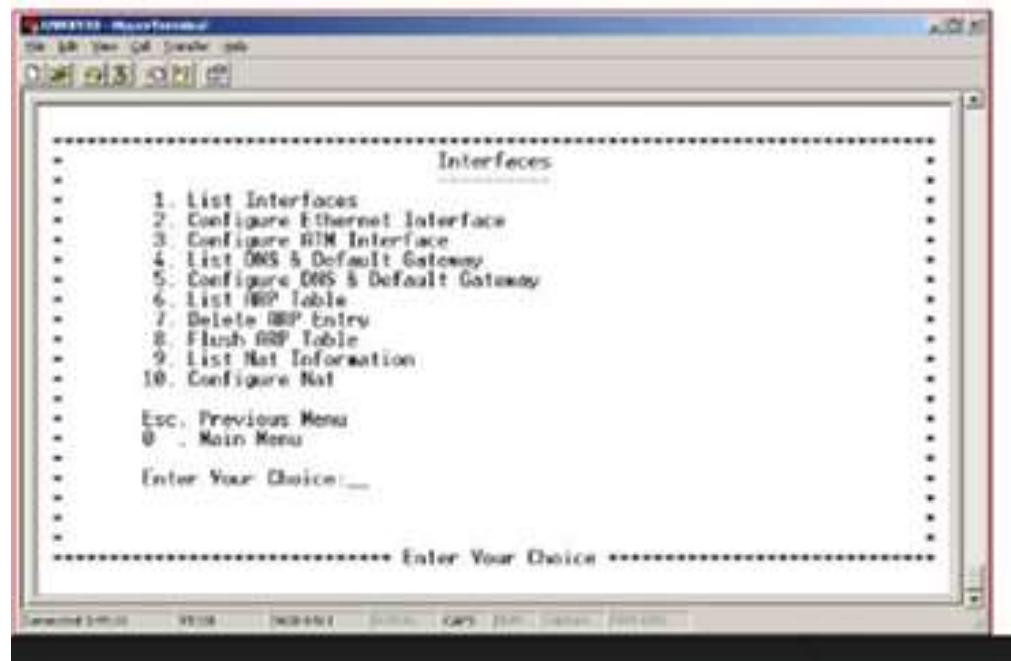
3

Osborne y Chikofsky [Osb90]

Características del Mantenimiento

En general las características de los sistemas son:

- ✓ Viejos.
- ✓ Sin metodología ni documentación.
- ✓ Sin modularidad.



Mantenimiento

»Atención del sistema a lo largo de su evolución después que el sistema se ha entregado.

El mantenimiento de software es el proceso de modificar, actualizar y cambiar el software para satisfacer las necesidades del cliente. Es una actividad amplia que incluye: corregir errores, mejorar las capacidades, eliminar funciones obsoletas y optimizar otras.

»A esta fase se la llama “Evolución del Sistema”.

»En ocasiones debe realizarse mantenimiento a sistemas “heredados”. ¿Qué problemas podemos encontrar en los sistemas heredados?

Mantenimiento

»Es necesario evaluar cuándo es conveniente cerrar el ciclo de vida de ese sistema y reemplazarlo por otro.

6

La decisión se toma en función del costo del ciclo de vida del viejo proyecto y la estimación del nuevo proyecto

En ocasiones la complejidad del sistema crece por los cambios.

Dinámica de la evolución de los programas

7

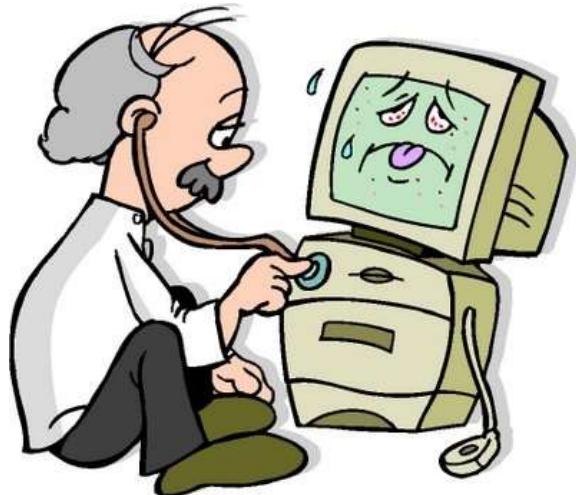
- » Manny Lehman y sus colaboradores realizaron análisis detallados de software de grado industrial y de sistemas en general con la intención de desarrollar una “teoría unificada para evolución del software”.

Leyes de Lehman

Cambio continuado	Un programa que se usa en un entorno real necesariamente debe cambiar o se volverá progresivamente menos útil en ese entorno.
Complejidad creciente	A medida que un programa en evolución cambia, su estructura tiende a ser cada vez más compleja. Se deben dedicar recursos extras para preservar y simplificar la estructura.
Evolución prolongada del programa	La evolución de los programas es un proceso autorregulativo. Los atributos de los sistemas, tales como tamaño, tiempo entre entregas y el número de errores documentados, son aproximadamente invariantes para cada entrega del sistema.
Estabilidad organizacional	Durante el tiempo de vida de un programa, su velocidad de desarrollo es aproximadamente constante e independiente de los recursos dedicados al desarrollo del sistema.
Conservación de la familiaridad	Durante el tiempo de vida de un sistema, el cambio incremental en cada entrega es aproximadamente constante.
Crecimiento continuado	La funcionalidad ofrecida por los sistemas tiene que crecer continuamente para mantener la satisfacción de los usuarios.
Decremento de la calidad	La calidad de los sistemas comenzará a disminuir a menos que dichos sistemas se adapten a los cambios en su entorno de funcionamiento.
Realimentación del sistema	Los procesos de evolución incorporan sistemas de realimentación multiagente y multibucle y éstos deben ser tratados como sistemas de realimentación para lograr una mejora significativa del producto.

Mantenimiento

- » Solucionar errores
 - » Añadir mejoras
 - » Optimizar
- » Esto provoca altos costos adicionales



9

EL FENÓMENO DE LA **"BARRERA DE MANTENIMIENTO"**

Mantenimiento - Características

- » Su consecuencia es la disminución de otros desarrollos.
- » Pueden existir efectos secundarios sobre código, datos, documentación.
- » Las modificaciones pueden provocar disminución de la calidad total del producto.
- » Las tareas de mantenimiento generalmente provocan reiniciar las fases de análisis, diseño e implementación.
- » Involucra entre un 40% a 70% del costo total de desarrollo.
- » Los errores provocan insatisfacción del cliente.

10

Mantenimiento - ¿Por qué es problemático?

- » No es un trabajo atractivo
- » No siempre en el diseño se prevén los cambios
- » Es difícil comprender código ajeno, más aún sin documentación o con documentación inadecuada

11



Actividades de Mantenimiento

Debe utilizarse un mecanismo para realizar los cambios que permita: identificarlos, controlarlos, implementarlos e informarlos

12

El proceso de cambio se facilita si en el desarrollo están presentes los atributos calidad como , modularidad, documentación interna del código fuente y de apoyo

Mantenimiento – Ciclo de mantenimiento

»Análisis:

comprender el alcance y el efecto de la modificación

»Diseño:

rediseñar para incorporar los cambios

»Implementación:

recodificar y actualizar la documentación interna del código

»Prueba:

revalidar el software

»Actualizar la documentación de apoyo

»Distribuir e instalar las nuevas versiones



13

Facilidades en el desarrollo para ayudar al mantenimiento

» Análisis:

Señalar principios generales, armar planes temporales, especificar controles de calidad, identificar posibles mejoras, estimar recursos para mantenimiento

» Diseño arquitectónico:

Claro, modular, modificable, con notaciones estandarizadas

» Diseño detallado:

Notaciones para algoritmos y estructuras de datos, especificación de interfaces, manejo de excepciones, efectos colaterales

» Implementación:

Indentación, comentarios de prólogo e internos, codificación simple y clara

» Verificación:

Lotes de prueba y resultados

¿Quién realiza el mantenimiento?

- » El equipo que desarrolla un sistema no siempre es el que se utiliza para mantener el sistema una vez que esté operativo.
- » A menudo, un equipo de mantenimiento independiente se emplea para garantizar que el sistema funcione correctamente.
- » El equipo de mantenimiento involucra a los usuarios, operadores o representantes del cliente se acercan al equipo de mantenimiento con un comentario o problema. Los analistas o programadores determinan qué partes del código se ve afectado, el impacto en el diseño y los recursos probables (incluido el tiempo y esfuerzo) para realizar los cambios necesarios.

15

Tareas del equipo de mantenimiento

- ✓ Comprender el sistema
- ✓ Localizar información en la documentación del sistema
- ✓ Mantener actualizada la documentación del sistema
- ✓ Ampliar las funciones existentes para adaptarse a requisitos nuevos o cambiantes
- ✓ Agregar nuevas funciones al sistema
- ✓ Encontrar la fuente de fallas o problemas del sistema
- ✓ Localizar y corregir fallos
- ✓ Responder preguntas sobre la forma en que funciona el sistema
- ✓ Reestructuración del diseño y los componentes del código
- ✓ Rescribir componentes de diseño y código
- ✓ Eliminar componentes de diseño y código que ya no son útiles
- ✓ Gestionar los cambios en el sistema a medida que se realizan

16

Tipos de Mantenimiento



Mantenimiento correctivo

- Diagnóstico y corrección de errores.

Mantenimiento adaptativo

- *Modificación del software para interaccionar correctamente con el entorno.*

Mantenimiento perfectivo

- *Mejoras al sistema.*

Mantenimiento preventivo

- *Se efectúa antes que haya una petición, para facilitar el futuro mantenimiento. Se aprovecha el conocimiento sobre el producto.*

17

Mantenimiento correctivo

- » El mantenimiento correctivo de software es una acción reactiva que se realiza cuando se detecta un fallo o error en una pieza de software. El objetivo es restablecer el funcionamiento óptimo del sistema y minimizar el impacto del problema en la producción.
- » Suele realizarse lo más rápido posible. Los encargados del mantenimiento deben detectar la causa del fallo, decidir si se puede reparar o no y aplicar la solución elegida.

18

Mantenimiento adaptativo

- » El Mantenimiento adaptativo de software tiene que ver con las tecnologías cambiantes, así como con las políticas y reglas relacionadas con su software.
- » Las cuales incluyen cambios en el sistema operativo, almacenamiento en la nube, hardware, etc. Cuando se realizan estos cambios, su software debe adaptarse para cumplir adecuadamente los nuevos requisitos y continuar funcionando bien

19

Mantenimiento perfectivo

- » El mantenimiento perfectivo de software es un tipo de mantenimiento que se realiza para mejorar la eficiencia o rendimiento de un sistema. Este mantenimiento se centra en características que mejoran la experiencia del usuario a través de mejoras funcionales, en respuesta a los comentarios de los clientes.
- » Los usuarios pueden ver la necesidad de nuevas características o requisitos que les gustaría ver en el software para convertirlo en la mejor herramienta disponible para sus necesidades. El mantenimiento perfectivo de software tiene como objetivo ajustar el software agregando nuevas características según sea necesario y eliminando características que son irrelevantes o no efectivas en el software dado. Este proceso mantiene el software relevante a medida que el mercado y las necesidades del usuario cambian.

20

Mantenimiento preventivo

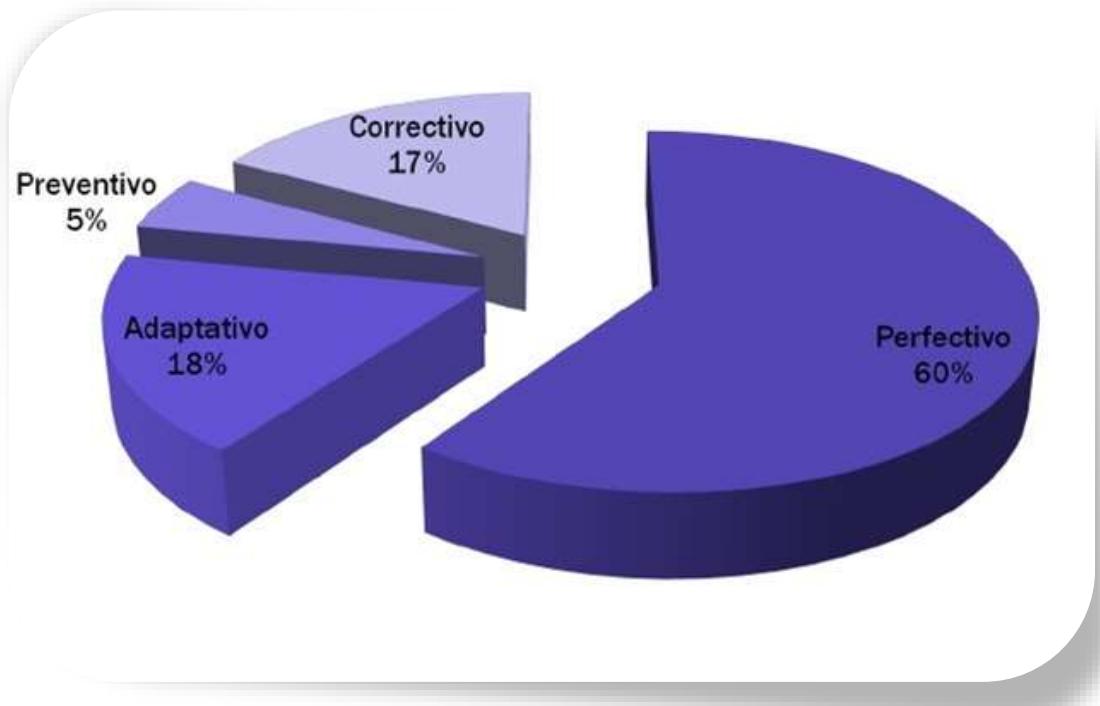
El mantenimiento preventivo de software es un conjunto de actividades planificadas y realizadas regularmente para prevenir posibles fallos en el futuro. Estas actividades incluyen:

- » Realizar cambios necesarios
- » Actualizaciones
- » Prevenir fallas en el sistema operativo, antivirus o programas ofimáticos
- » Instalar y configurar un cortafuego para impedir accesos no autorizados de terceros
- » Programas antimalware para impedir que el equipo se infecte con malware
- » Activar y configurar los puntos de restauración del sistema para poder volver a un estado anterior en caso de algún desastre

21

Mantenimiento

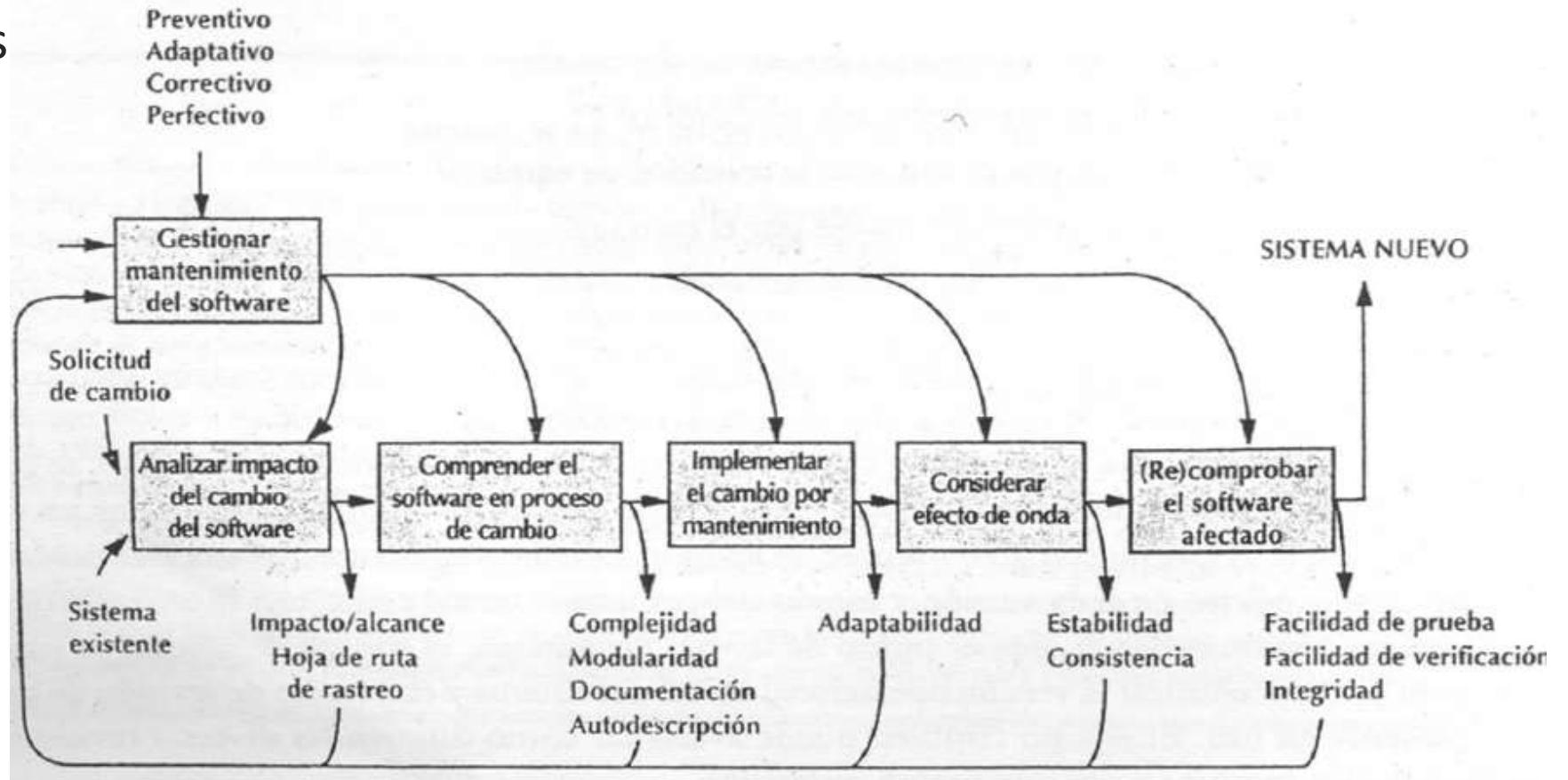
»Tipos de Mantenimiento que se realiza en un software



22

Mantenimiento

» Actividades



23

Herramientas para realizar mantenimiento

- » Editores de texto
- » Comparadores de archivos (<https://winmerge.org/>)
- » Compiladores y linkeadores
- » Debuggers
- » Analizadores de código estático
(<https://kinsta.com/es/blog/herramientas-de-revision-de-codigo/>)
- » Repositorios de gestión de configuración (Git, Docker, Terraform, Ansible, SaltStack, Chef, Puppet)

24

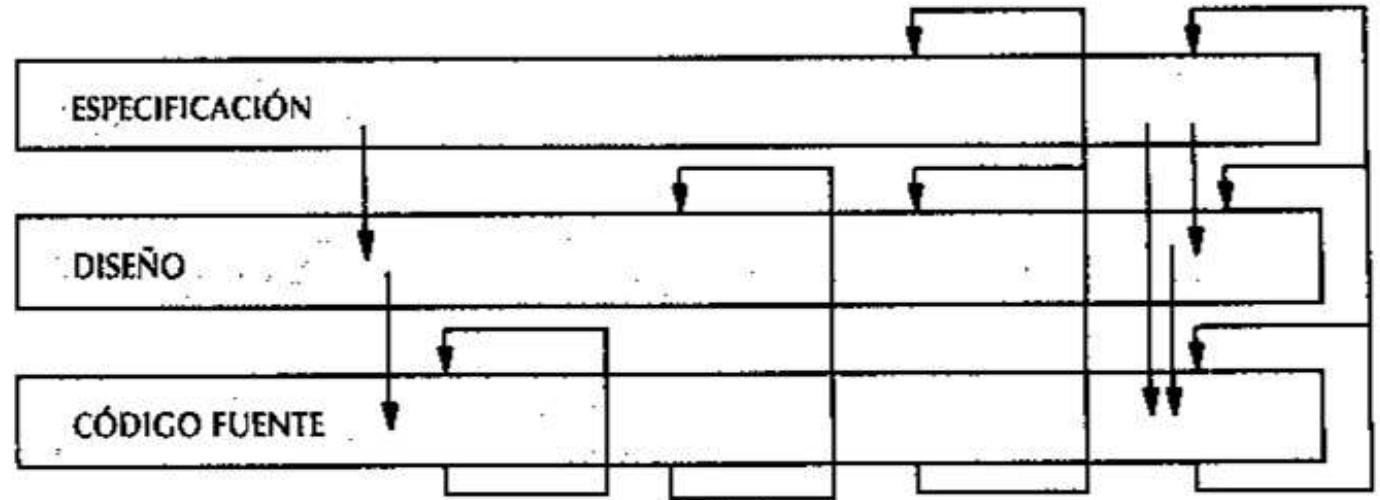
Rejuvenecimiento del Software

- » Es un desafío del mantenimiento, intentando aumentar la calidad global de un sistema existente
- » Contempla retrospectivamente los subproductos de un sistema para intentar derivar la información adicional o reformarlo de un modo comprensible
- » Tipos de Rejuvenecimiento
 - Re-documentación
 - Re-estructuración
 - Ingeniería Inversa
 - Re-ingeniería

25

Rejuvenecimiento del Software

26



Ingeniería progresiva
• avanza a través del proceso

Reestructuración
• desde el código
• representa internamente
• simplifica iterativamente la estructura y elimina código muerto
• regenera el código

Reestructuración de documentos
• desde el código
• informa el análisis estático sobre estructura, complejidad, volumen, datos, etc.
• no está basado en métodos de software

Ingeniería inversa
• desde el código
• produce especificación y diseño basados en métodos aceptados del software
• gestiona la representación

Reingeniería
• desde el código
• hace ingeniería reversa del código
• hace ingeniería progresiva: completa y modifica la representación, regenera el código

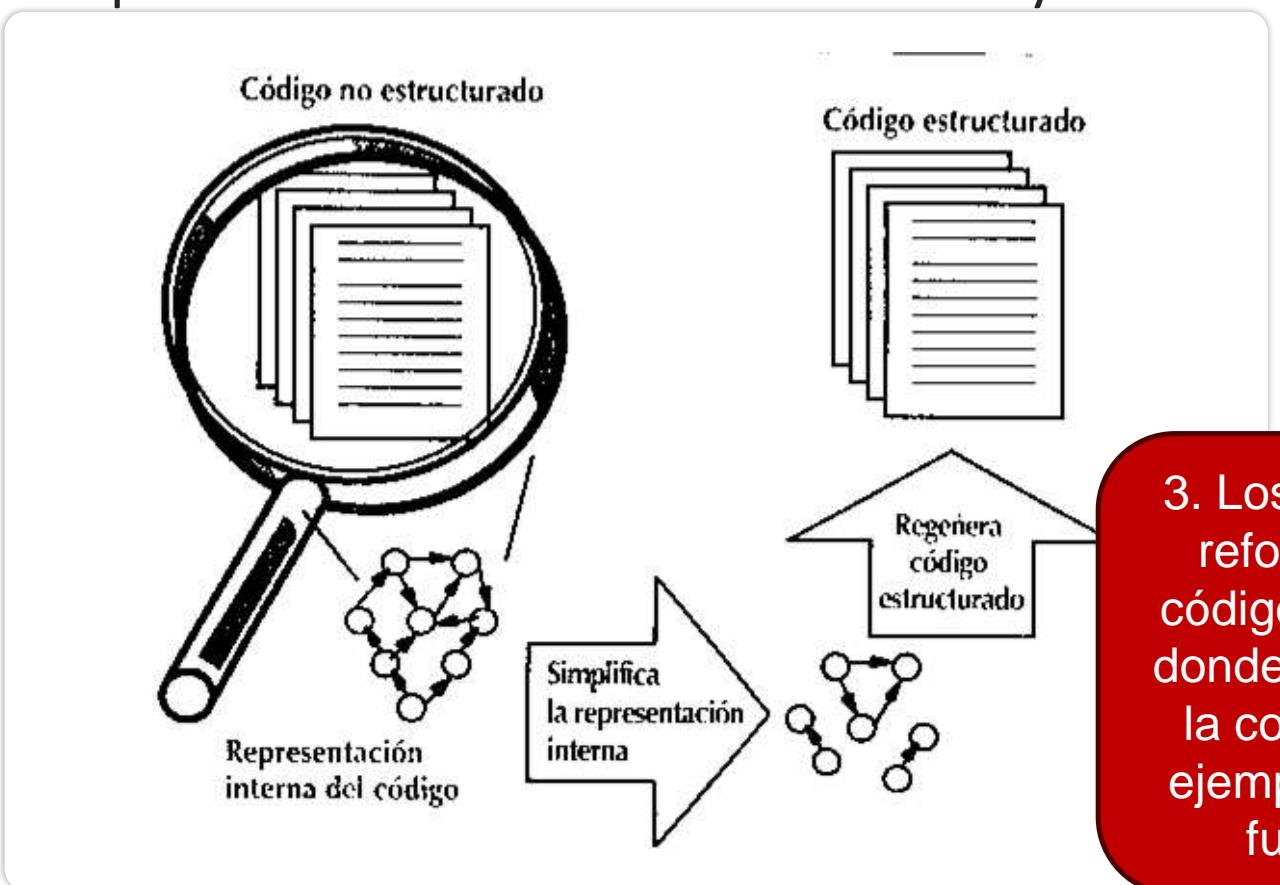
Rejuvenecimiento del Software

» Re-estructuración

Se reestructura el software para hacerlo más fácil de entender y de cambiar

1. Interpretamos el código fuente y lo representamos internamente como una red semántica o grafo.

2. Se utilizan reglas de transformación para simplificar la representación interna.



3. Los resultados se reformulan como código estructurado, donde se espera que la complejidad por ejemplo ciclomática fue reducida

27

Rejuvenecimiento del Software

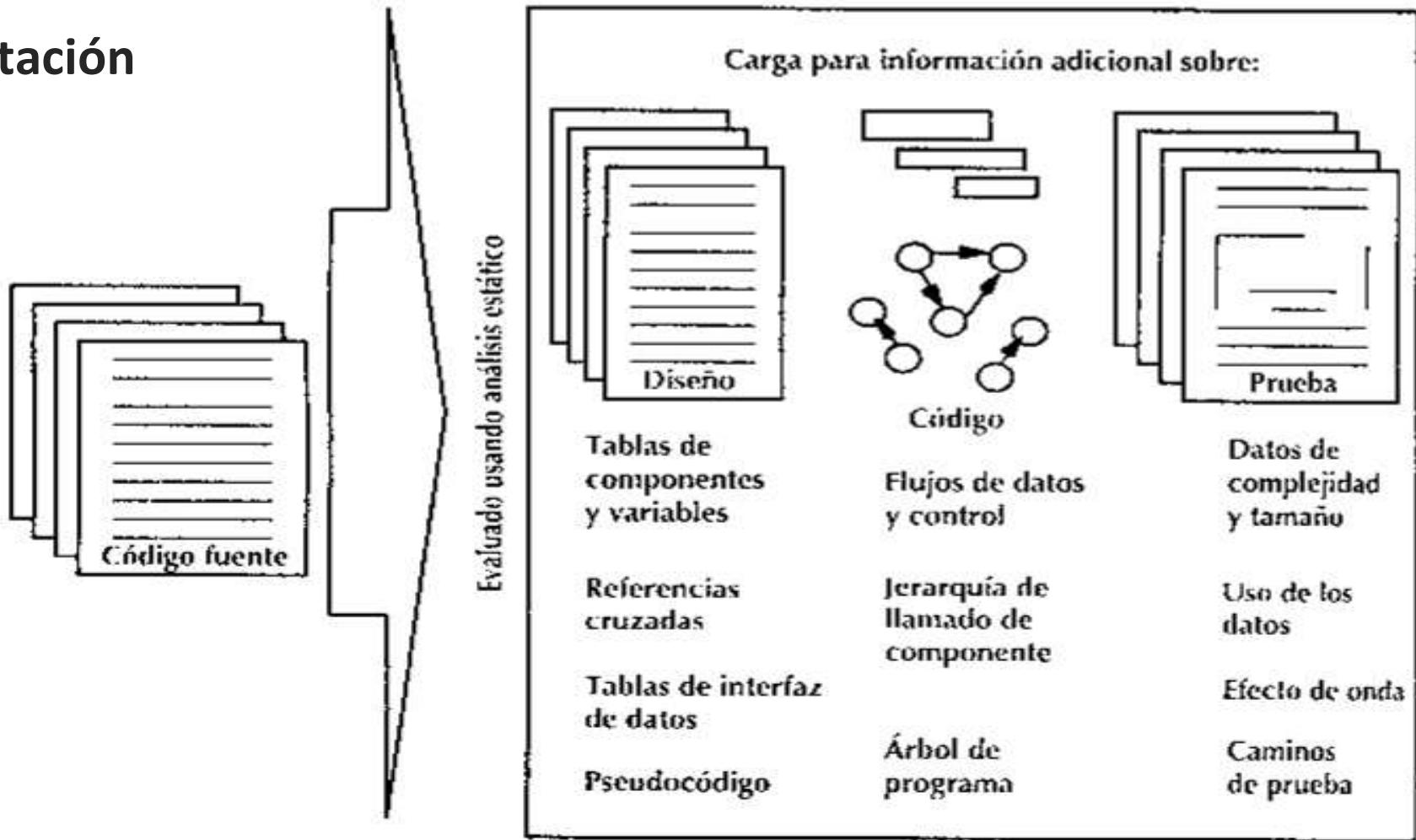
Re-documentación: Representa un análisis estático del código para producir la documentación del sistema.

- La información producida por un análisis de código estático puede ser gráfico o textual.
- Podemos obtener: relaciones de llamada de componentes, jerarquías de clases, tablas de interfaz de datos, información del diccionario de datos, tablas o diagramas de flujo de datos, tablas o diagramas de flujo de control, pseudocódigo, caminos de prueba, referencias cruzadas de componentes y variables.

28

Rejuvenecimiento del Software

» Re-documentación



29

Rejuvenecimiento del Software

»Ingeniería Inversa

La ingeniería inversa es el proceso de analizar un producto o sistema para comprender su diseño, funcionamiento interno y funcionalidad.

Parte del código fuente y recupera el diseño y en ocasiones la especificación, para aquellos sistemas en los que no hay documentación.

Se puede utilizar Ghidra (<https://ghidra-sre.org/>). Se utiliza para desensamblar, descompilar y analizar código binario. Incluye un descompilador que puede convertir el código ensamblador en un lenguaje de nivel superior, como C o Java, lo que puede facilitar la comprensión de la funcionalidad de un archivo binario

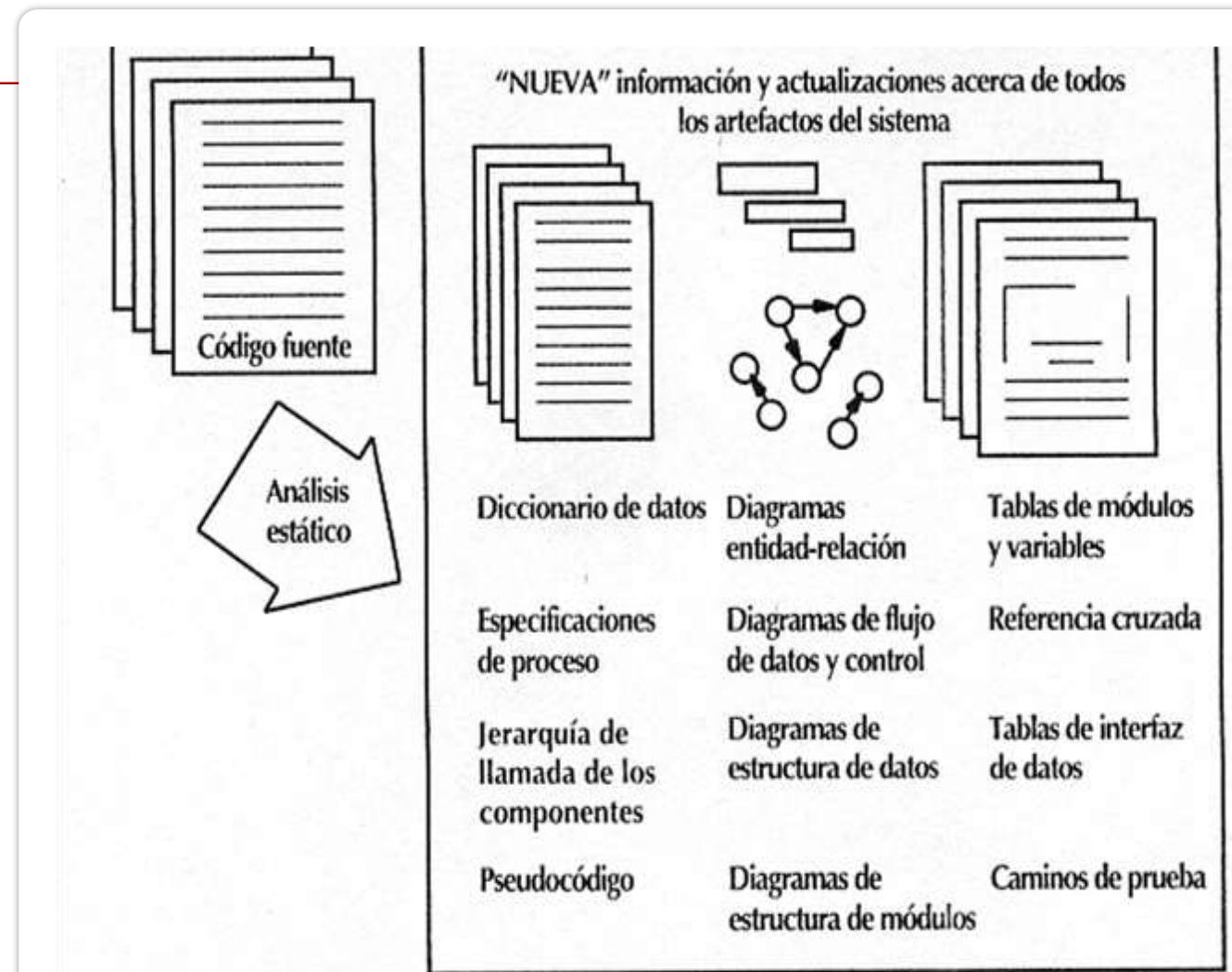
30

Rejuvenecimiento del Software

»Ingeniería Inversa

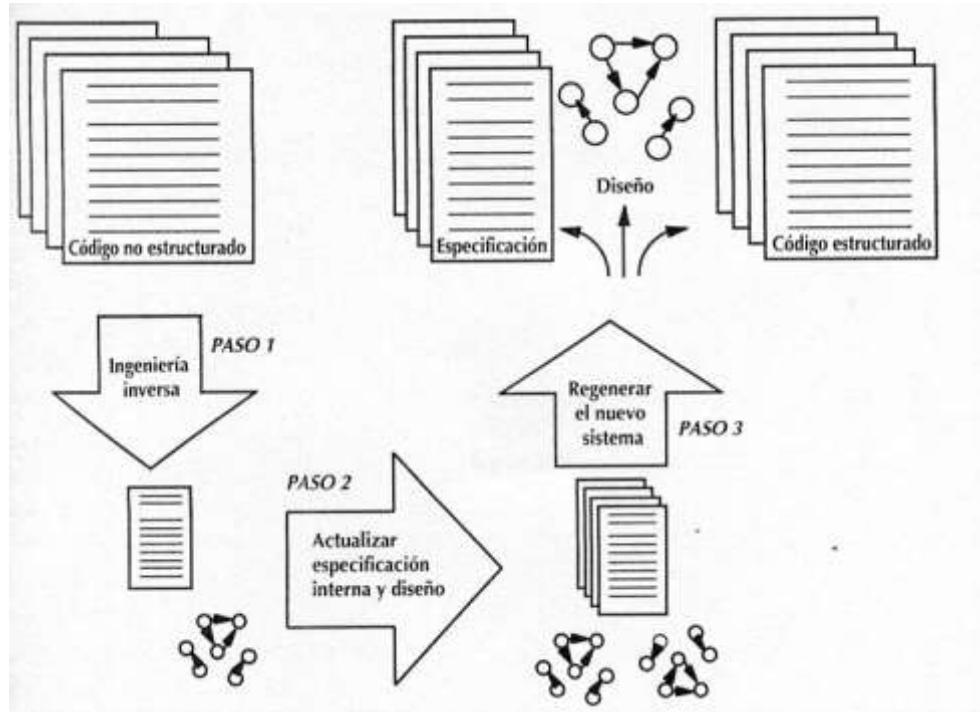
1. Se envía el código fuente, conectado a una herramienta de ingeniería inversa, que interpreta la estructura y la información de nomenclatura.

2. Se obtiene múltiples formatos hasta llegar a la especificación del sistema



Fuente.

Rejuvenecimiento del Software

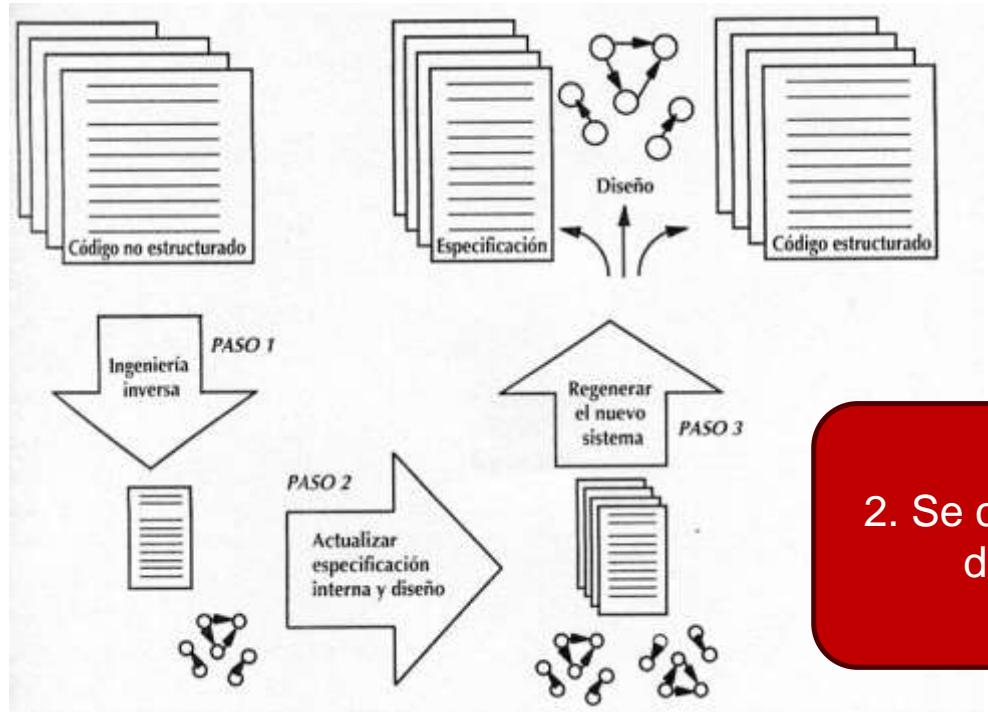


»Re-ingenería

Extensión de la ingeniería Inversa
Produce un nuevo código fuente
correctamente estructurado, mejorando
la calidad sin cambiar la funcionalidad
del sistema

32

Rejuvenecimiento del Software



1. El sistema tiene ingeniería inversa y se representa internamente para humanos y modificaciones de código basadas en métodos actuales para especificar y diseñar software.

2. Se corrige o completa el modelo del sistema de software.

3. El nuevo sistema se genera a partir de esta nueva especificación o diseño.

33

Futuro del rejuvenecimiento del software

- » Las herramientas comerciales de ingeniería inversa recuperan parcialmente un sistema de software pueden identificar, presentar y analizar información del código fuente, pero no reconstruyen, capturan ni expresan abstracciones de diseño que no son representado explícitamente en el código fuente.
- » La información del código fuente no contiene mucha información sobre el contexto original.

34

Futuro del rejuvenecimiento del software

- » En general para una buena recuperación se requiere información del código, documentación de diseño existente, experiencia personal y conocimiento general sobre el dominio del problema. Conocimiento lingüístico informal sobre el dominio del problema y se necesitan modismos de aplicación antes de que un diseño completo pueda ser entendido, reconstruido y estructurado.

- » El rejuvenecimiento del software avanzará cuando la tecnología y los métodos puedan capturar reglas, políticas, decisiones de diseño, terminología, convenciones de nomenclatura y otros elementos de información informal.

35



Ingeniería de Software II

Auditoría Informática

Auditoría Informática - Concepto

- » Permite definir estrategias para prevenir delitos, problemas legales, etc..
- » Es una actividad preventiva, el auditor sugiere.
- » Los procedimientos de auditoría en informática varían de acuerdo con la filosofía y técnica de cada organización y departamento de auditoría en particular.
- » La auditoría en informática debe evaluar todo: informática, organización del centro de cómputo, computadoras, comunicación y programas.



Auditoría Informática - Objetivos

-
- » Salvaguardar los activos.
 - » Integridad de datos.
 - » Efectividad de sistemas.
 - » Eficiencia de los sistemas.
 - » Seguridad y confidencialidad.

3



Auditoría Informática - Concepto

» Auditoría

Es un examen crítico que se realiza con el objeto de evaluar la eficiencia y la eficacia de una sección o de un organismo y determinar cursos alternativos de acción para mejorar la organización y lograr los objetivos propuestos.

4

No es una actividad meramente mecánica

Puede ser interna, externa o una combinación de ambas.



Auditoría Informática - Definiciones

“Es una función que ha sido desarrollada para asegurar la salvaguarda de los activos de los sistemas de computadoras, mantener la integridad de los datos y lograr los objetivos de la organización en forma eficaz y eficiente”. **Ron Weber.**

5

“Es la verificación de los controles en las siguientes tres áreas de la organización (informática): Aplicaciones, Desarrollo de sistemas, Instalación del centro de cómputos”. **William Mair.**

Auditoría Informática - Concepto

» Por lo tanto, es la revisión y evaluación de:

- los controles, sistemas y procedimientos de la informática;
- los equipos de cómputo;
- la organización que participa en el procesamiento de la información.

6



Influencia de la auditoría en informática

Factores que pueden influir en la organización a través del control y la auditoría en informática:

- » Controlar el uso de la computadora.
- » Pérdida de capacidades de procesamiento de datos.
- » Necesidad de mantener la privacidad individual.
- » Posibilidad de pérdida de información o mal uso de la misma.
- » Toma de decisiones incorrectas.
- » Necesidad de mantener la privacidad de la organización.

- » ...

Auditoría Informática – Campo de acción

-
1. Evaluación administrativa del área de informática.
 2. Evaluación de los sistemas y procedimientos, y de la eficiencia que se tiene en el uso de la información.
 3. Evaluación del proceso de datos, de los sistemas y de los equipos de cómputo (software, hardware, redes, bases de datos, comunicaciones).
 4. Seguridad y confidencialidad.
 5. Aspectos legales de los sistemas y de la información.

¿Qué hace un auditor informático?

- » Es responsable de evaluar los sistemas informáticos y los procesos relacionados con la tecnología de una empresa, lo que incluye la infraestructura tecnológica.
- » El objetivo de dicha evaluación es asegurarse de que los sistemas y procesos son los que la empresa verdaderamente necesita, al mismo tiempo que ofrece soluciones viables para cualquier problema que se haya detectado durante la evaluación.

Funciones de un Auditor Informático

- » Identificar el potencial de optimización de procesos informáticos a partir de un análisis de debilidades y errores.
- » Seguir los estándares de auditoría establecidos por cada empresa y la normativa vigente.
- » Analizar la situación actual de los sistemas informáticos y procesos de la empresa.
- » Elaborar informes detallados que incluyan los resultados de cada auditoría realizada.

10

Funciones de un Auditor Informático

- » Controlar y verificar el funcionamiento de los sistemas informáticos internos y de cualquier otro servicio que dependa de la infraestructura tecnológica de la empresa.
- » Detectar riesgos del entorno informático para prevenir posibles ciberataques y desvíos de datos.
- » Diseñar soluciones para los problemas o errores detectados en la auditoría.
- » Proponer estrategias para mejorar los sistemas informáticos.

11

¿Dónde trabaja un auditor informático?

- » Este profesional encuentra oportunidades laborales en grandes empresas que cuentan con su propio departamento de informática y ciberseguridad.
- » Por otro lado, puede ser contratado de forma externa por las compañías que requieran de las funciones de un auditor informático, o bien, puede ejercer de forma independiente.

12

Etapas genéricas de la Auditoria Informática

2. Se recopilan datos a partir de entrevistas, revisión de documentos y del uso de herramientas tecnológicas.



1. En esta etapa se identifican los objetivos específicos de la auditoría y se define su alcance, considerando los recursos disponibles y las áreas clave a examinar.

Se elabora un plan detallado donde se incluyen los métodos y herramientas que se van a utilizar, así como un cronograma de las actividades.

3. Se evalúan los controles internos y procesos de los sistemas para determinar su eficacia y eficiencia, así como su alineación con los objetivos de la empresa. Se analizan e identifican posibles deficiencias o riesgos en los sistemas.

4. Una vez recopilados y analizados los datos, se redacta un informe detallado que incluye las conclusiones de la auditoría, las áreas de mejora, y las recomendaciones. Este documento es crucial para la toma de decisiones a futuro y se presenta a la dirección y otros interesados de la organización.

Finalización de la auditoría

- » Al finalizar la auditoría se deben implementar las mejoras recomendadas para fortalecer los sistemas de información de la empresa.
- » Además, es esencial realizar un seguimiento continuo para asegurar que las mejoras sean efectivas, así como para adaptarse a cualquier cambio en el entorno tecnológico y empresarial.
- » Aunque la frecuencia con la que debe hacerse una auditoría informática depende de varios factores, incluyendo el tamaño de la organización y la naturaleza de sus actividades, se recomienda hacerla anualmente.

14

Tipos de auditoría

INTERNA: Es la realizada con recursos materiales y personas que pertenecen a la empresa auditada. Los empleados que realizan esta tarea son remunerados económicaamente. La auditoría interna existe por expresa decisión de la Empresa, o sea, que puede optar por su disolución en cualquier momento



EXTERNA: Es realizada por personas afines a la empresa auditada; es siempre remunerada. Se presupone una mayor objetividad que en la auditoría Interna, debido al mayor distanciamiento entre auditores y auditados.

15

Principios aplicados al Auditor Informático

» PRINCIPIO DE BENEFICIO DE AUDITADO

En este principio el auditor debe conseguir la máxima eficacia y rentabilidad de los medios informáticos de la empresa auditada, no debe de ningún modo obtener beneficio propio.

» PRINCIPIO DE CALIDAD

En el auditor deberá prestar sus servicios conforme las posibilidades de la ciencia y medios a su alcance con absoluta libertad respecto a la utilización de dichos medios y en unas condiciones técnicas adecuadas para el idóneo cumplimiento de su labor.

» PRINCIPIO DE CONFIANZA

El auditor deberá facilitar e incrementar la confianza del auditor en base a una actuación de transparencia en su actividad profesional sin alardes científicos-técnicos.

16

Principios aplicados al Auditor Informático

» PRINCIPIO DE CAPACIDAD

El auditor debe estar plenamente capacitado para la realización de la auditoría encomendada, máxime teniendo en cuenta que, a los auditados en algunos casos les puede ser extremadamente difícil verificar sus recomendaciones y evaluar correctamente la precisión de las mismas.

» PRINCIPIO DE COMPORTAMIENTO PROFESIONAL

El auditor, tanto en sus relaciones con el auditado como con terceras personas, deberá, en todo momento, actuar conforme a las normas, implícitas o explícitas, de dignidad de la profesión y de corrección en el trato personal.

» PRINCIPIO DE CRITERIO PROPIO

El auditor durante la ejecución deberá actuar con criterio propio y no permitir que esté subordinado al de otros profesionales, aun de reconocido prestigio, que no coincidan con el mismo.

Principios aplicados al Auditor Informático

» PRINCIPIO DE CONCENTRACIÓN EN EL TRABAJO

El auditor deberá evitar que un exceso de trabajo supere sus posibilidades de concentración y precisión en cada una de las tareas a él encomendadas, y a que la estructuración y dispersión de trabajos suele, a menudo, si no está debidamente controlada, provocar la conclusión de los mismos sin las debidas garantías de seguridad.

18

» PRINCIPIO DE DISCRECIÓN

El auditor deberá en todo momento mantener una cierta discreción en la divulgación de datos, aparentemente inocuos, que se le hayan puesto de manifiesto durante la ejecución de la auditoría.

» PRINCIPIO DE ECONOMÍA

El auditor deberá proteger, en la medida de sus conocimientos, los derechos económicos del auditado evitando generar gastos innecesarios en el ejercicio de su actividad.

Principios aplicados al Auditor Informático

» PRINCIPIO DE FORMACIÓN CONTINUADA

Este principio impone a los auditores el deber y la responsabilidad de mantener una permanente actualización de sus conocimientos y métodos a fin de adecuarlos a las necesidades de la demanda y a las exigencias de la competencia de la oferta.

» PRINCIPIO DE FORTALECIMIENTO Y RESPETO DE LA PROFESIÓN

La defensa de los auditados pasa por el fortalecimiento de la profesión de los auditores informáticos, lo que exige un respeto por el ejercicio, globalmente considerado, de la actividad desarrollada por los mismos y un comportamiento acorde con los requisitos exigibles para el idóneo cumplimiento de la finalidad de las auditorias.

» PRINCIPIO DE INDEPENDENCIA

Está relacionado con el principio de criterio propio, obliga al auditor, tanto si actúa como profesional externo o con dependencia laboral respecto a la empresa en la que deba realizar la auditoria informática, a exigir una total autonomía e independencia en su trabajo.

19

Principios aplicados al Auditor Informático

» PRINCIPIO DE INFORMACIÓN SUFFICIENTE

Este principio obliga al auditor a aportar, en forma pormenorizada, clara, precisa e inteligible para el auditado, información de los puntos y conclusiones relacionados con la auditoría.

» PRINCIPIO DE INTEGRIDAD MORAL

Este principio, inherentemente ligado a la dignidad de la persona, obliga al auditor a ser honesto, leal y diligente en el desempeño de su misión, a ajustarse a las normas morales de justicia y prioridad.

» PRINCIPIO DE LEGALIDAD

La primacía de esta obligación exige del auditor un comportamiento activo de oposición a todo intento, por parte del auditado o de terceras personas, tendente a infringir cualquier precepto integrado en el derecho positivo.

20

Principios aplicados al Auditor Informático

» PRINCIPIO DE LIBRE COMPETENCIA

La actual economía de mercado exige que el ejercicio de la profesión se realice en el marco de la libre competencia siendo rechazables, por tanto, las prácticas colusorias tendentes a impedir o limitar la legítima competencia de otros profesionales.

» PRINCIPIO DE NO DISCRIMINACIÓN

El auditor en su actuación previa, durante y posterior a la auditoría deberá evitar cualquier tipo de condicionantes personalizados y actuar en todos los casos con similar diligencia.

» PRINCIPIO DE NO INJERENCIA

El auditor, deberá evitar injerencias en los trabajos de otros profesionales, respetar su labor y eludir hacer comentarios que pudieran interpretarse como despectivos de la misma, deberá igualmente evitar aprovechar los datos.

21

Principios aplicados al Auditor Informático

» PRINCIPIO DE PRECISIÓN

Este principio exige del auditor la no conclusión de su trabajo hasta estar convencido, en la medida de lo posible, de la viabilidad de sus propuestas.

» PRINCIPIO DE PUBLICIDAD ADECUADA

La oferta y promoción de los servicios de auditoría deberán en todo momento ajustarse a las características, condiciones y finalidad perseguidas.

» PRINCIPIO DE RESPONSABILIDAD

El auditor deberá, como elemento intrínseco de todo comportamiento profesional, responsabilizarse de lo que haga, diga o aconseje.

22

Principios aplicados al Auditor Informático

» PRINCIPIO DE SECRETO PROFESIONAL

La confidencia y confianza entre el auditor y el auditado e imponen al primero la obligación de guardar en secreto los hechos e informaciones que conozca en el ejercicio de su actividad profesional.

» PRINCIPIO DE SERVICIO PÚBLICO

La aplicación de este principio debe incitar al auditor a hacer lo que este en su mano y sin perjuicio de los intereses de su cliente, para evitar daños sociales.

» PRINCIPIO DE VERACIDAD

El Auditor en sus comunicaciones con el auditado deberá tener siempre presente la obligación de asegurar la veracidad de sus manifestaciones con los límites impuestos por los deberes de respeto, corrección, y secreto profesional.

23

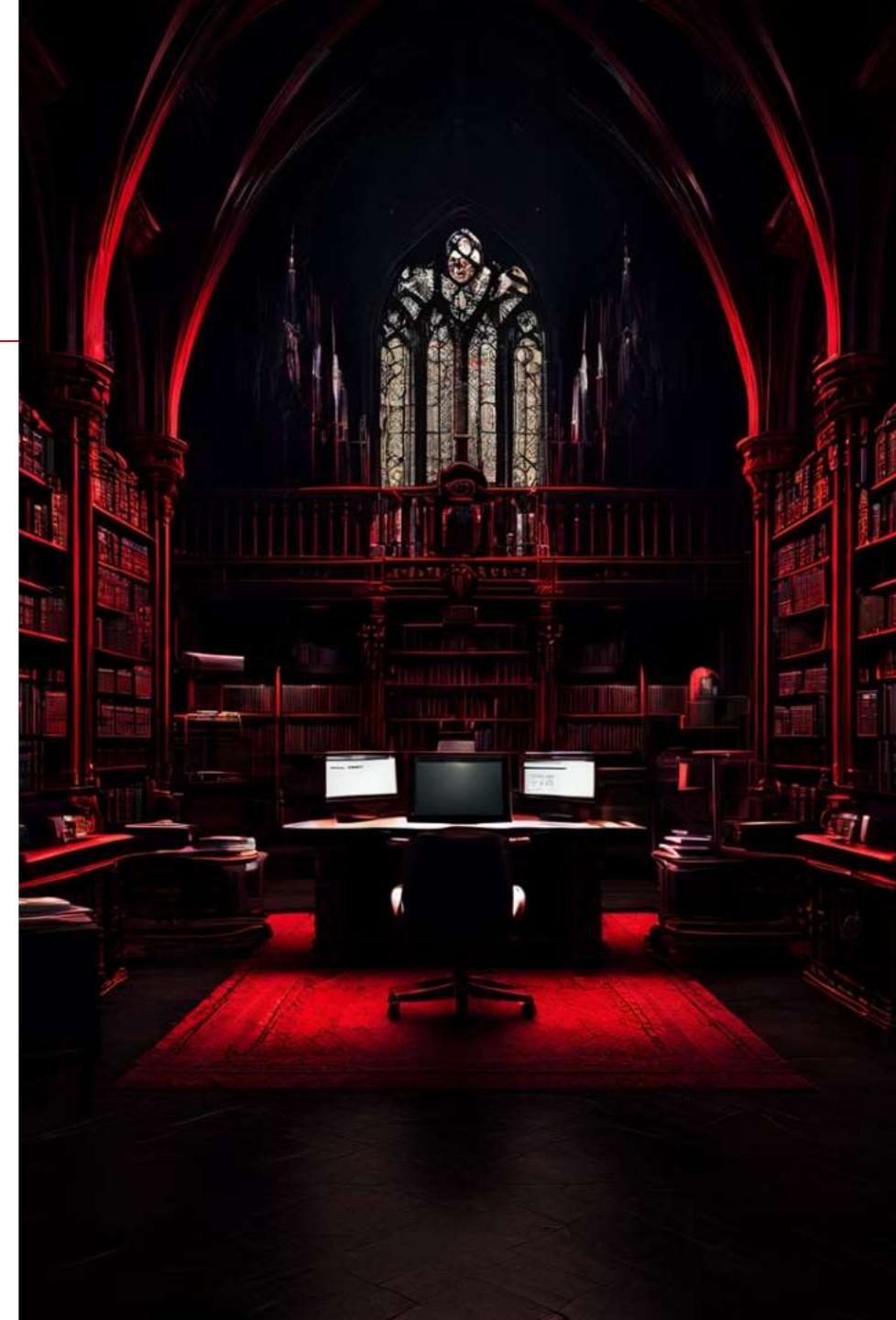
Herramientas para realizar auditorías informáticas

Las auditorías informáticas requieren de un conjunto de herramientas especializadas para examinar y evaluar los sistemas, procesos y controles de una organización.

Abarcan desde escáneres de vulnerabilidades hasta analizadores de tráfico de red, pasando por herramientas de monitoreo y diagnóstico.

Algunas de las herramientas más comunes utilizadas en auditorías informáticas son: Nessus, Wireshark, Metasploit, Burp Suite, Sqlmap, Maltego y OSSEC, entre otras.

Permiten identificar puntos débiles, detectar intrusiones, analizar el flujo de información y evaluar el cumplimiento normativo.



Metodología Octave para auditoría informática

1

Identificar activos

Enumerar los activos de información clave

2

Analizar amenazas

Evaluuar las posibles amenazas a los activos

3

Determinar vulnerabilidades

Identificar las debilidades que pueden ser explotadas

La metodología Octave (Operationally Critical Threat, Asset, and Vulnerability Evaluation) es un enfoque integral para la evaluación de riesgos de seguridad de la información. Consiste en una serie de pasos estructurados que ayudan a identificar los activos críticos, analizar las amenazas y evaluar las vulnerabilidades de una organización.

Metodología Margerit para auditoría informática



La metodología Margerit es un marco completo para realizar auditorías informáticas de manera sistemática y exhaustiva. Comienza con una etapa de planificación donde se definen los objetivos y el alcance del proyecto. Luego sigue un análisis detallado de los riesgos y los controles existentes en la organización. La evaluación permite valorar la eficacia de estos controles y finalmente se generan recomendaciones específicas para mejorar la seguridad informática.

Evaluación de controles en auditoría informática



Revisión de controles técnicos

Durante la auditoría, el equipo evaluará los controles técnicos implementados, como firewalls, sistemas de detección de intrusos y configuración de servidores, para asegurar su eficacia y cumplimiento con las políticas de seguridad.



Inspección de controles físicos

Se realizarán inspecciones in situ para verificar la efectividad de los controles físicos, como sistemas de acceso, monitoreo por cámaras y resguardo de áreas críticas, que protejan la infraestructura tecnológica.



Evaluación de controles administrativos

Además, se revisarán los controles administrativos, como políticas de seguridad de la información, procedimientos de gestión de incidentes y concienciación de los usuarios, para evaluar su implementación y eficacia.

Reporte de hallazgos en auditoría informática

Un ejemplo

El reporte de hallazgos es un componente clave de la auditoría informática. En esta etapa, se documentan de manera clara y concisa los problemas, debilidades o riesgos identificados durante el proceso de evaluación. El informe debe proporcionar una visión general de los hallazgos más importantes, clasificados por nivel de criticidad y acompañados de recomendaciones específicas.

Hallazgo	Descripción	Nivel de Riesgo
Falta de controles de acceso adecuados	Se identificó que no hay controles suficientes para restringir el acceso a sistemas críticos. Varios usuarios tienen privilegios excesivos.	Alto
Vulnerabilidades en el sistema de backups	El proceso de respaldo de información presenta fallas y no se realiza de manera sistemática. Existe riesgo de pérdida de datos críticos.	Medio
Falta de monitoreo de actividad en la red	No se implementan herramientas ni procesos para el monitoreo y análisis del tráfico de red. Esto dificulta la detección temprana de incidentes de seguridad.	Alto

Recomendaciones y plan de acción en auditoría informática



Recomendaciones

Basado en los hallazgos de la auditoría, se brindan recomendaciones específicas y detalladas para mejorar la seguridad, eficiencia y cumplimiento de los sistemas y procesos informáticos.



Plan de Acción

Se establece un plan de acción claro y estructurado, con plazos, responsables y recursos definidos, para implementar las recomendaciones de manera efectiva y oportuna.



Seguimiento

Se diseña un sistema de seguimiento y monitoreo para evaluar el progreso y el impacto de las acciones implementadas, con el fin de asegurar el cumplimiento de los objetivos de la auditoría.

Beneficios de la Auditoría Informática

La auditoría informática ofrece múltiples beneficios a las organizaciones, como la identificación de riesgos, la mejora de controles internos y la optimización de recursos tecnológicos. Además, ayuda a garantizar el cumplimiento normativo y la seguridad de la información.

Al implementar recomendaciones de auditoría, las empresas pueden fortalecer su posición competitiva, aumentar la confianza de clientes y partes interesadas, y prepararse mejor para enfrentar amenazas ciberneticas.





Diferencias entre Consultor, Auditor y Perito Informático

CONSULTOR INFORMÁTICO

Función: El consultor informático asesora a las organizaciones en cuestiones relacionadas con la tecnología y los sistemas de información.

Enfoque: Su enfoque es más amplio y estratégico. Busca optimizar los procesos, mejorar la eficiencia y alinear la tecnología con los objetivos empresariales.

Actividades: Realiza análisis de necesidades, propone soluciones tecnológicas, diseña estrategias de implementación y brinda recomendaciones.



Diferencias entre Consultor, Auditor y Perito Informático

AUDITOR INFORMATICO

Función: El auditor informático evalúa y verifica los sistemas de información y los controles internos de una organización.

Enfoque: Su enfoque es más específico y técnico. Busca identificar riesgos, vulnerabilidades y deficiencias en los sistemas.

Actividades: Realiza revisiones, pruebas y análisis exhaustivos de los procesos, la seguridad, la integridad de los datos y el cumplimiento normativo.



Diferencias entre Consultor, Auditor y Perito Informático

PERITO INFORMATICO

Función: El perito informático actúa como experto en casos legales o judiciales relacionados con la informática.

Enfoque: Su enfoque es legal y forense. Ayuda a recopilar pruebas digitales, analiza incidentes de seguridad y presenta informes periciales.

Actividades: Participa en investigaciones, realiza análisis forenses de dispositivos y colabora con abogados y tribunales.



Bibliografía de consulta

Echenique García, J. A. (2001). Auditoría en informática.
Compañía Editorial Continental. 2da Edición

Piattini, M., & del Peso, E. (2008). Auditoria informática: Un enfoque práctico. 2ª Edición ampliada y revisada.

También pueden acceder a 2 informes de auditoría que se subieron en el aula virtual