



INGENIERÍA DEL SOFTWARE I

Tema 3

Procesos de Ingeniería del Software

Universidad Cantabria – Facultad de Ciencias

Juan Hernández, Francisco Ruiz



Objetivos y Bibliografía

- Comprender las relaciones entre los conceptos de **proceso software**, **ciclo de vida** del software y **metodología**.
- Conocer las características de los procesos software y cuales pueden ser dichos procesos.
- Conocer los principales ciclos de vida del software.
- Comprender la finalidad y características de una metodología software y los principales tipos.
- **Básica**
 - Caps. 2 y 3 del libro de Piattini (2007).
 - ISO/IEC 12207 Information Technology / Software Life Cycle Processes. 1995. [Disponible versión española (norma UNE 71044)]
- **Complementaria**
 - Caps. 2 y 3 del libro de Pressman (2005).
 - Caps. 2 y 4 del libro de Sommerville (2005).
 - Cap. 2 del libro de Pfleeger (2002).



Contenido

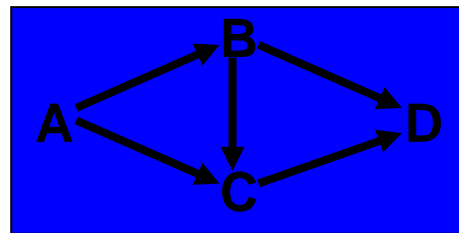
- Procesos Software
 - Naturaleza y Elementos
 - Relación con otros tipos de Procesos
- Concepto de Ciclo de Vida.
- Estándar ISO 12207.
- Ciclos de Vida Tradicionales
 - En Cascada
 - Incremental
 - En Espiral
 - Prototipado
 - Reutilización
 - Síntesis Automática
 - Comparativa
- Ciclos de Vida para Sistemas OO
 - Modelo de Agrupamiento
 - Modelo Fuente
 - Modelo Remolino
 - Modelo Pinball
- Metodologías de Desarrollo de Software
 - Definición y Objetivos
 - Elementos
 - Características Deseables
 - Conceptos Relacionados
 - Impacto en el Entorno
- Tipos de Metodologías
 - Estructuradas
 - Orientadas a Procesos
 - Orientadas a Datos
 - Orientadas a Objetos (OO)
 - Ágiles.
- APÉNDICE A: Evolución Metodologías de Desarrollo de Software.
- APÉNDICE B: Ejemplos de Metodologías.



Un **Proceso Software** (PS) es

“Un conjunto coherente de políticas, estructuras organizacionales, tecnologías, procedimientos y artefactos que son necesarios para concebir, desarrollar, instalar y mantener un producto software.”

(Fugetta, 2000)



Métodos y Procedimientos que definen la relaciones entre las Tareas.

Personal



**PROCESO
SW**

**Herramientas y
Metodologías.**

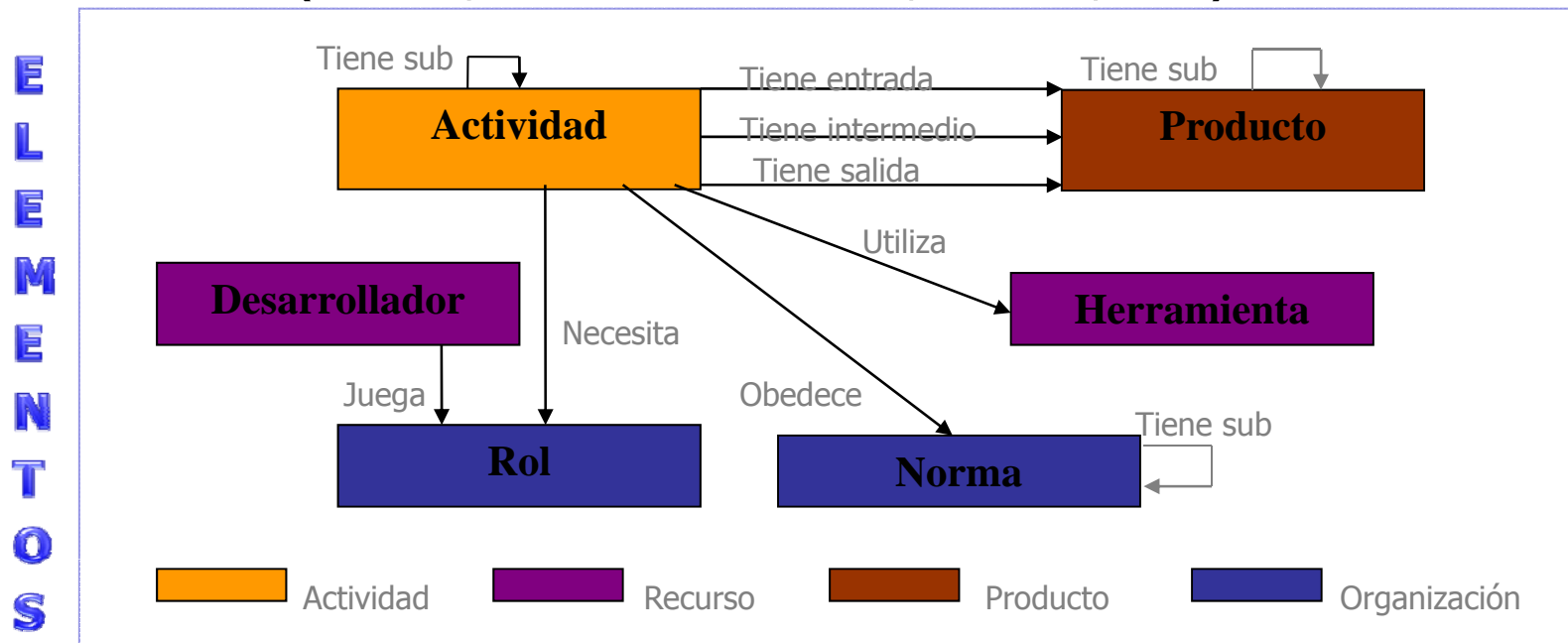




- Son complejos.
- No son procesos de producción.
 - Dirigidos por excepciones,
 - Muy determinados por circunstancias impredecibles,
 - Cada uno con sus peculiaridades.
- No son procesos de ingeniería "*pura*".
 - Desconocemos las abstracciones adecuadas,
 - Dependen demasiado de demasiada gente,
 - Diseño y producción no están claramente separados,
 - Presupuestos, calendarios, calidad no pueden ser planificados de forma fiable.
- No son (*completamente*) procesos creativos:
 - Algunas partes pueden ser descritas en detalle,
 - Algunos procedimientos han sido impuestos.



- Basados en descubrimientos dependientes de la comunicación, coordinación y cooperación en los marcos de trabajo definidos.
 - Los entregables generan nuevos requerimientos,
 - Los costes del cambio del software no suelen reconocerse,
 - El éxito depende de la implicación del usuario y de la coordinación de muchos roles (ventas, desarrollo técnico, cliente, etc.).





	Industriales	Información	Negocio
Foco	COSAS	DATOS	RELACIONES
Propósito	Transformar y ensamblar materiales y componentes en otros componentes y productos finales, usando recursos	Procesar y transmitir datos estructurados y no estructurados, y conocimiento	Alcanzar las condiciones que satisfacen las necesidades de los participantes, clientes o usuarios
Características	Tradiciones de la ingeniería industrial	Tradiciones de la ingeniería informática	Basados en estructuras de comunicación y coordinación humanas encontradas en todos los lenguajes y culturas
Acciones	Ensamblar, Transformar, Transportar, Almacenar, Inspeccionar	Enviar, Invocar, Grabar, Recuperar, Consultar, Clasificar.	Solicitar, Prometer, Ofrecer, Rechazar, Proponer, Cancelar, Medir



Concepto de Ciclo de Vida

"Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software".

IEEE 1074

"Un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso".

ISO 12207

- Responde a la pregunta ¿Qué procesos se pueden realizar? (no cómo)

Ciclo de Vida \neq Ciclo de Desarrollo



Estándar ISO 12207

“Establece un marco de referencia común para los procesos del ciclo de vida del software, con una terminología bien definida, que puede ser referenciada por la industria del software”.
ISO/IEC 12207: Information Technology/Software Life Cycle Processes. 1995

- Define los **procesos**, **actividades** (que forman cada proceso) y **tareas** (que constituyen cada actividad) presentes en la adquisición, suministro, desarrollo, operación y mantenimiento del software.
- Según esta norma, un **proceso** es un conjunto de actividades interrelacionadas que transforman entradas en salidas. Un *proceso define quién, qué, cuándo, y cómo, para alcanzar un determinado objetivo.*
- Versión en Español. AENOR Norma UNE 71044: Tecnología de la información / Procesos del ciclo de vida del software. 1999.
- Actualización y ampliación integrando el ciclo de vida del software en el ciclo de vida de sistemas. ISO/IEC FDIS 12207: Systems and software engineering — Software life cycle processes. 2007.



Estándar ISO 12207



International Organization for Standardization

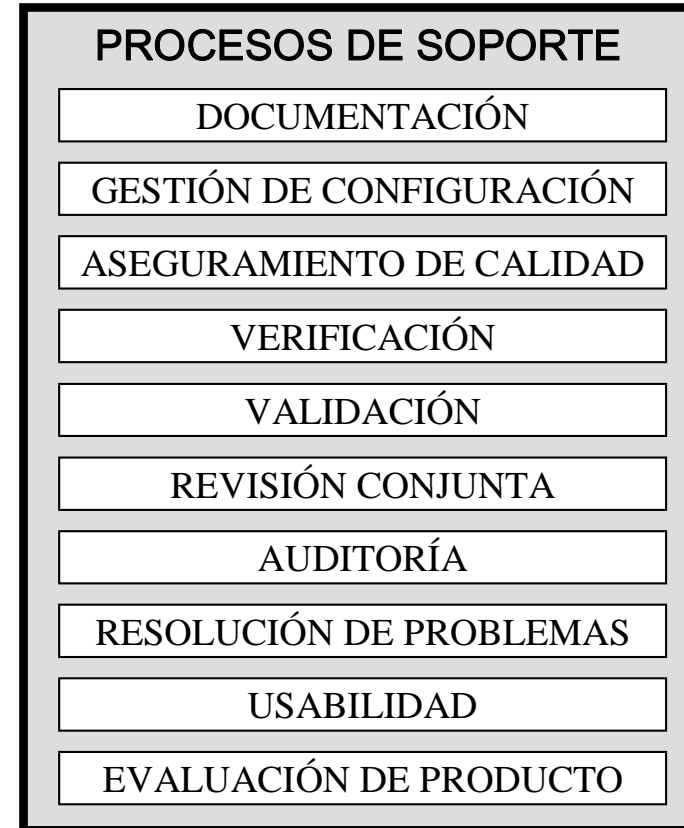


© AENOR 2004



Estándar ISO 12207

**C
P
R
O
C
E
S
O
S
D
E
A**





ISO 12207: Procesos Principales



- **Proceso de Adquisición.**

- Actividades y tareas que el **comprador**, cliente o usuario, realiza para adquirir un sistema o producto software.

- **Proceso de Suministro.**

- Actividades y tareas que efectúa el **suministrador**, para proporcionar un producto al **cliente**.

- **Proceso de Explotación.**

- Incluye la operación del producto software en su entorno final y el soporte operativo a los clientes.

- **Proceso de Mantenimiento.**

- Incluye la modificación de un sistema o producto software después de la entrega para:
 - Corregir los fallos (**correctivo**)
 - Mejorar el rendimiento u otros atributos (**de mejora**)
 - Adaptarlo a un entorno modificado (**adaptativo**).
- Esta modificación (o retirada) debe hacerse **preservando la integridad**.



ISO 12207: Procesos Principales

- **Proceso de Desarrollo.**

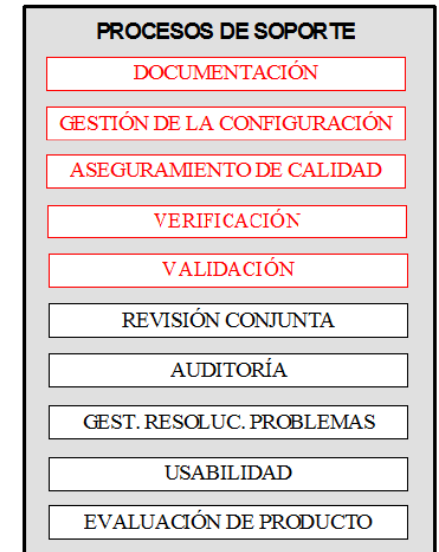
- Captura de Requisitos
- Análisis de Requisitos del Sistema
- Diseño Arquitectónico del Sistema
- Análisis de los Requisitos del Software
- Diseño de la Arquitectura del Software
- Diseño del Software
- Construcción del Software
- Integración del Software
- Prueba del Software
- Integración del Sistema
- Prueba del Sistema
- Instalación del Software





ISO 12207: Procesos de Soporte

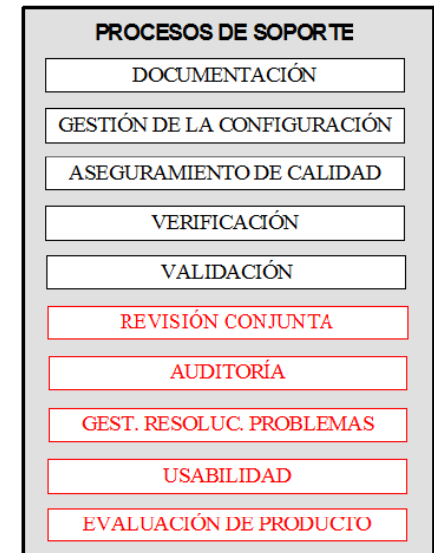
- **Proceso de Documentación:** Desarrollo y mantenimiento de la información software registrada por un proceso.
- **Proceso de Gestión de la Configuración:** Establecer y mantener de la integridad de todos los productos de trabajo de un proceso o proyecto y hacerlos disponibles para las partes involucradas.
- **Proceso de Aseguramiento de la Calidad:** Asegura que los productos de trabajo y los procesos cumplen las previsiones y planes predefinidos.
- **Proceso de Verificación:** Confirmación de que todos los productos de trabajo y/o servicios software de un proceso o proyecto reflejan de forma apropiada los requisitos especificados. ¿Estamos construyendo correctamente el producto?
- **Proceso de Validación:** Sirve para determinar si el sistema o software final cumple con los requisitos previstos para su uso. ¿Estamos construyendo el producto correcto?





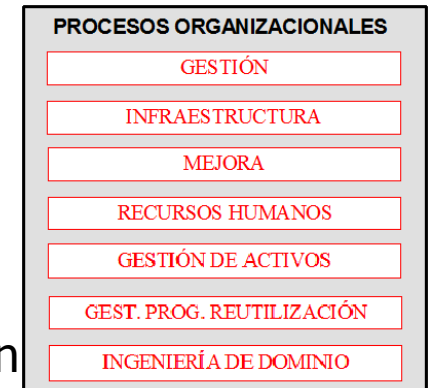
ISO 12207: Procesos de Soporte

- **Proceso de Revisión Conjunta:** Entendimiento común entre las diferentes partes involucradas sobre el progreso respecto de los objetivos y sobre lo que debe hacerse para ayudar a asegurar el desarrollo de un producto que satisfice a las partes involucradas.
- **Proceso de Auditoría:** Permite determinar, de forma independiente, la conformidad de los productos y procesos seleccionados con los requisitos, planes y acuerdos.
- **Proceso de Resolución de Problemas:** Asegurar que todos los problemas descubiertos se analizan y resuelven.
- **Proceso de Usabilidad:** Permitir la optimización del soporte y de la formación, la mejora de la productividad, calidad y condiciones de trabajo de las personas y la reducción de probabilidad de rechazo del sistema.
- **Proceso de Evaluación de Productos:** Aseguramiento mediante el examen y la medición sistemáticos, que un producto satisfice las necesidades implícitas y explícitas de los usuarios.





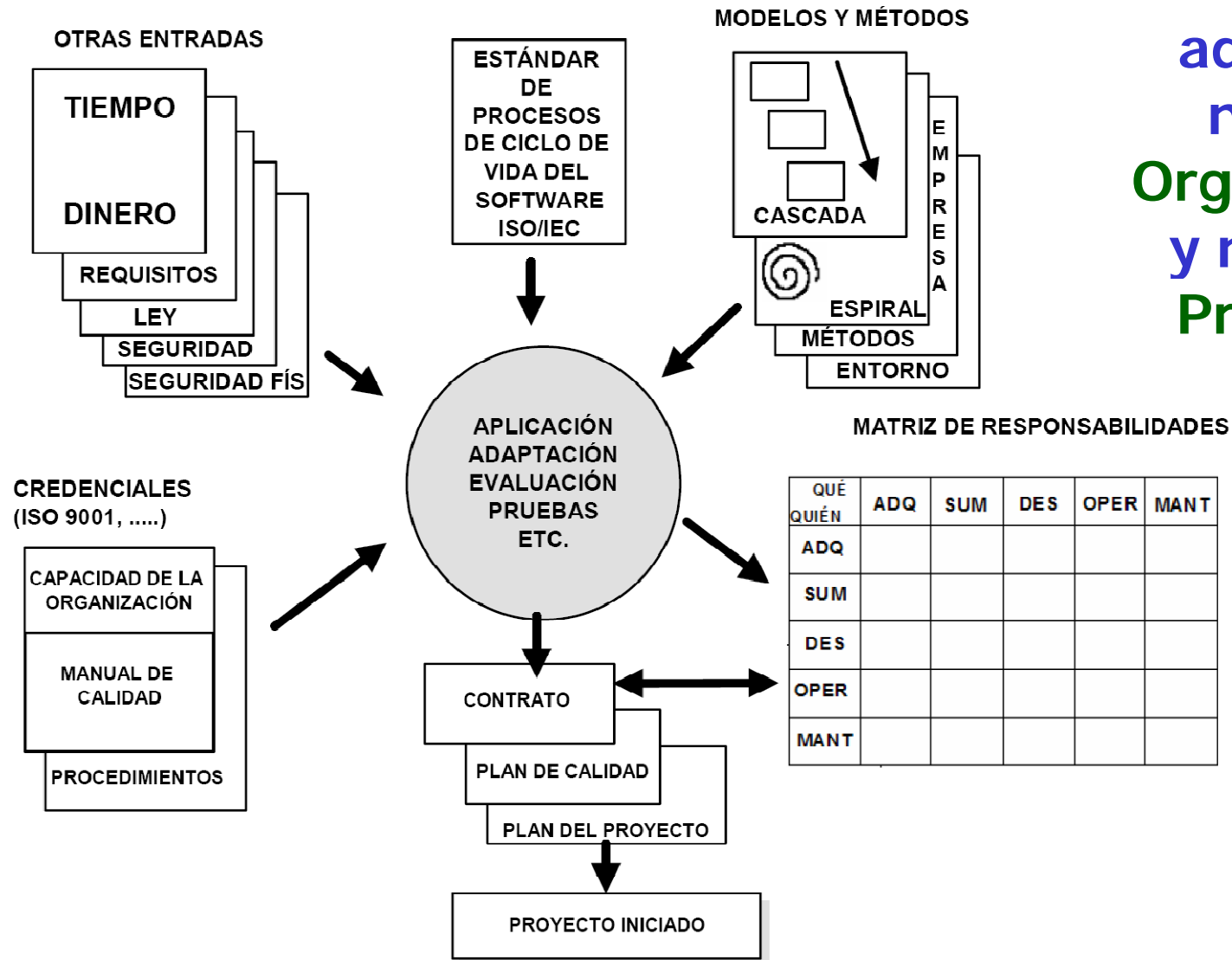
ISO 12207: Procesos Organizacionales



- **Proceso de Gestión:** Organizar, supervisar, y controlar el inicio y el desempeño de cualquier proceso para conseguir sus objetivos de acuerdo a los objetivos de negocio de la organización
- **Proceso de Infraestructura:** Mantener una infraestructura fiable y estable necesaria para cualquier otro proceso.
- **Proceso de Mejora:** Establecer, evaluar, medir, control y mejorar los procesos del ciclo de vida del software.
- **Proceso de Recursos Humanos:** Proporcionar a la organización los recursos humanos adecuados y mantener su competencia, consistente con las necesidades de la empresa
- **Proceso de Gestión de Activos:** Gestionar la vida de los activos reutilizables desde su concepción hasta su retirada.
- **Proceso de Gestión del Programa de Reutilización:** Planificar, gestionar y controlar el programa de reutilización de una organización y explotar de forma sistemática las oportunidades de reutilización.
- **Proceso de Ingeniería del Dominio:** Desarrollar y mantener modelos de dominio, arquitecturas de dominio y activos para el dominio.



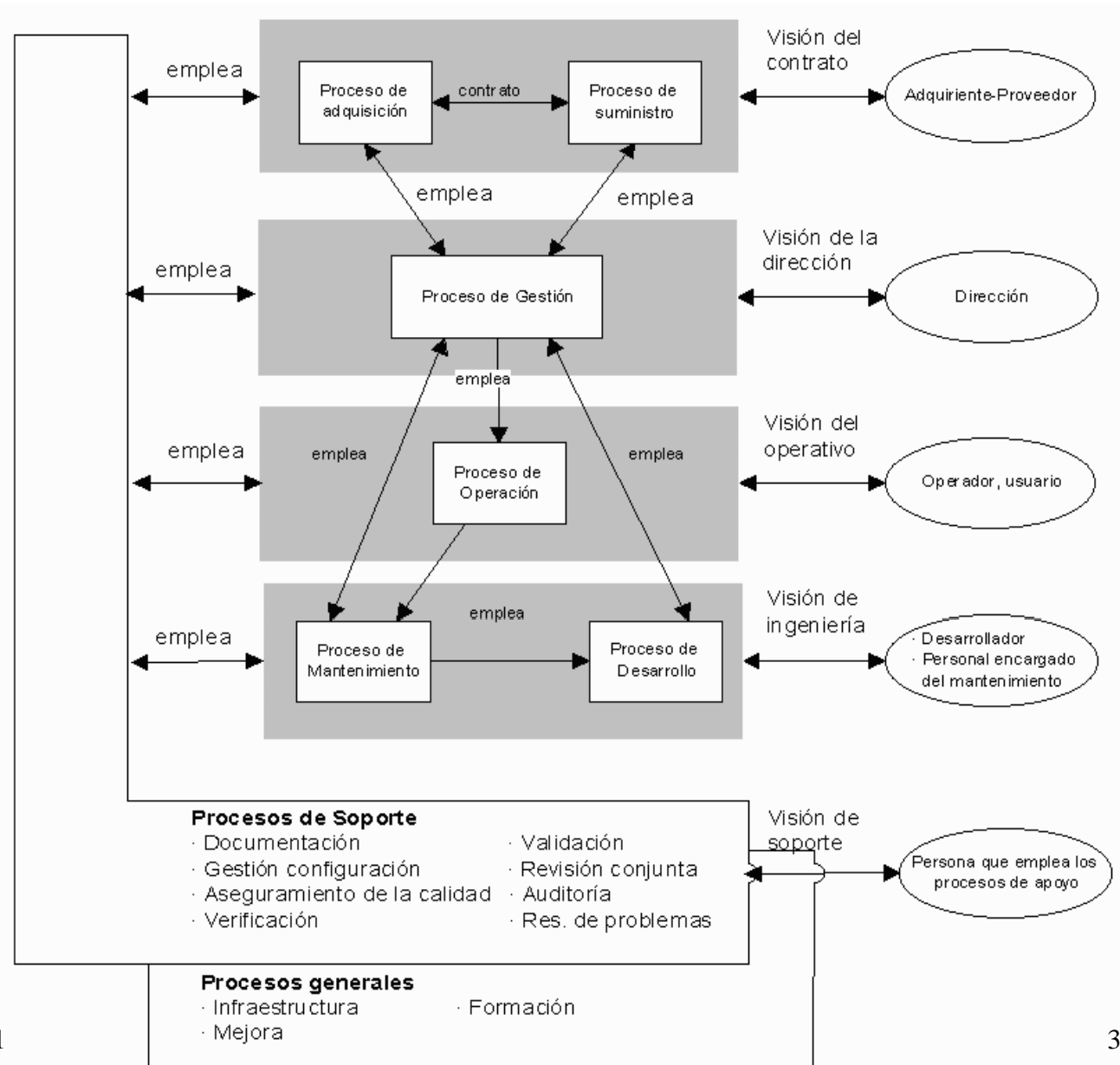
ISO 12207: Proceso de Adaptación



adecuar a
nuestra
Organización
y nuestros
Proyectos

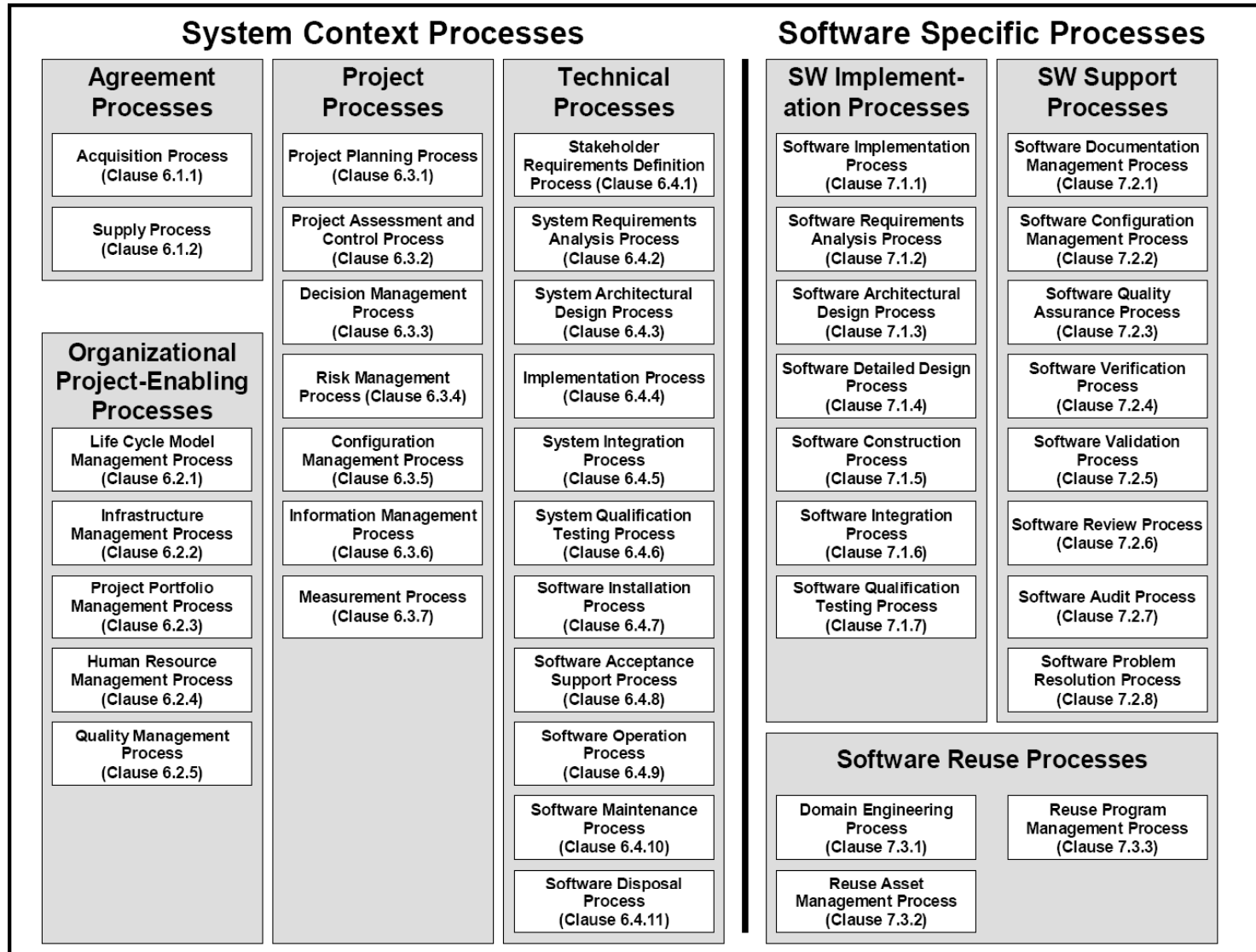


ISO 12207: Procesos y Roles Participantes





ISO 12207: Integración con Sistemas





Ciclos de Vida Tradicionales

- Existen diferentes modelos de ciclo de vida del software que han intentado resolver el problema de crear software.
- El auge de cada uno está asociado a un momento en el tiempo, unas tecnologías determinadas y una ciertas metodologías asociadas.





Modelo en Cascada



- Originalmente propuesto por Royce en 1970 y posteriormente refinado por diversos autores.
- Conocido como **ciclo de vida lineal o básico**.
- Cada fase empieza cuando termina la fase anterior, para lo cual es necesario conseguir todos los objetivos de la etapa previa.
- Ayuda a prevenir que se sobrepasen las fechas de entrega y los costes esperados.
- Al final de cada fase el personal técnico y los usuarios tienen la oportunidad de revisar el progreso del proyecto.



Modelo en Cascada

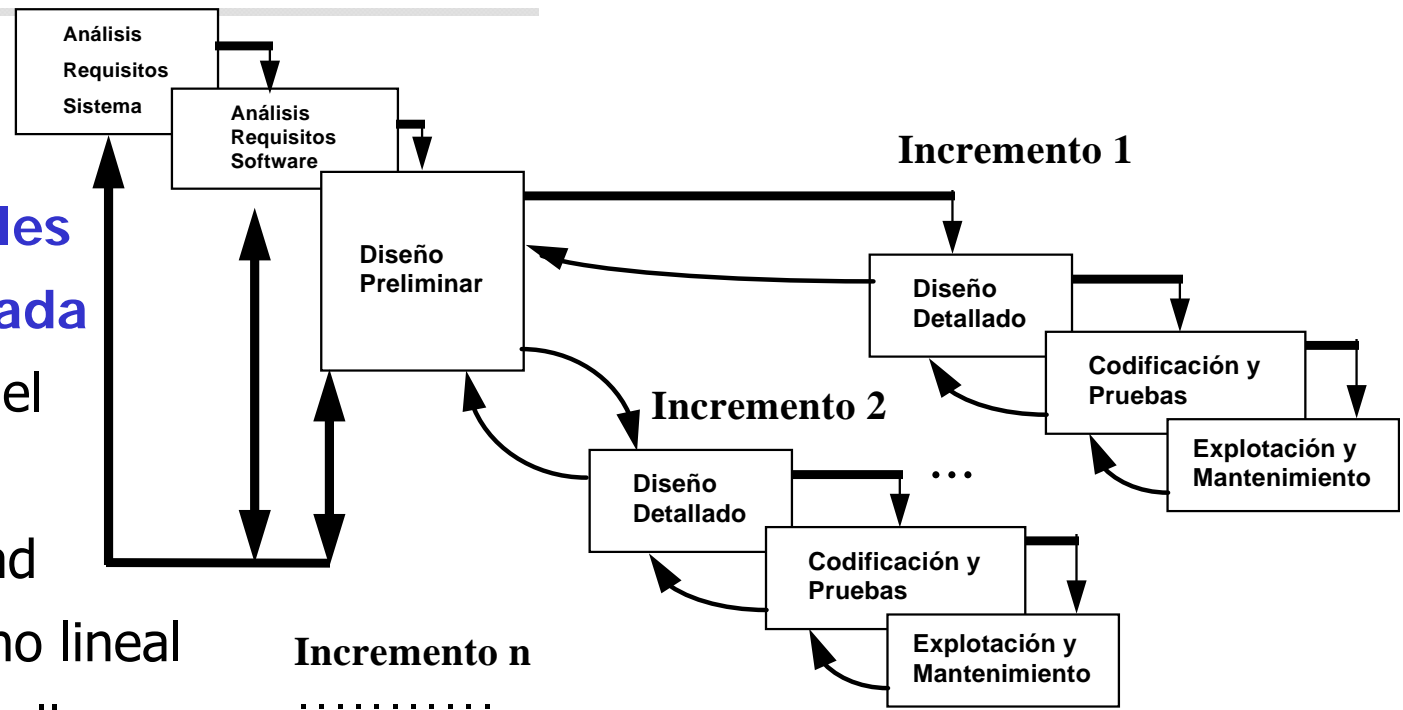
- **Críticas:**

- 👉 No refleja realmente el proceso de desarrollo del software.
- 👉 Se tarda mucho tiempo en pasar por todo el ciclo.
- 👉 Acentúa el fracaso de la industria del software en su comunicación con el usuario final.
- 👉 Se convierten las especificaciones en implementaciones de manera informal.
- 👉 El mantenimiento se realiza en el código fuente.
- 👉 Las revisiones de proyectos de gran complejidad son muy difíciles.
- 👉 Impone una estructura de gestión de proyectos.



Modelo Incremental

- Se aplican **secuencias lineales** de forma **escalonada** mientras progresa el Calendario.
- Corrige la necesidad de una secuencia no lineal de pasos de desarrollo.
- El sistema se crea añadiendo componentes funcionales → **incrementos**.
- El sistema no se ve como una entidad monolítica con una fecha fija de entrega, sino que es una integración de resultados sucesivos obtenidos después de cada iteración.
- Se ajusta a entornos de alta incertidumbre.





Modelo Incremental

- **Ventajas:**

- 👍 Se evitan proyectos largos y se entrega "algo de valor" a los usuarios con cierta frecuencia.
- 👍 El usuario se involucra más.
- 👍 Mayor retorno de la inversión.

- **Inconvenientes:**

- 👎 Difícil de evaluar el coste total.
- 👎 Requiere gestores experimentados.
- 👎 Difícil de aplicar a sistemas transaccionales que tienden a ser integrados y a operar como un todo.
- 👎 Los errores en los requisitos se detectan tarde y su corrección resulta costosa.



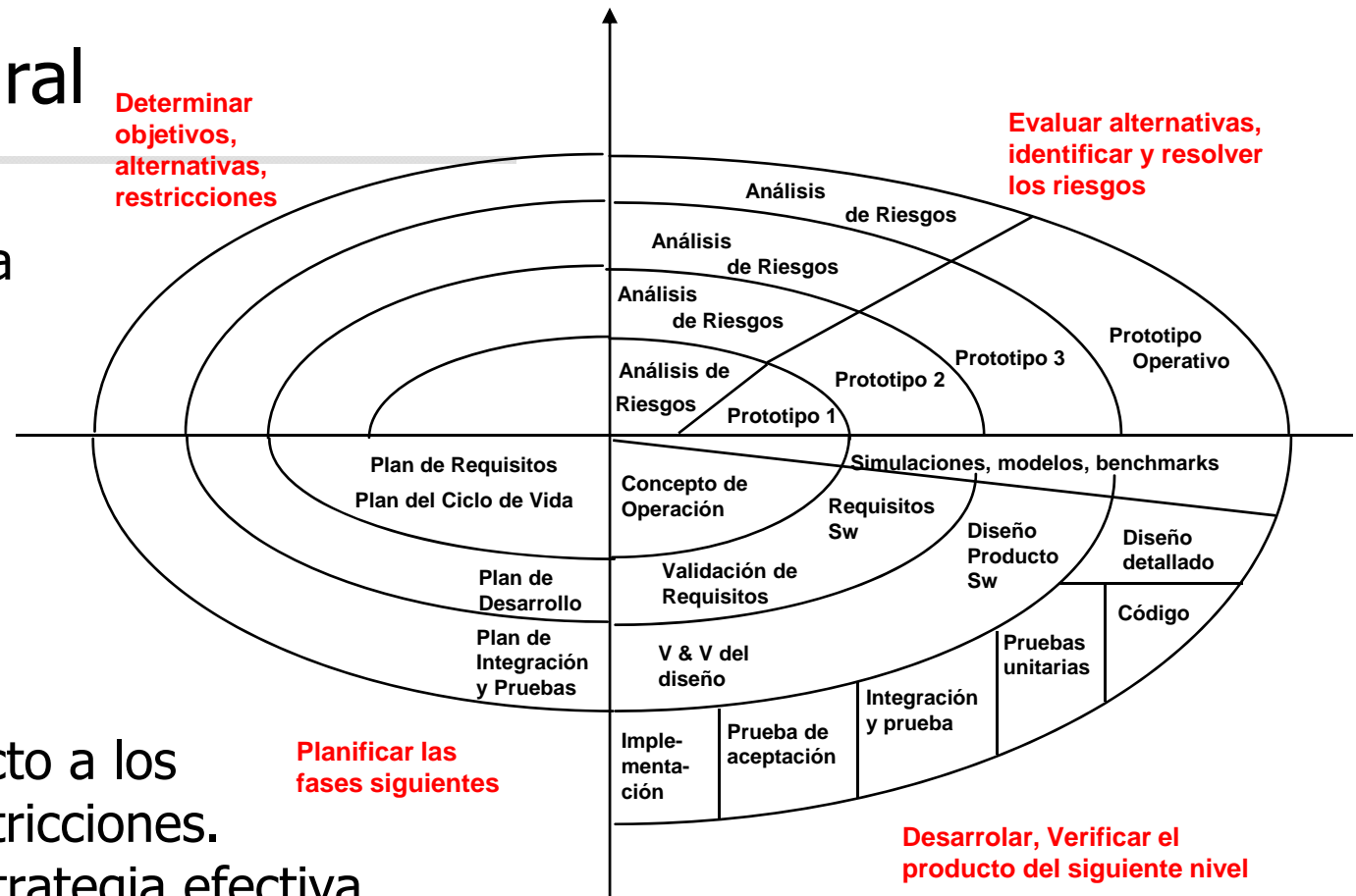
Modelo en Espiral

- Modelo de proceso de software evolutivo que combina la naturaleza iterativa de construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial.
- El software se desarrolla en una serie de versiones incrementales.
 - Durante las primeras iteraciones, la versión incremental podría ser un modelo en papel o un prototipo.
 - Durante las últimas iteraciones, se producen versiones cada vez más completas del sistema diseñado.
- **Diferencias** con los métodos más tradicionales (cascada):
 - Existe un reconocimiento explícito de las **diferentes alternativas** para alcanzar los objetivos de un proyecto.
 - La identificación de **riesgos** asociados con cada una de las alternativas .
 - La división de los proyectos en **ciclos**.
 - El modelo se **adapta** a **cualquier tipo de actividad**.



Modelo en Espiral

- Cada **ciclo** empieza identificando:
 - Los objetivos de la porción correspondiente.
 - Las alternativas.
 - Restricciones.
- Se evalúan las alternativas respecto a los objetivos y las restricciones.
- Se formula una estrategia efectiva para resolver las fuentes de riesgos (simulación, prototipado, etc.).
- Se plantea el próximo prototipo.
- Una vez resueltos los riesgos se sigue el ciclo en cascada.
- Cada ciclo se completa con una revisión que incluye todo el ciclo anterior y el plan para el siguiente.





Modelo en Espiral

- **Ventajas:**

- 👍 Permite acomodar otros modelos.
- 👍 Incorpora objetivos de calidad y gestión de riesgos.
- 👍 Elimina errores y alternativas no atractivas al comienzo.
- 👍 Permite iteraciones, vuelta atrás y finalizaciones rápidas.

- **Inconvenientes:**

- 👎 Es difícil de adaptar a los contratos. Más adecuado para desarrollo interno.



Prototipado

- **Paradigma de construcción de prototipos:**
 - Escuchar al cliente.
 - Construir/revisar maqueta.
 - Probar maqueta.
- Los prototipos tienen una doble función:
 - El cliente ve el producto y refina sus requisitos.
 - El desarrollador comprende mejor lo que necesita hacer.

CLAVE

Definir reglas del juego entre desarrollador y cliente

TIPOS

Rápido

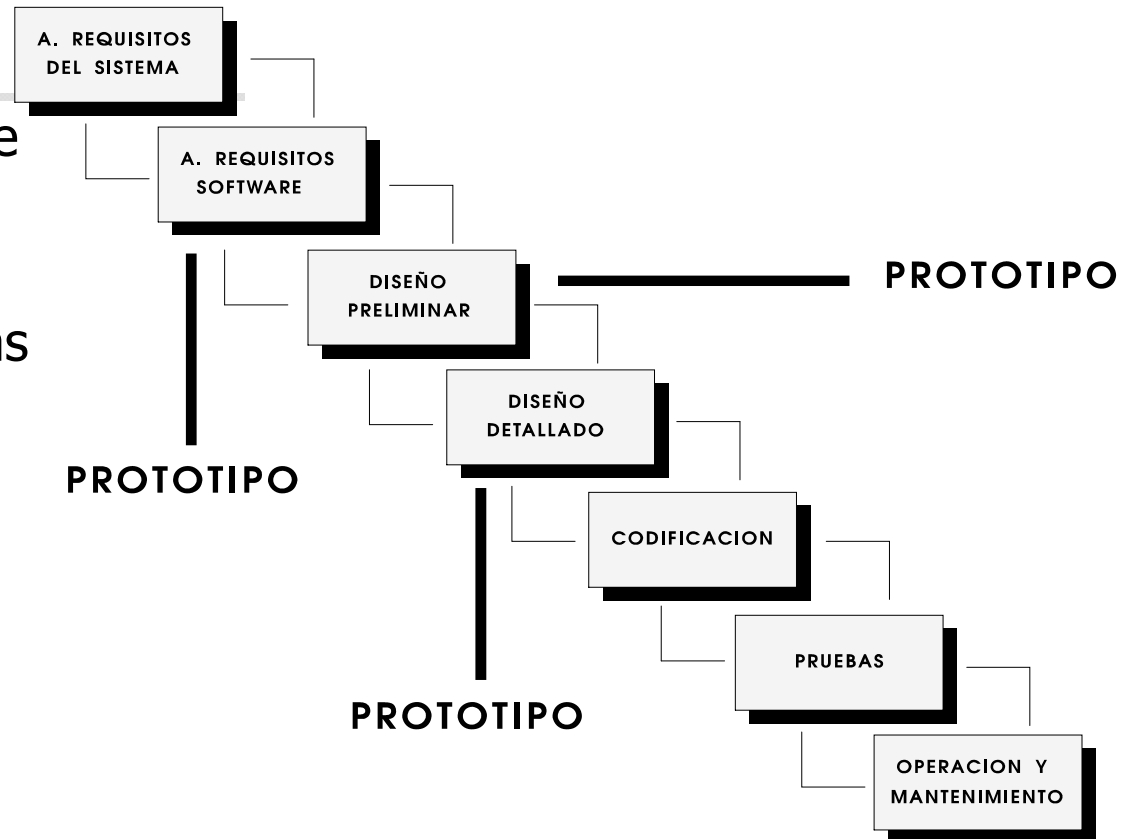
Evolutivo

Operacional



Prototipado - Rápido

- No modifica el flujo del ciclo de vida.
- Reduce el riesgo de construir productos que no satisfagan las necesidades de los usuarios.
- Reduce costos y aumenta la probabilidad de éxito.
- Exige disponer de las herramientas adecuadas.
- No presenta calidad ni robustez.
- Suele utilizarse principalmente en dos áreas:
 - Prototipado de la interfaz de usuario
 - Prototipado del rendimiento.
- El prototipado es un medio excelente para recoger el 'feedback' (realimentación) del usuario final.





Prototipado - Rápido

- Para que sea **efectivo**:
 - Debe ser un sistema con el que se pueda experimentar.
 - Debe ser comparativamente barato (< 10%).
 - Debe desarrollarse rápidamente.
 - Énfasis en la interfaz de usuario.
 - Equipo de desarrollo reducido.
 - Herramientas y lenguajes adecuados.
- Aunque la construcción de prototipos puede ser efectiva, también pueden surgir **problemas**:
 - El **cliente** ve funcionando lo que para él es la primera versión del producto, que ha sido construido con “plastilina y alambres”, y puede desilusionarse al decirle que el sistema aún no ha sido construido.
 - El **desarrollador** puede caer en la tentación de ampliar el prototipo para construir el sistema final, sin tener en cuenta los compromisos de calidad y de mantenimiento que tiene con el cliente.



Prototipado - Evolutivo

- Construcción de una implementación parcial que cubre los requisitos conocidos, para **ir aprendiendo el resto** y, paulatinamente, incorporarlos al sistema.
- **Características:**
 - Reduce el riesgo y aumenta la probabilidad de éxito, en cuanto a la satisfacción del cliente.
 - No se conocen niveles apropiados de calidad y documentación.
 - Problemas de gestión de configuración.
 - Construir software para que pueda ser modificado fácilmente es un "arte desconocido".



Prototipado - Operacional

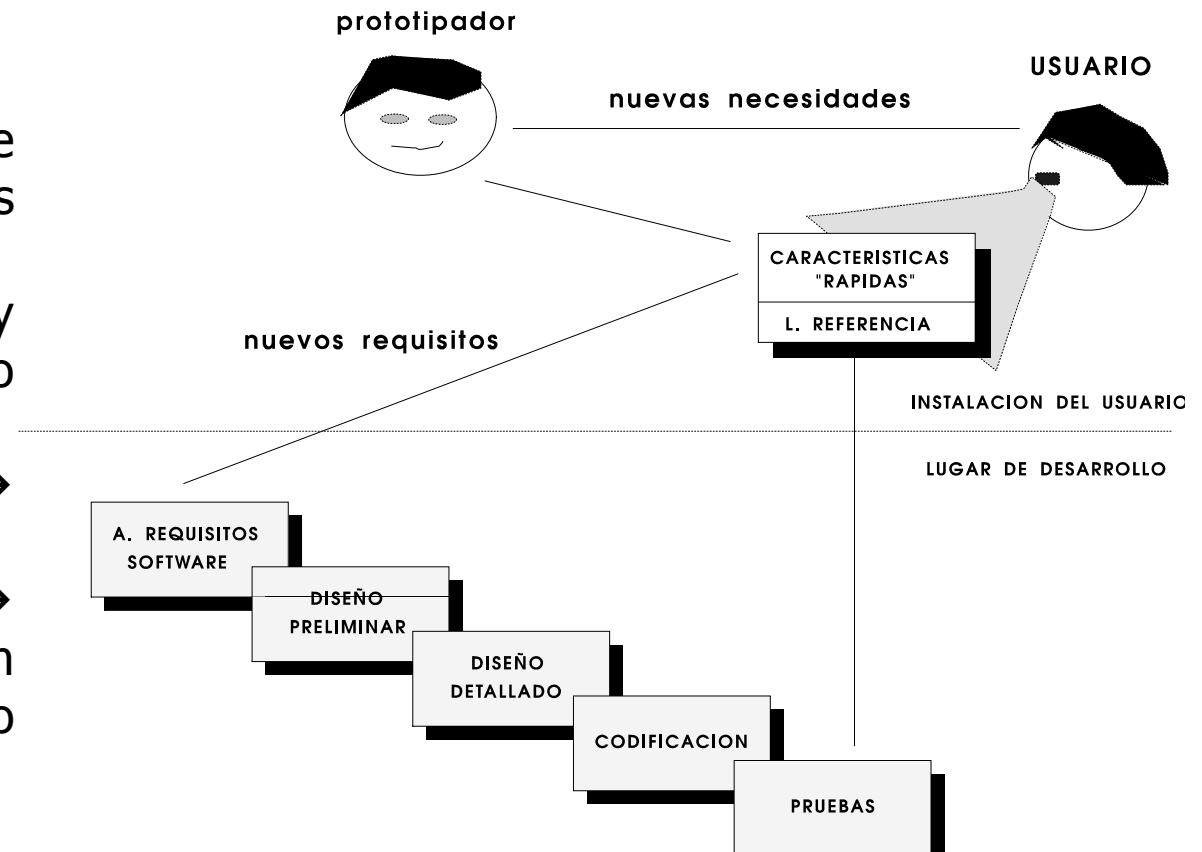
- Es una mezcla entre el prototipado rápido y el evolutivo.
- En algunos sistemas ni el prototipado rápido ni el evolutivo por sí solos son aceptables porque los requisitos son:
 - Críticos al diseño y bien entendidos.
 - No críticos al diseño y pobremente entendidos.
 - Desconocidos.
- El prototipado rápido por sí solo es poco efectivo porque los requisitos pobremente entendidos no son críticos.
- El prototipado evolutivo por sí solo es poco efectivo porque no ayuda a clarificar los requisitos que no se entienden.



Prototipado - Operacional

Modo de trabajo:

- Un prototipo evolutivo se construye con los requisitos bien conocidos.
- El usuario lo maneja y especifica nuevos cambios o detecta problemas.
- Si el usuario dice que "no" → se desechan.
- Si el usuario dice que "sí" → Se implementan y completan para añadirlos al producto evolutivo.



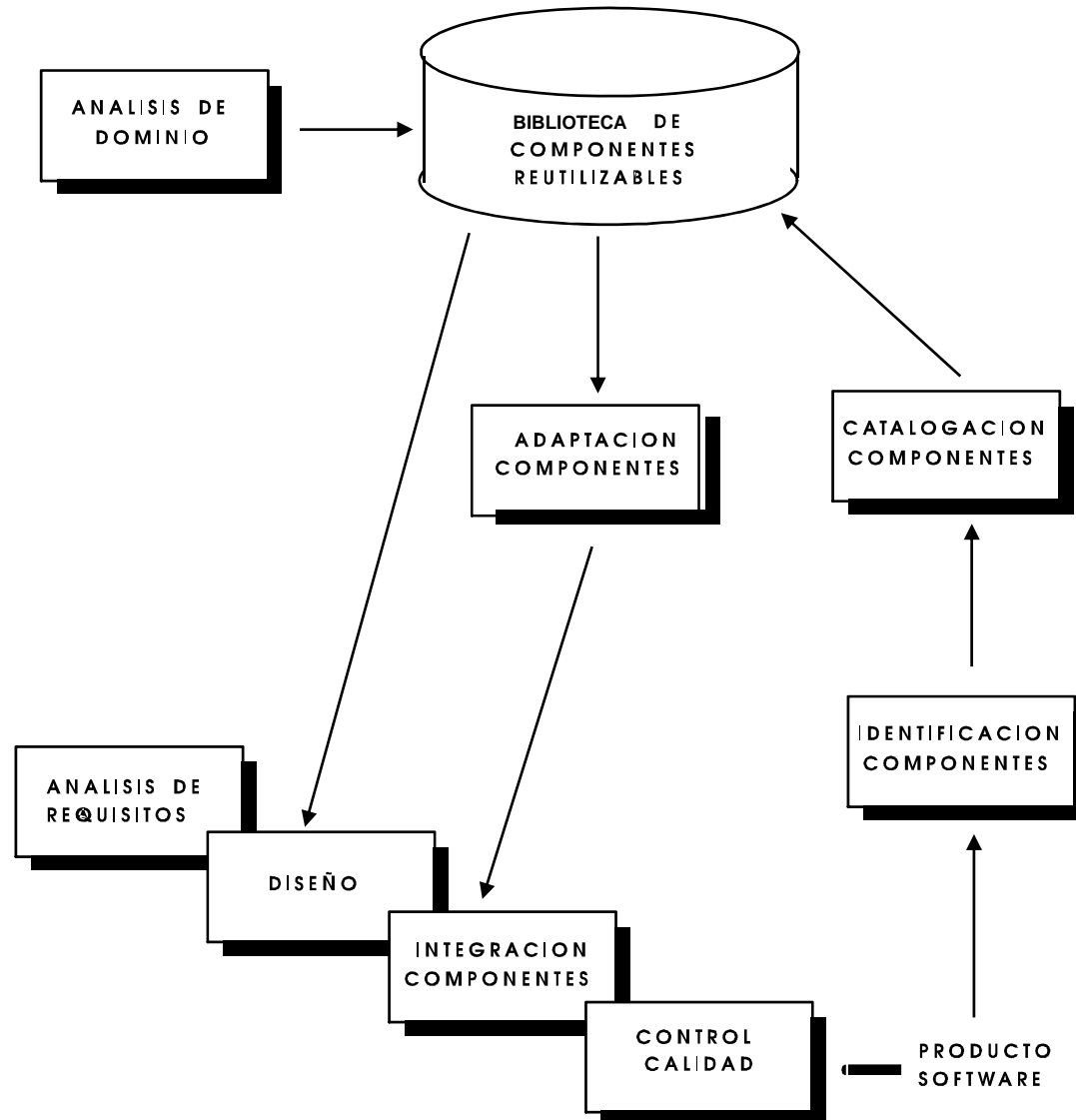


Reutilización

- **Principios de la reutilización:**
 - Existen similitudes entre distintos sistemas de un mismo dominio de aplicación. Los sistemas nuevos se pueden caracterizar por diferencias respecto a los antiguos.
 - El software puede representarse como una combinación de módulos
 - Diseñar aplicaciones = especificar módulos + interrelaciones.
- Procedimiento para **construir un sistema mediante la reutilización de “algo” procedente de algún esfuerzo de desarrollo anterior.**
- En la mayoría de los proyectos de SW existe algo de reutilización. Esto pasa cuando las personas que trabajan en el proyecto conocen diseño o código similares al requerido.
- La reutilización sobre todo se usa en el paradigma OO y supone cambios en el propio ciclo de vida.
- El software se puede construir igual que el hardware, mediante el ensamblaje de piezas (componentes). El uso de los componentes software facilita la reutilización.



Reutilización



Si la reutilización predomina durante el desarrollo se habla de **Desarrollo Basado en Componentes:**

- Análisis de componentes (se buscan componentes adecuados).
- Modificación (se modifican los componentes para satisfacer los requisitos).
- Diseño con reutilización (se diseña o utiliza un nuevo trabajo para el sistema).
- Desarrollo e integración.



Reutilización

- **Ventajas:**

- 👍 Reduce tiempos y costes de desarrollo.
- 👍 Aumenta la fiabilidad.

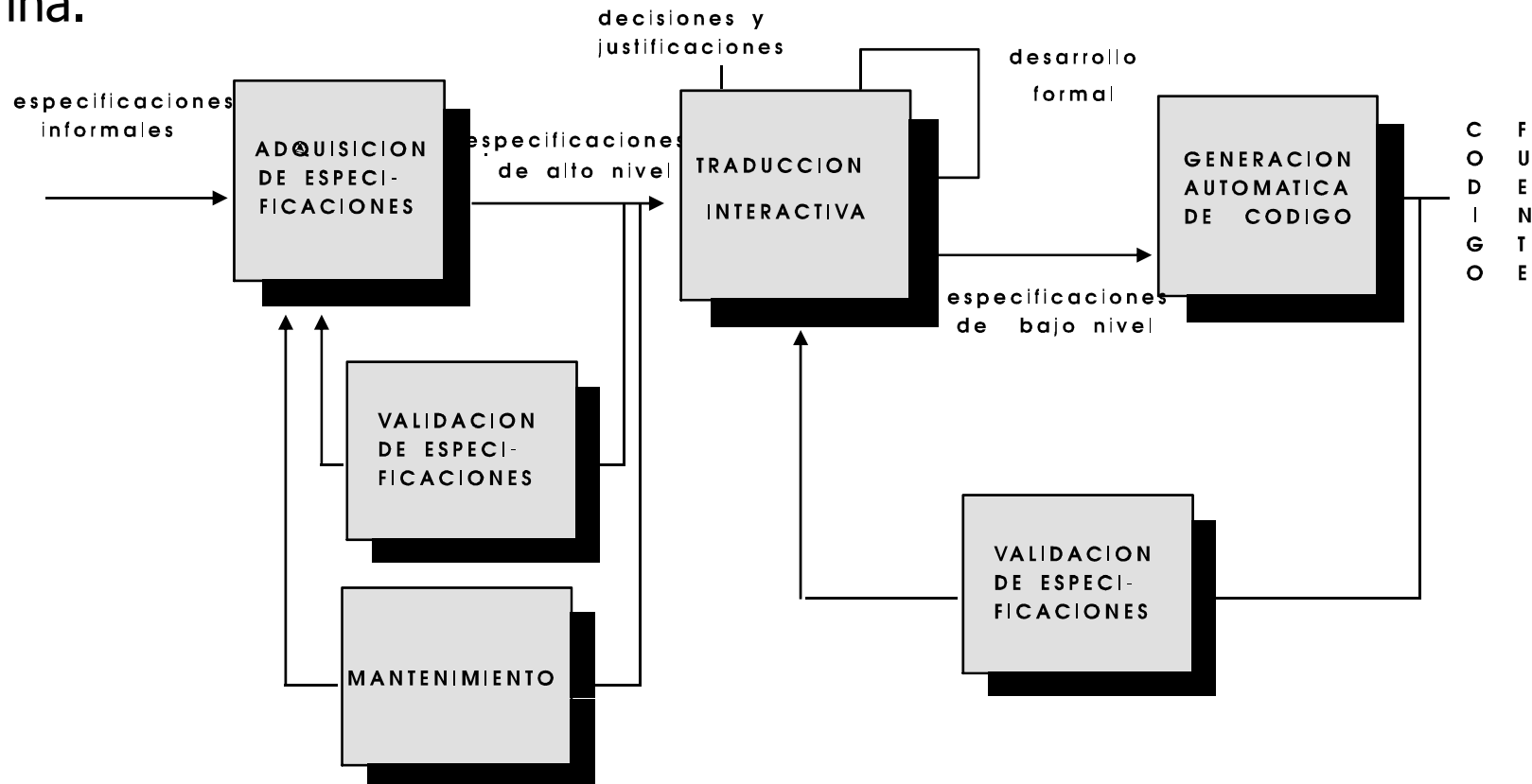
- **Inconvenientes:**

- 👎 Dificultad para reconocer los componentes potencialmente reutilizables.
- 👎 Dificultad de catalogación y recuperación.
- 👎 Problemas de motivación.
- 👎 Problemas de gestión de configuración.



Síntesis Automática

- Los Requisitos se expresan en una **especificación formal** detallada expresada en notación matemática.
- Los procesos (diseño, implementación y pruebas) se reemplazan por un proceso basado en transformaciones donde la especificación formal se refina.





Síntesis Automática

- **Ventajas:**

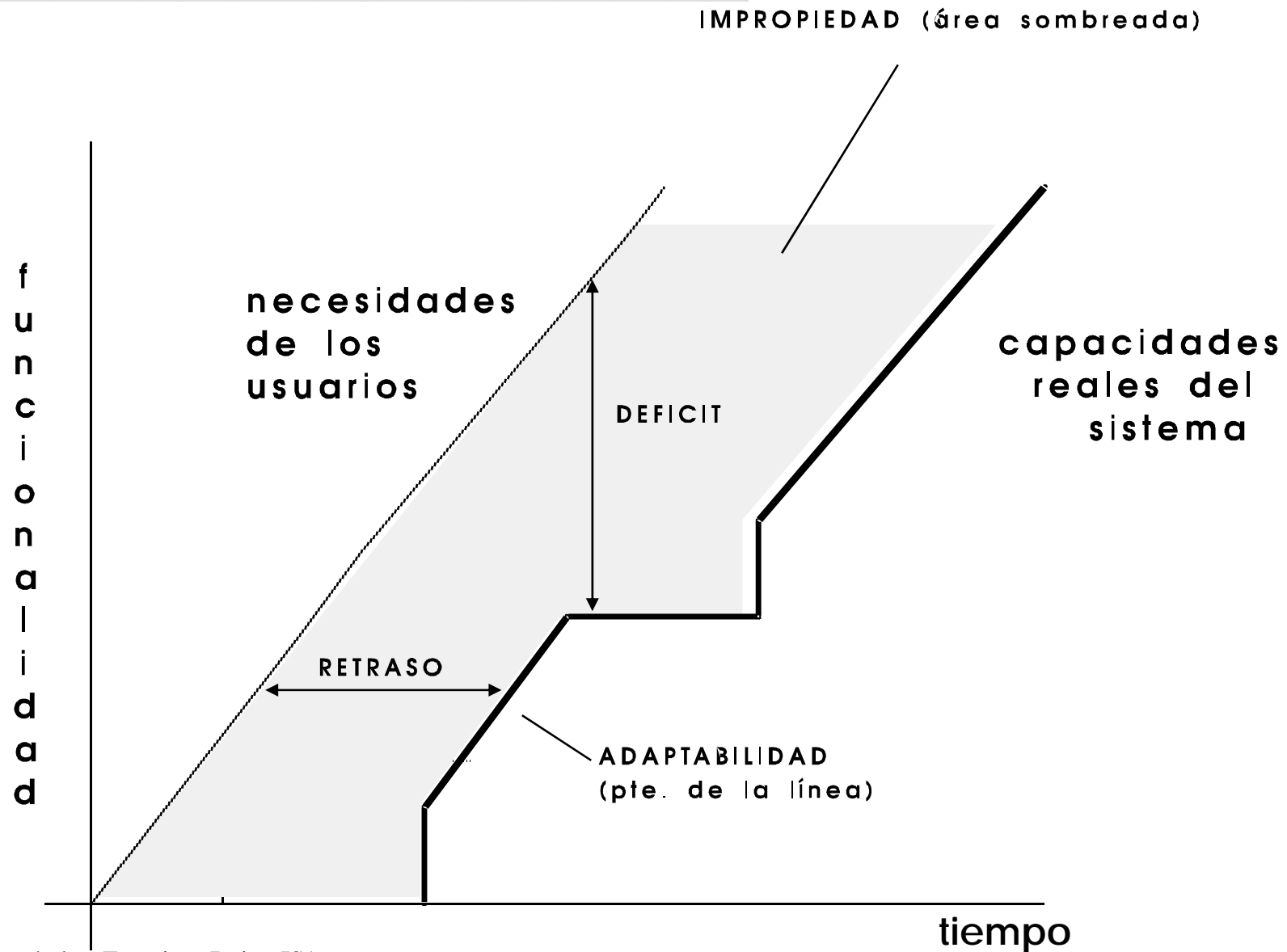
- 👍 Se define el sistema utilizando un lenguaje formal.
- 👍 La implementación es automática, asistida por el ordenador.
- 👍 La documentación se genera de forma automática.
- 👍 El mantenimiento se realiza "por sustitución" en las especificaciones, no mediante "parches".

- **Inconvenientes:**

- 👎 Hay dificultad en la participación del usuario.
- 👎 Los diseños están poco optimizados.

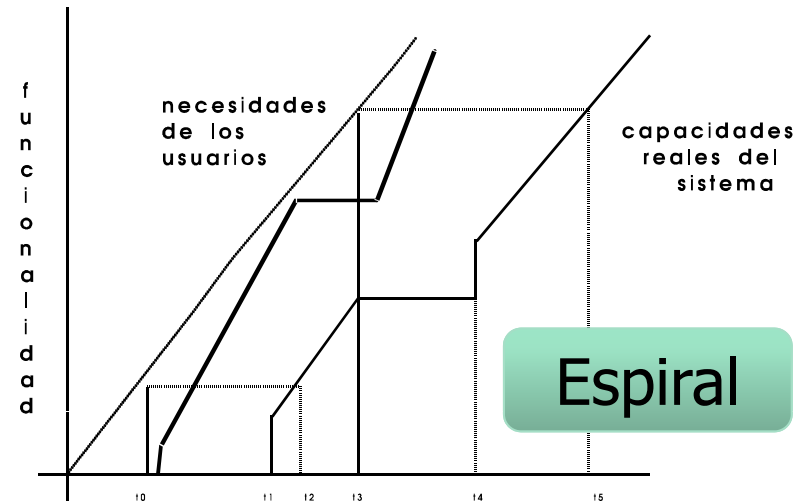
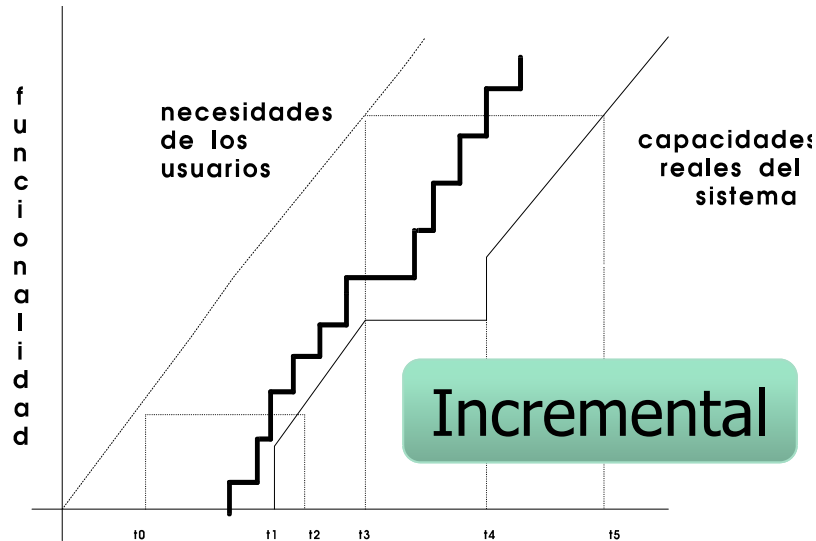
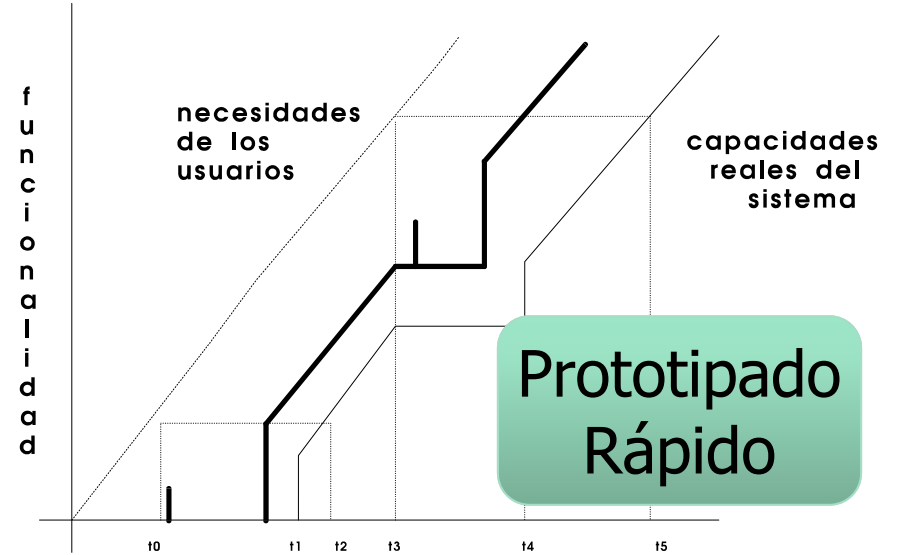
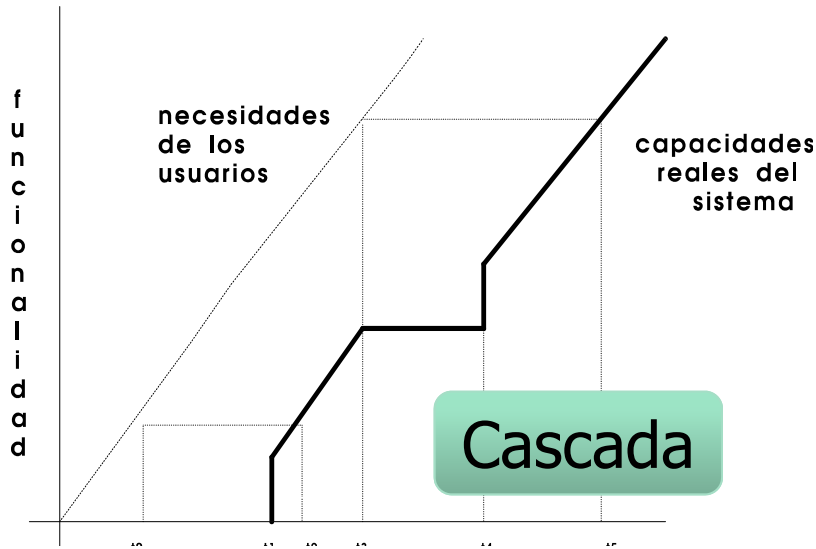


Comparación entre Ciclos de Vida



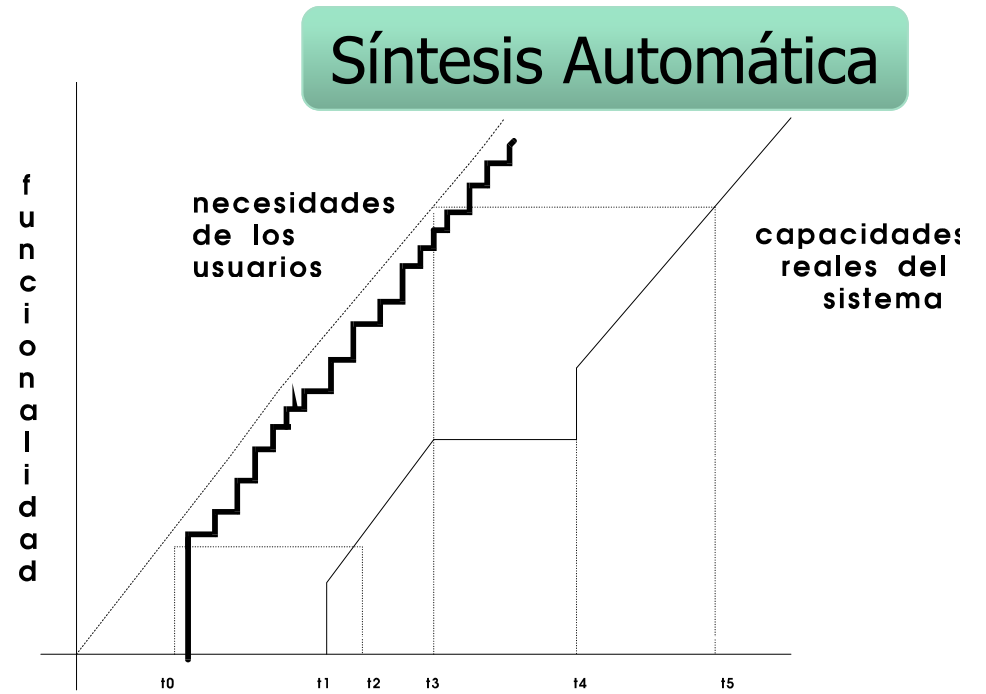
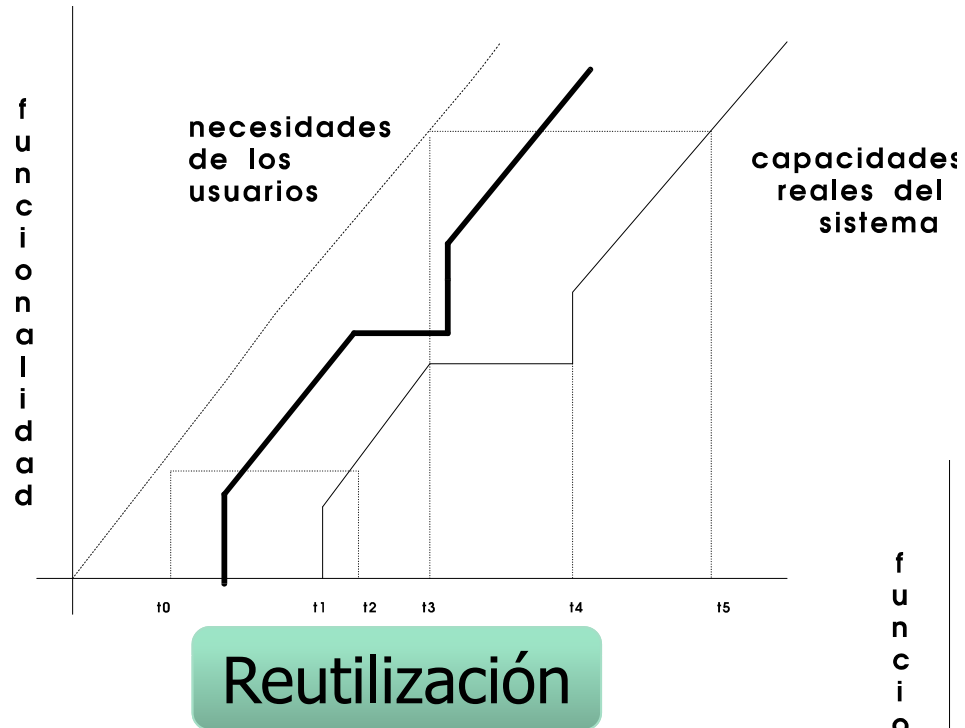


Comparación entre Ciclos de Vida





Comparación entre Ciclos de Vida





Ciclos de Vida para Sistemas OO

- El modelo en cascada no permite aprovechar las ventajas de la tecnología OO.
- Modelos **tradicionales** de ciclo de vida → **Proyecto**.
- Desarrollo **Orientado a Objetos** → **Producto**.
 - Pretende acelerar el desarrollo de sistemas de una manera **iterativa** e **incremental**.
 - Generalizar los componentes para que sean **reutilizables**, por la nueva forma de concebir los lenguajes de programación y su uso.
 - **Eliminación de fronteras entre fases**, ya que éstas se difuminan por
 - La naturaleza iterativa del desarrollo OO.
 - Un alto grado de **iteración y solapamiento**, lo que lleva a una forma de trabajo muy dinámica.

C
I
C
L
O
S

O
O

Agrupamiento

Fuente

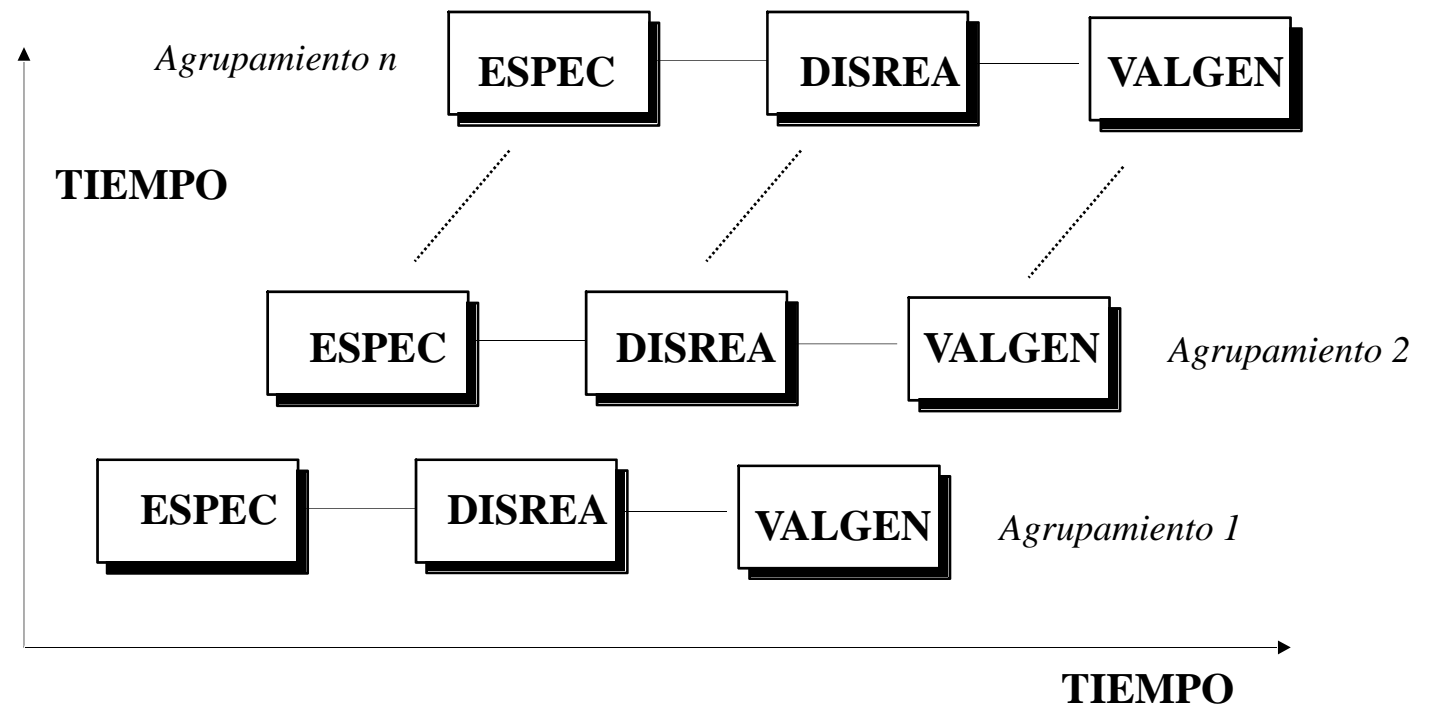
Remolino

Pinball



Ciclos de Vida para OO – Agrupamiento (Meyer, 1990)

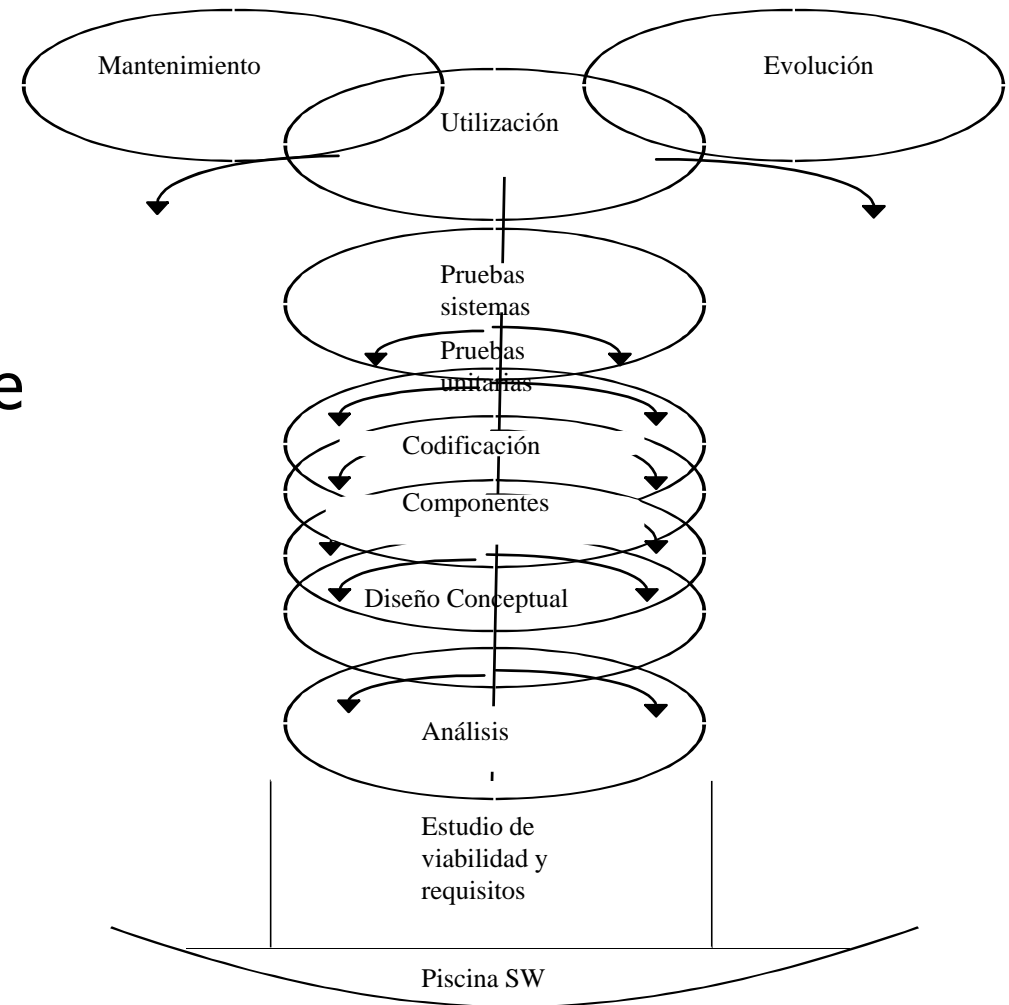
- Adopta Filosofía de Producto vs Proyecto.
- **Agrupamiento:** Conjunto de clases relacionadas con un objetivo común.
- Subciclos de vida, cada uno con:
 - Especificación.
 - Diseño.
 - Realización.
 - Validación.
 - Generalización.





Ciclos de Vida para OO – Fuente (Henderson, Sellers y Edwards, 1990)

- Representa gráficamente:
 - El alto grado de iteración y solapamiento de la OO.
 - Reutilización.
- Aplicable a nivel de clase individual o agrupamientos.





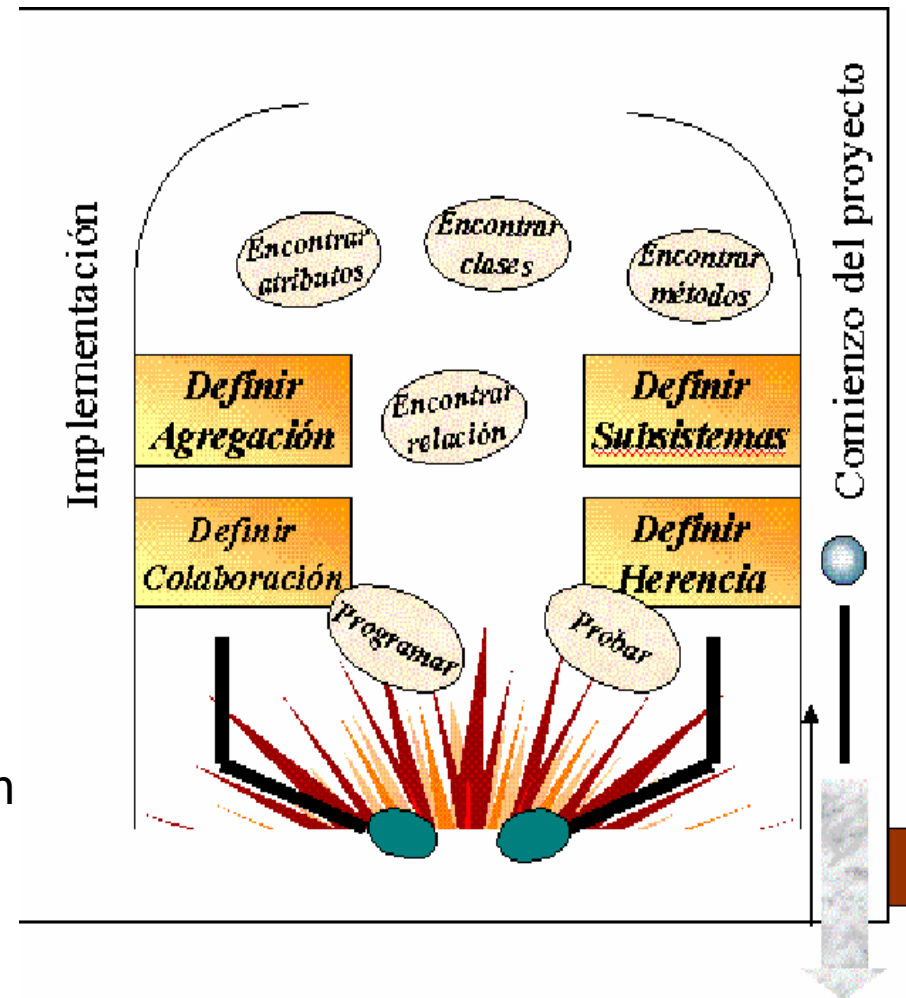
Ciclos de Vida para OO – Remolino (Rumbaugh, 1992)

- Las metodologías de desarrollo no ofrecen una visión real. En la práctica, el desarrollo es desordenado e implica múltiples iteraciones relacionadas.
- El modelo en cascada asume una sola dimensión de iteración: la fase del proyecto.
- Pero puede haber otras **dimensiones** de iteración:
 - Amplitud → Tamaño de desarrollo.
 - Profundidad → Nivel de abstracción o detalle.
 - Madurez → Grado de compleción, corrección y elegancia.
 - Alternativas → Diferentes soluciones a un problema.
 - Alcance → Objetivos del Sistema (requisitos cambiantes).
- Proceso multicíclico no lineal con forma de remolino.



Ciclos de Vida para OO – Pinball (Ambler, 1994)

- El **Pinball** refleja el proceso de desarrollo OO:
 - La **pelota** representa un proyecto completo o un subproyecto.
 - El **jugador** es el equipo de desarrollo.
 - Se procede de forma iterativa a encontrar clases, atributos, métodos e interrelaciones y definir colaboraciones, herencia, agregación y subsistemas.
 - Por último se pasa a la programación, prueba e implementación.
 - La habilidad y la experiencia son los factores más importantes, aunque también se requiere algo de suerte.
- Hay **dos estilos** a la hora de “jugar”:
 - Seguro → Tecnologías y métodos probados.
 - Al límite → Mayor riesgo, más ventajas.





Conjunto de procedimientos, técnicas, herramientas y soporte documental que ayuda a los desarrolladores a realizar nuevo software.

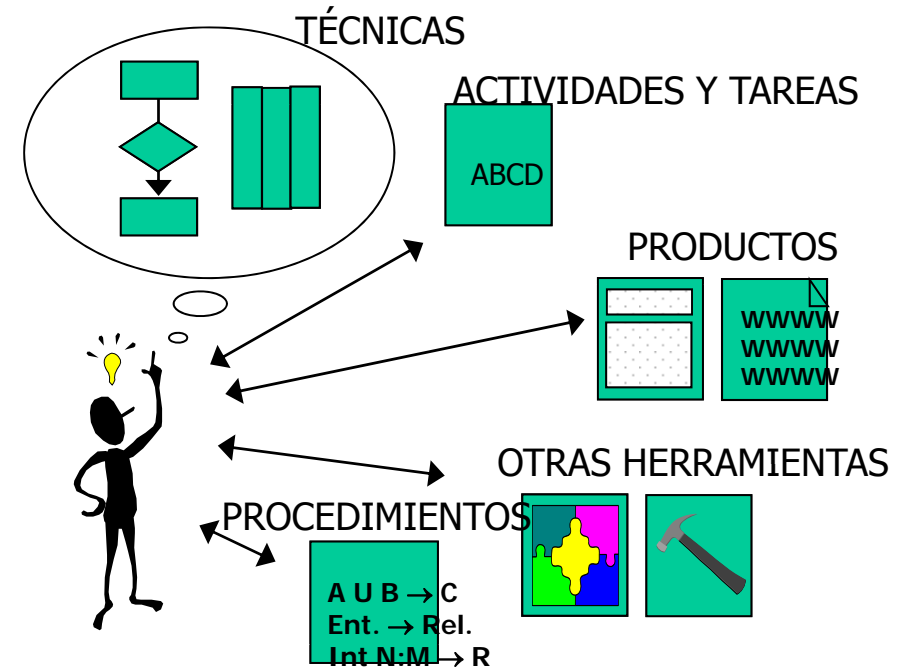
- Por tanto, una metodología representa el **camino a seguir** para **desarrollar software** de manera **sistemática**.
- **Objetivos:**
 - **Mejores Aplicaciones.**
 - Un mejor **Proceso de Desarrollo** que identifique salidas (o productos intermedios) de cada fase de forma que se pueda planificar y controlar los proyectos.
 - Un **Proceso Estándar** en la organización.



Metodologías de Desarrollo de Sw

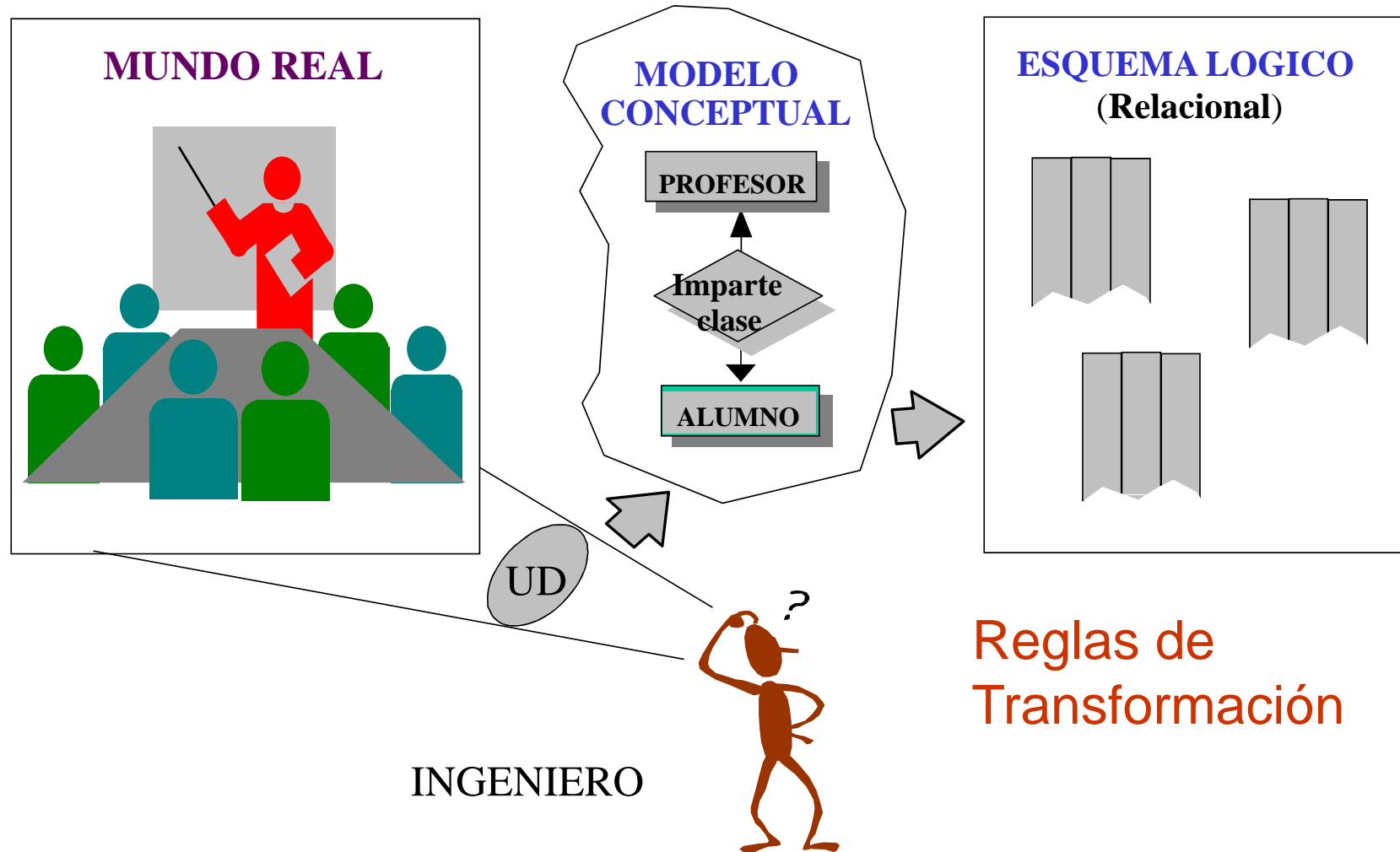
Elementos

- **Actividades y Tareas.**
 - El **Proceso** se descompone hasta el nivel de Actividades y Tareas (actividades elementales).
- **Procedimientos.**
 - Definen la forma de llevar a cabo las **Tareas**.
 - Vínculo de Comunicación entre usuarios y Desarrolladores.
- **Productos.**
 - Obtenidos como resultado de seguir un **Procedimiento**.
 - Pueden ser Intermedios o Finales.
- **Técnicas.**
 - Se utilizan para aplicar un **Procedimiento**
 - Pueden ser Gráficas y/o Textuales
 - Determinan el formato de los **Productos** resultantes en cada **Tarea**
- **Herramientas Software.**
 - Proporcionan soporte a la aplicación de las Técnicas





EJEMPLO: Diseño de Bases de Datos





Metodologías de Desarrollo de Sw

Características Deseables

- **En general:**

- Claridad y facilidad de comprensión.
- Capacidad de soportar la evolución de los sistemas (mantenimiento).
- Facilitar la portabilidad.
- Versatilidad respecto a los tipos de aplicaciones.
- Flexibilidad / Escalabilidad (independencia respecto de la dimensión de los proyectos).
- Rigurosidad.
- Adopción de estándares.

- **En Desarrollo de Software:**

- Existencia de reglas predefinidas.
- Cobertura total del ciclo de desarrollo.
- Verificaciones intermedias.
- Planificación y control.
- Comunicación efectiva.
- Utilización sobre un abanico amplio de proyectos.
- Fácil formación.
- Herramientas CASE.
- Actividades que mejoren el proceso de desarrollo.
- Soporte al mantenimiento.
- Soporte de la reutilización de software.



Metodologías de Desarrollo de Sw

Conceptos Relacionados

- Ciclo de Vida vs Metodología.
 - Una **Metodología** puede seguir uno o varios modelos de Ciclo de Vida.
 - Un **Ciclo de Vida** indica qué obtener, pero no cómo.
 - Metodología \cong Ciclo de Vida + ¿cómo?
 - Detalle de técnicas, procedimientos y artefactos.

**METODOLOGÍA = CICLO DE VIDA + colección de
MÉTODOS para llevarlo a cabo**

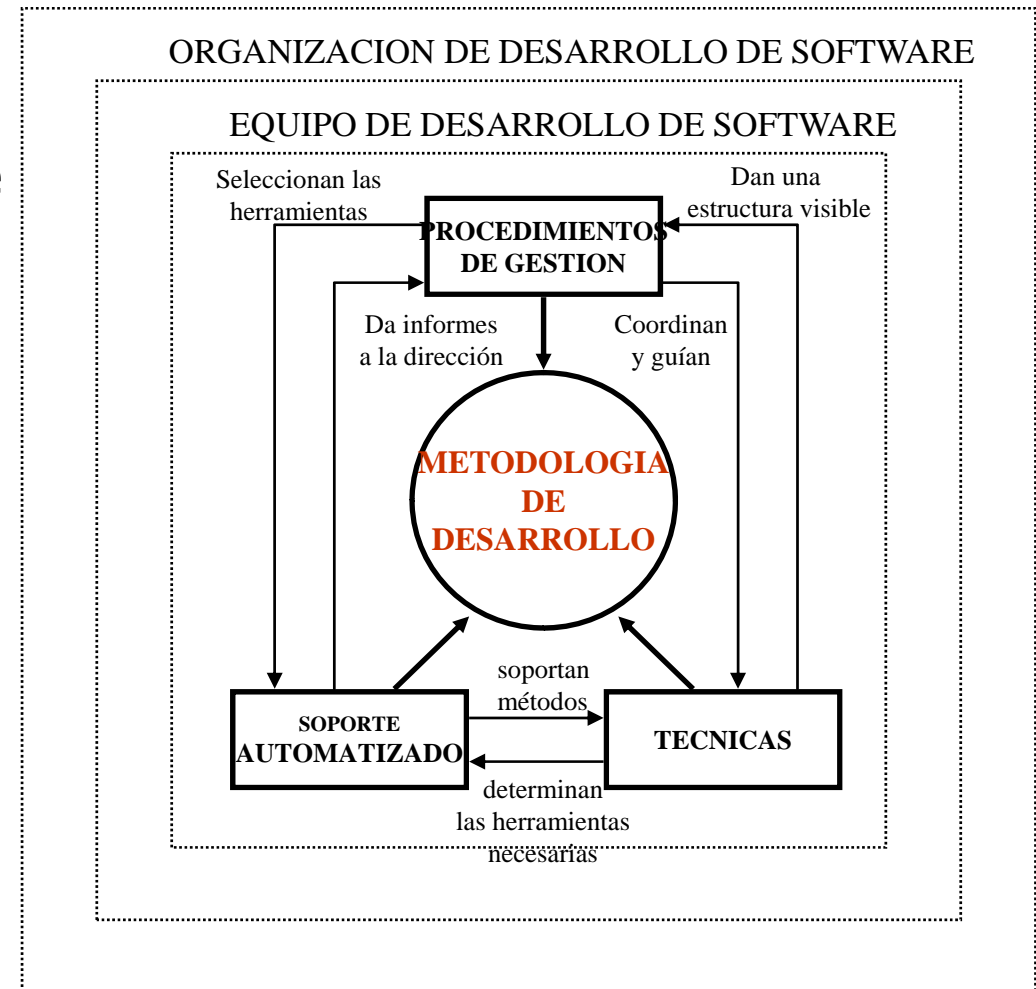


Metodologías de Desarrollo de Sw

Impacto en el Entorno

- Opciones para la implantación de Metodologías:
 - Seleccionar entre un gran número de posibilidades y combinaciones de métodos de gestión, técnicas de desarrollo y soporte automatizado, para crear y desarrollar una Metodología de Desarrollo Software específica.
 - Analizar y evaluar las metodologías existentes y seleccionar la que más se adapte a las necesidades.
- Factores que influyen en las metodologías:
 - Tamaño y estructura de la organización.
 - Tipo de aplicaciones a desarrollar.

ENTORNO DE DESARROLLO DE SOFTWARE





Tipos de Metodologías

Estructuradas

Orientadas a Objetos

Ágiles



Tipos de Metodologías - Estructuradas

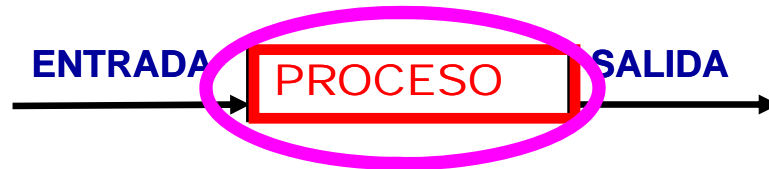
- Proponen la creación de **modelos del sistema que representan**:
 - Los **procesos**.
 - Los **flujos**.
 - Las **estructuras de los datos** .
- Enfoque Top-Down
 - Desde una visión general hasta un nivel de abstracción más sencillo.





Tipos de Metodologías - Estructuradas

Orientadas a Procesos



Se apoyan en técnicas gráficas para obtener una **ESPECIFICACIÓN ESTRUCTURADA**:

- Modelo gráfico, particionado, descendente y jerárquico de los procesos del sistema y de los datos utilizados por éstos.
- Componentes:
 - Diagrama de Flujo de Datos
 - Diccionario de Datos
 - Especificaciones de Procesos

Técnicas de Análisis Estructurado Orientadas a Procesos

FASES DEL ANALISIS ESTRUCTURADO

Método de DeMarco	Método de Gane y Sarson
<ol style="list-style-type: none">1. Construir el modelo físico actual (DFD físico actual)2. Construir el modelo lógico actual (DFD lógico actual)3. Derivación del nuevo modelo lógico4. Crear un conjunto de modelos físicos alternativos5. Estimar los costes y tiempos de cada opción6. Seleccionar un modelo7. Empaquetar la especificación	<ol style="list-style-type: none">1. Construir el modelo lógico actual (DFD lógico actual)2. Construir el modelo del nuevo sistema: elaborar una especificación estructurada y construir un modelo lógico de datos en tercera forma normal que exprese el contenido de los almacenes de datos.3. Seleccionar un modelo lógico4. Crear el nuevo modelo físico del sistema5. Empaquetar la especificación

Metodología de **Yourdon/Constantine** (Análisis+Diseño orientados a Procesos):

- *Realizar los DFD del sistema*
- *Realizar el diagrama de estructuras*
- *Evaluar el diseño*
- *Preparar el diseño para la implantación.*



Tipos de Metodologías - Estructuradas

Orientadas a Datos



- Los datos son más estables que los procesos y pueden ser jerárquicos o no.
- La estructura de control del programa debe ser jerárquica o no (en función de los datos) y se debe derivar de la estructura de datos del programa.
- El proceso de diseño consiste en definir primero las estructuras de los datos de entrada y salida, mezclarlas todas en una estructura, jerárquica o no, de programa y después ordenar detalladamente la lógica procedimental para que se ajuste a esta estructura.
- El diseño lógico debe preceder y estar separado del diseño físico.



Tipos de Metodologías - Orientadas a Objetos

- Cambian los principios de las metodologías estructuradas:
 - **Estructurado**: Examinar el sistema desde las funciones y tareas.
 - **OO**: Modelado del Sistema examinando el dominio del problema como un conjunto de **objetos** que **interactúan** entre sí.
 - **Objetos**: Encapsulan **Funciones + Datos**.
- Enfoques OO:
 - **“Revolucionarios” o “Puros”**
 - La OO se entiende como un cambio profundo de las metodologías estructuradas que se ven como obsoletas.
 - OOD (Booch), CRC/RDD (Wirfs-Brock).
 - **“Sintetistas” o “Evolutivos”**
 - **Án**alisis y **Diseño Estructurado** se consideran como la base para el desarrollo OO.
 - OMT, RUP.



Tipos de Metodologías - Ágiles

- Simplifican la complejidad de otras metodologías haciendo que la carga de gestión y control sea más liviana.
 - La Agilidad es un aspecto que se puede incorporar a las metodologías estructuradas u OO.
 - Entre las más conocidas se encuentran:
 - XP (eXtreme Programming).
 - SCRUM.
 - RAD (Rapid Application Development).
- Reducir los esfuerzos que no están centrados en el puro código también tiene inconvenientes.



APÉNDICE A: Evolución Metodologías de Desarrollo de Sw

- Hay una larguísima lista de metodologías de desarrollo de software.
- En los últimos 30 años se ha desarrollado software siguiendo tres filosofías principales :
 - **Convencional**
 - **Estructurada**
 - **Orientada a Objetos**

AÑO	METODOLOGÍA
1968	Conceptos sobre la programación estructurada de DIJKSTRA
1974	Técnicas de programación estructurada de WARNIER y JACKSON
1975	Primeros conceptos sobre diseño estructurado de MYERS y YOURDON
1977	Primeros conceptos sobre análisis estructurado GANE y SARSON
1978	Análisis estructurado: DEMARCO y WEINBERG Nace MERISE
1981	SSADM (versión inicial) Information Engineering (versión inicial)
1985	Análisis y Diseño estructurado para sistemas de tiempo real de WARD y MELLOR
1986	SSADM Versión 3
1987	Análisis y Diseño estructurado para sistemas de tiempo real de HATLEY y PIRHBAY
1989	MEIRICA (versión inicial)
1990	SSADM Versión 4
1993	MEIRICA Versión 2
1995	MEIRICA Versión 2.1
1998	MÉTRICA Versión 3



APÉNDICE A: Evolución Metodologías de Desarrollo de Sw

Desarrollo Convencional:



- Años 50.
- Desarrollo artesanal y ausencia de Metodología.
- Enfocado en la Tarea de Programación.
- Inconvenientes:
 - Los resultados finales son impredecibles.
 - No hay forma de controlar lo que está sucediendo en el Proyecto
 - Los cambios organizativos afectan negativamente al proceso de desarrollo.
 - El éxito de los proyectos se basa mucho en la figura del “**héroe**” y los héroes siempre acaban cansándose.



APÉNDICE A: Evolución Metodologías de Desarrollo de Sw

Desarrollo Estructurado:

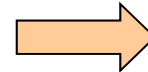
- Programación Estructurada [Años 60 (entorno académico), mediados 70 (industria)]



- Normas para escribir código
- Facilitar comprensión de Programas
- Normas para la aplicación de estructuras de datos y de control

Convencional

```
10      CLS
20      A=10
30      INPUT B
40      IF B=A THEN GOTO 50 ELSE GOTO 70
50      PRINT "A Y B SON IGUALES"
60      GOTO 100
70      IF A>B THEN GOTO 80 ELSE GOTO 90
80      B= B + 1; GOTO 40
90      B= B - 1; GOTO 40
100     END
```



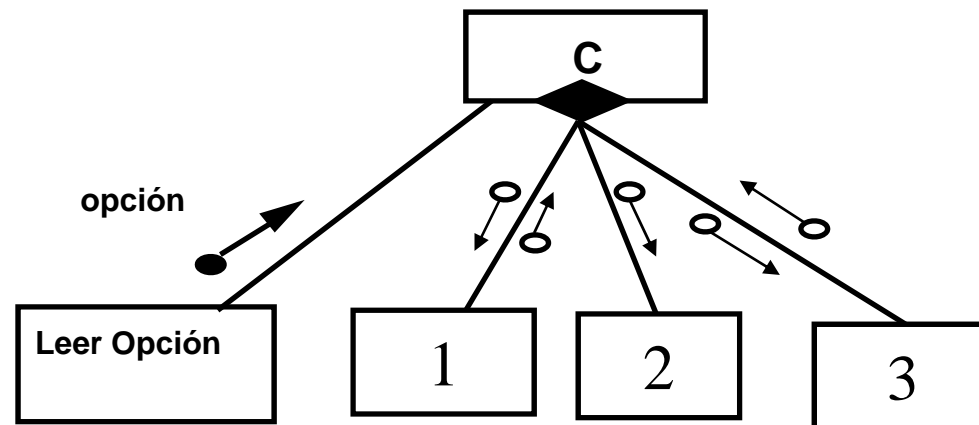
```
PROGRAM NUMEROSIGUALES                                Estructurado
BEGIN
  CLEARSCREEN;
  A :=10 ;
  INPUT B;
  REPEAT
    IF B=A THEN PRINT "A Y B SON IGUALES"
    ELSE REDUCEDIFERENCIA(A,B);
  UNTIL B=A;
END;
PROCEDURE REDUCEDIFENCIA(A,B);
BEGIN
  IF A>B THEN B:= B+1
  ELSE B:= B - 1
END
```



APÉNDICE A: Evolución Metodologías de Desarrollo de Sw

Desarrollo Estructurado:

- **Diseño Estructurado** (mitad años 70)
 - Mayor nivel abstracción (independencia del lenguaje programación)
 - Elemento básico de diseño: Módulo
 - Modularidad. Medidas de Calidad de Programas





APÉNDICE A: Evolución Metodologías de Desarrollo de Sw

Desarrollo Estructurado:

- **Análisis Estructurado (finales años 70)**
 - Previamente: Descripción Narrativa Requisitos → Especificaciones:
 - Monolíticas.
 - Redundantes.
 - Ambiguas.
 - Imposibles de Mantener.
 - Se obtienen **Especificaciones Funcionales**:
 - Gráficas.
 - Particionadas.
 - Mínimamente redundantes.



APÉNDICE A: Evolución Metodologías de Desarrollo de Sw

Desarrollo Orientado a Objetos:

- ❖ Esencia: Identificación y organización de conceptos del dominio de la aplicación y no tanto de su representación final en un lenguaje de programación
- ❖ Años 80
- ❖ Trata **Funcionalidad y Datos** de forma conjunta.
- ❖ Principios:
 - ✓ Abstracción
 - ✓ Ocultación de Información (Encapsulamiento)
 - ✓ Modularidad
- ❖ Las técnicas estructuradas han influido en estas metodologías.



APÉNDICE B: Ejemplos de Metodologías

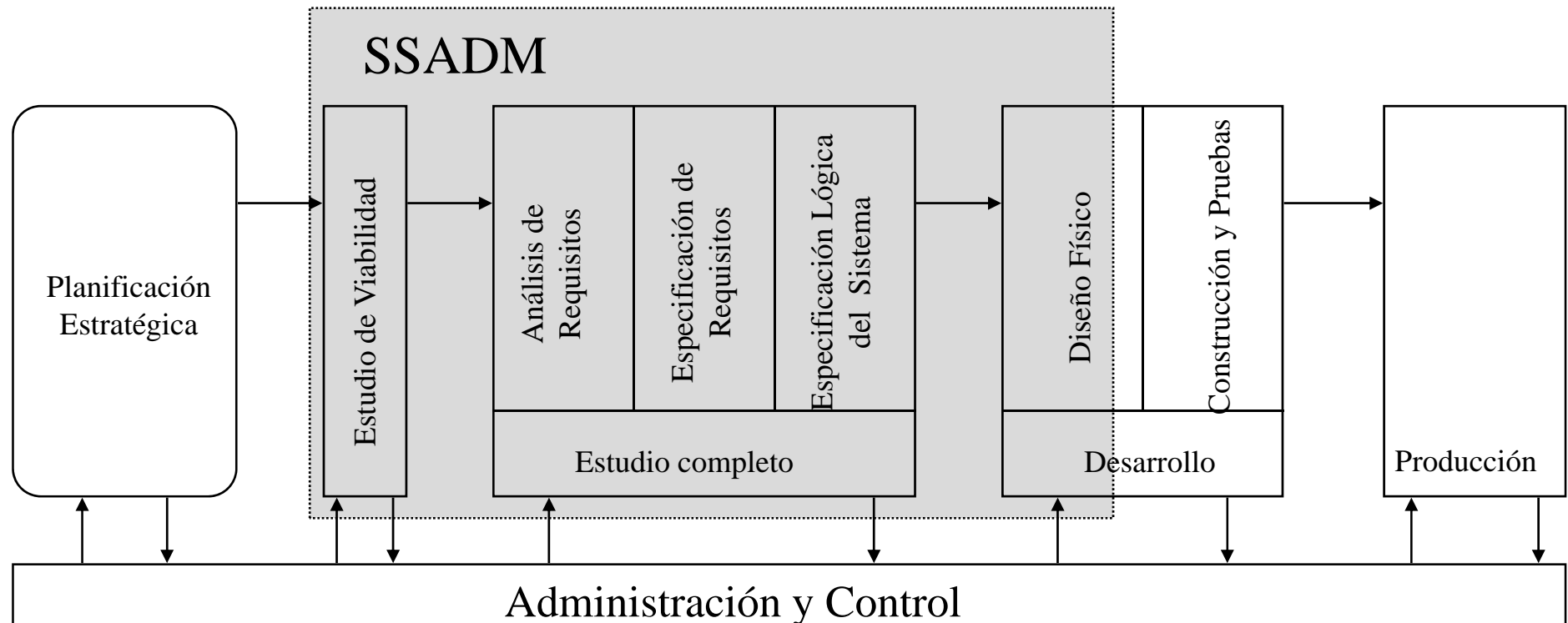
- Metodología **MERISE**. Administración Pública Francia (1976)
- Fases:
 - Estudio preliminar → Estudio detallado → Implementación → Realización y puesta en marcha.

NIVELES	DATOS	TRATAMIENTOS
CONCEPTUAL	Modelo Conceptual de Datos	Modelo Conceptual de Tratamientos
ORGANIZATIVO	Modelo Lógico de Datos	Modelo Organizativo de Tratamientos
FÍSICO	Modelo Físico de Datos	Modelo Operativo de Tratamientos



APÉNDICE B: Ejemplos de Metodologías

- Metodología **SSADM** Structured Systems Analysis and Design Method. Administración Pública Reino Unido (1980).





APÉNDICE B: Ejemplos de Metodologías

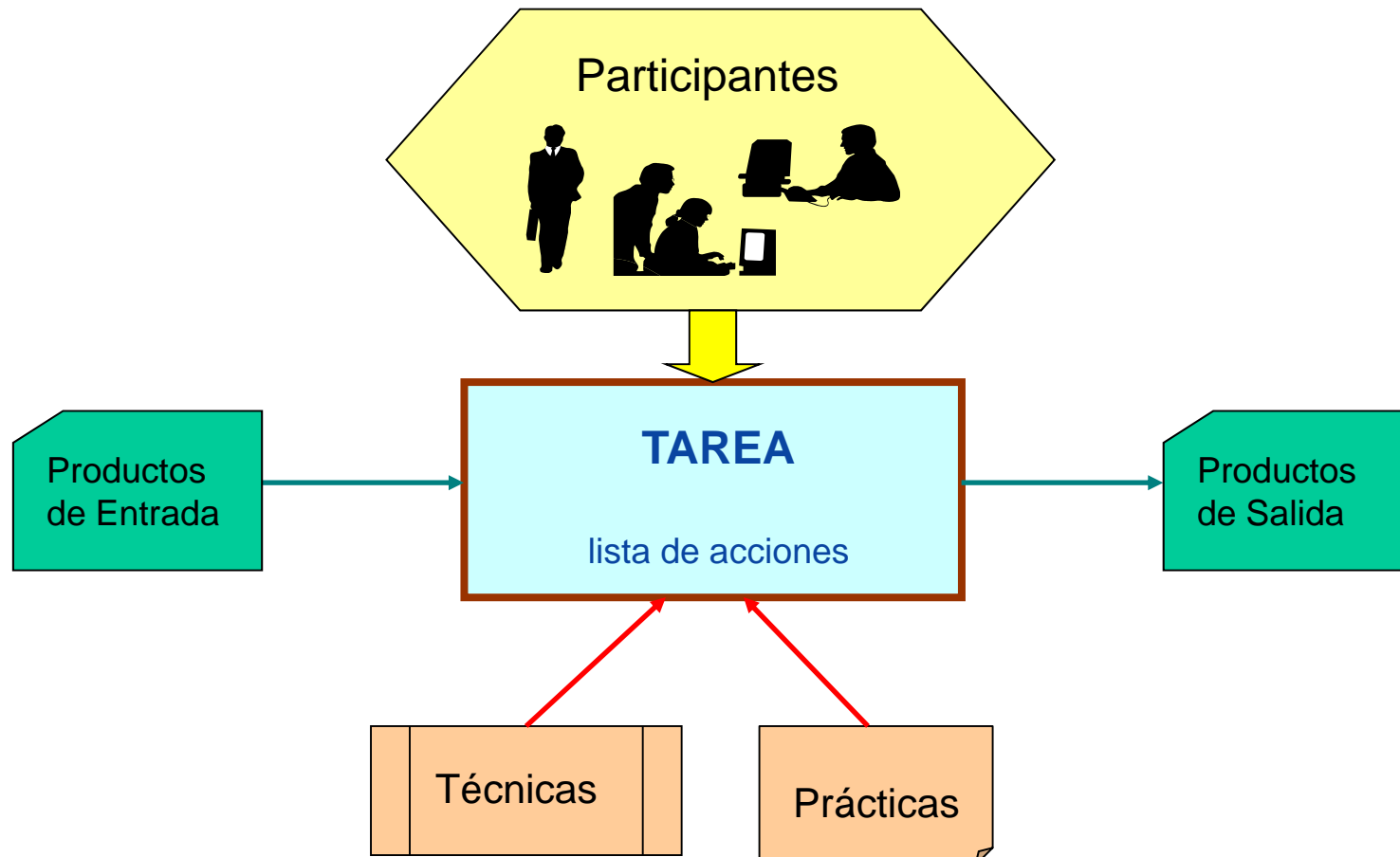
- **METRICA v.3.** Administración Pública España (2001).
 - Con una única estructura común cubre dos tipos de desarrollo:
 - Estructurado.
 - Orientado a objetos.
 - **Procesos:**
 1. Planificación de Sistemas de Información – (PSI)
 2. Desarrollo de Sistemas de Información:
 - a. Estudio de Viabilidad del Sistema (EVS)
 - b. Análisis del Sistema de Información (ASI)
 - c. Diseño del Sistema de Información (DSI)
 - d. Construcción del Sistema de Información (CSI)
 - e. Implantación y Aceptación del Sistema (IAS)
 3. Mantenimiento de Sistemas de Información (MSI)
 - A través de **interfaces** se facilita la realización de diversos procesos de soporte y organizativos: Gestión de Proyectos, Gestión de Configuración, Aseguramiento de Calidad, y Seguridad.



APÉNDICE B: Ejemplos de Metodologías

- **METRICA v.3.**

- **Patrón de Tarea:** participantes, productos de entrada y salida, técnicas y prácticas.





APÉNDICE B: Ejemplos de Metodologías

● MANTEMA

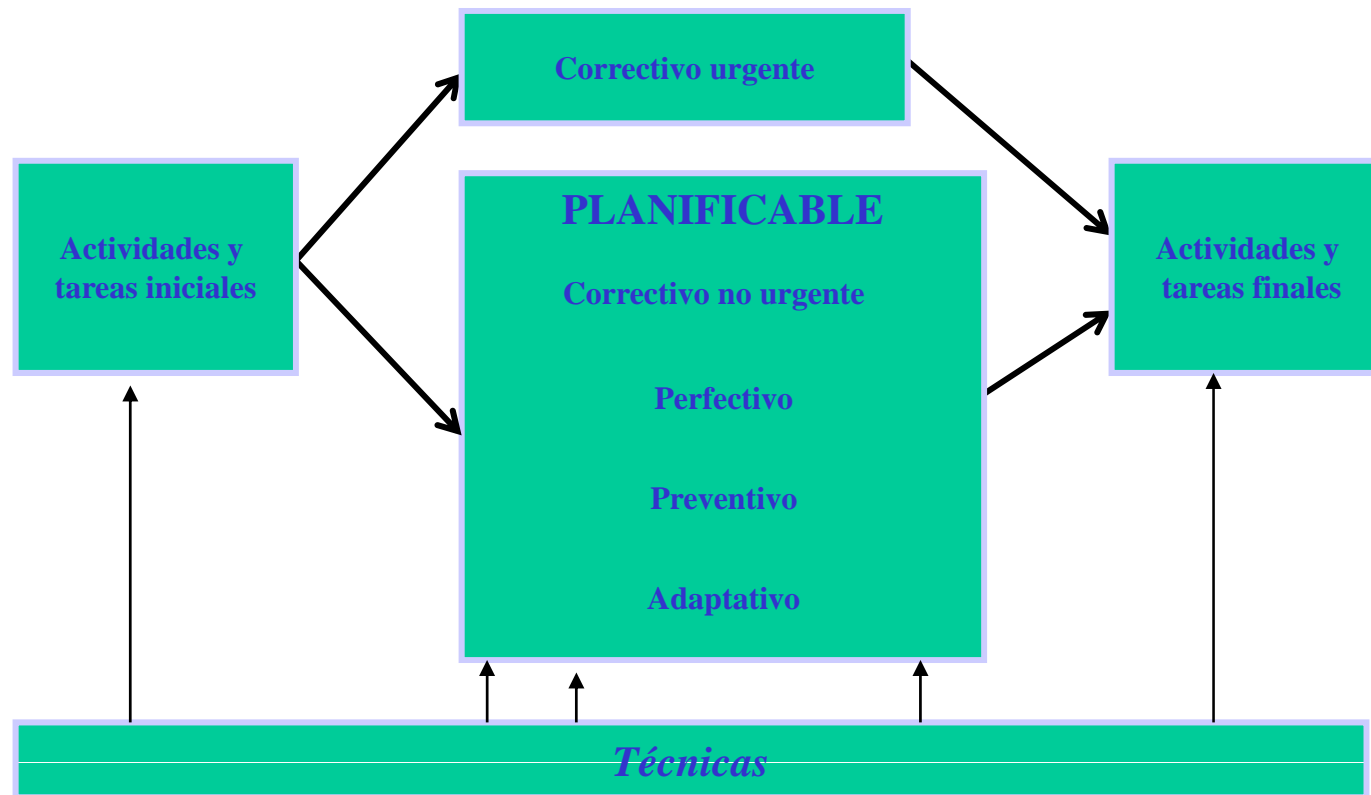
- Una de las pocas especializada en **Mantenimiento** en lugar de desarrollo. Desarrollada por la Universidad Castilla-La Mancha (1999).
- Contempla los siguientes **tipos de mantenimiento**:
 - **No Planificable (NP)**:
 - Correctivo Urgente (UC): localizar y eliminar los posibles defectos que bloquean el programa o los procesos de funcionamiento de la empresa.
 - **Planificable (P)**:
 - Correctivo No Urgente (NUC): localizar y eliminar los posibles defectos de los programas que no son bloqueantes.
 - Perfectivo (PER): añadir al software nuevas funcionalidades solicitadas por los usuarios.
 - Adaptativo (A): modificar el software para adaptarlo a cambios en el entorno de trabajo (hardware o software).
 - Preventivo (PRE): modificar el software para mejorar sus propiedades (calidad, mantenibilidad, etc.).



APÉNDICE B: Ejemplos de Metodologías

● MANTEMA

- Su modelo de proceso establece cuatro grupos de actividades y una colección de técnicas útiles.

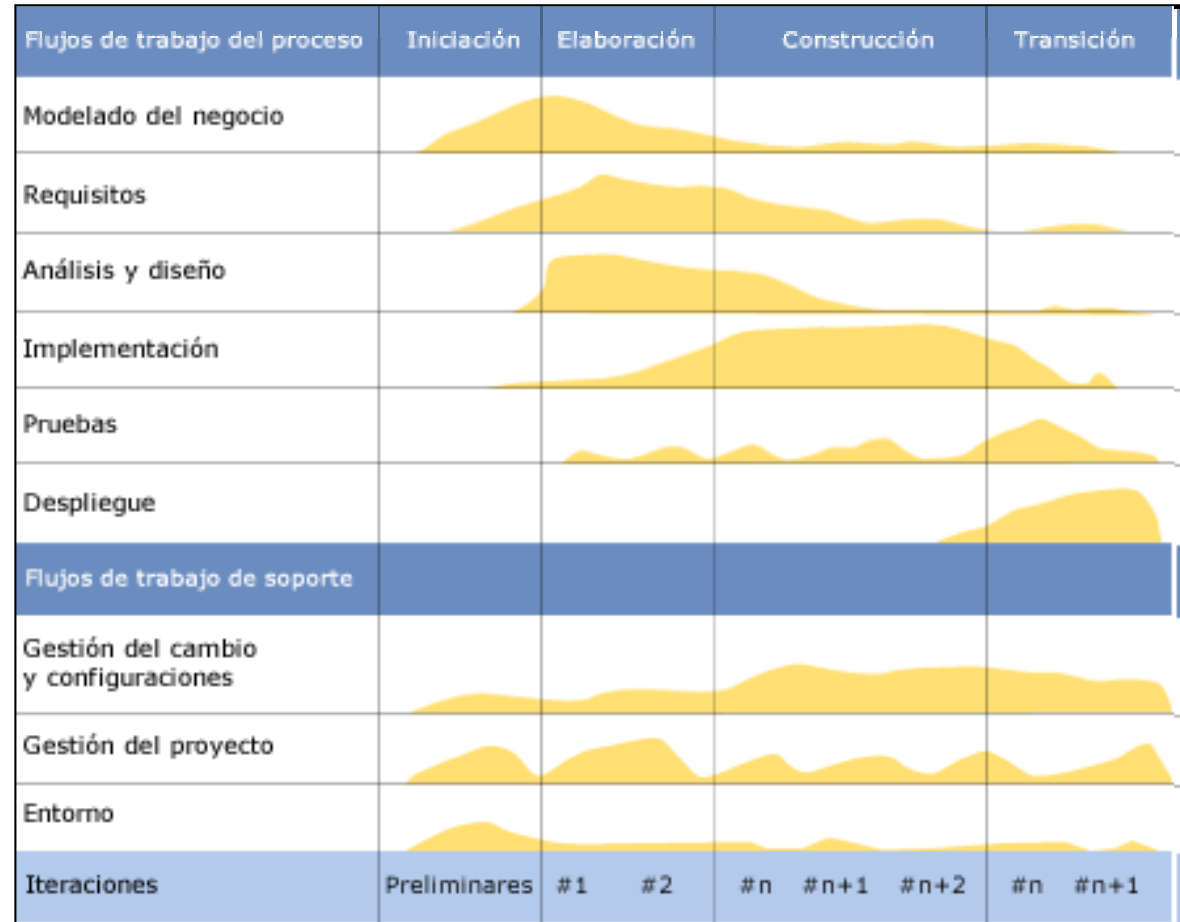




APÉNDICE B: Ejemplos de Metodologías

- **RUP** Rational Unified Process.

- Desarrollo OO iterativo e incremental usando UML.

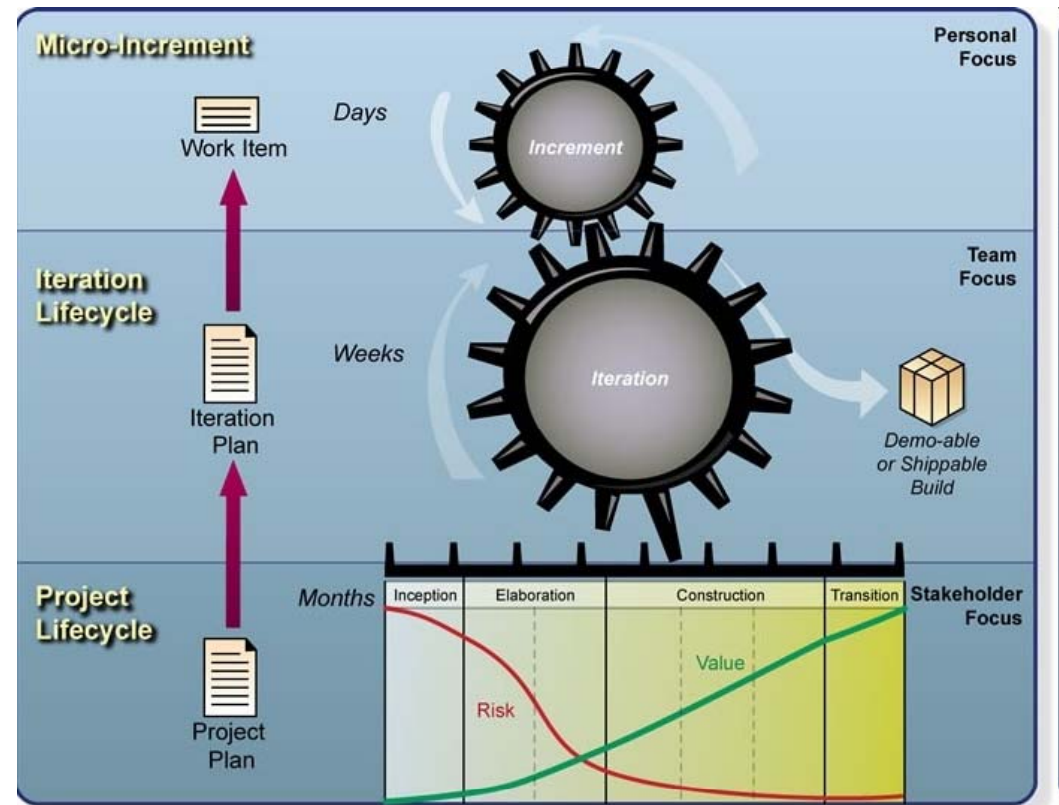




APÉNDICE B: Ejemplos de Metodologías

- **OpenUP.**

- <http://epf.eclipse.org/wikis/openup/>
- RUP abierto.
- Ciclo de desarrollo:
 - Iterativo.
 - Micro-Incrementos.
 - Gestión Ágil del Proyecto.





APÉNDICE B: Ejemplos de Metodologías

- **XP** eXTreme Programming.
 - <http://www.extremeprogramming.org>
 - Pone más énfasis en la adaptabilidad que en la previsibilidad: *“los cambios de requisitos son un aspecto natural e inevitable en un proyecto”*.
 - Se basa en la adopción de una serie de prácticas:
 - Desarrollo iterativo e incremental.
 - Pruebas unitarias frecuentes y automatizadas.
 - Peer programming (programación por parejas).
 - Cliente in-situ.
 - Refactorización del código (reescribir).
 - Responsabilidad compartida sobre el código.