INGENIERÍA DEL SOFTWARE

UNIDAD 1: INGENIERÍA DEL SOFTWARE
CICLO LECTIVO 2013

OBJETIVOS DE LA CLASE

- Productos de Software.
- Características del Software.
- Tipos de Software.
- Proceso de Desarrollo.
- Ciclo de Vida.
- IS: una solución?

PRODUCTOS DE SOFTWARE

- Productos genéricos.
  - Productos que son producidos por una organización para ser vendidos al mercado.
- Productos hechos a medida.
  - Sistemas que son desarrollados bajo pedido a clientes específicos.
- La mayor parte del gasto del software es en productos genéricos, pero hay más esfuerzo en el desarrollo de los sistemas hechos a medida.

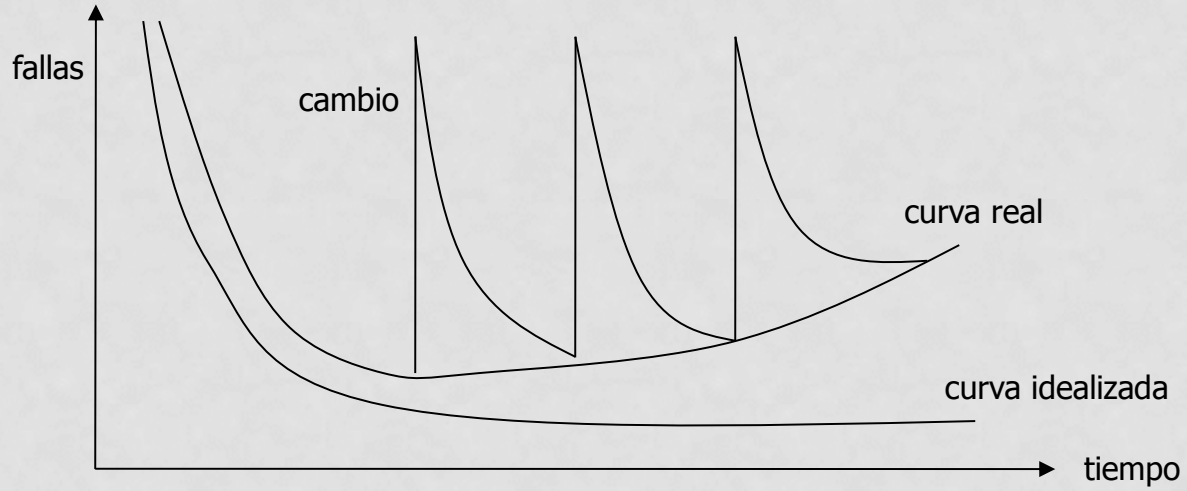
NOMBRAR EJEMPLOS

- **Productos genéricos???.**
- **Productos a medida???**
- **¿En otras ramas de las ingenierías?**

CARACTERÍSTICAS DEL SOFTWARE

- El software se desarrolla, no se fabrica/manufactura.
  - Los costos del software se concentran en la ingeniería.
- El software no se desgasta, pero sí se "deteriora".
- La mayoría del software se construye a medida.
  - La reutilización de componentes recién ha empezado.

CARACTERÍSTICAS DEL SOFTWARE



¿Qué significado le damos a este gráfico?
¿Cuáles son las razones?

CARACTERÍSTICAS DE LOS PRODUCTOS DE SOFTWARE

- Mantenible.
  - Debe ser posible que el software evolucione y que siga cumpliendo con sus especificaciones.
- Confiable.
  - Que el software funcione libre de fallas en un entorno determinado y durante un tiempo específico.
- Eficiente.
  - El software no debe desperdiciar los recursos del sistema. Debe usarlos adecuadamente (memoria, tiempos de respuesta, etc. ...).
- Interfaz adecuada.
  - El software debe contar con una interfaz de usuario adecuada y su documentación.

TIPOS DE SOFTWARE (PRESSMAN 2005)

- Por función:
  - Software de aplicación.
  - Software científico y de ingeniería.
  - Software empotrado.
  - Software de línea de productos.
  - Software de inteligencia artificial.
  - Juegos.

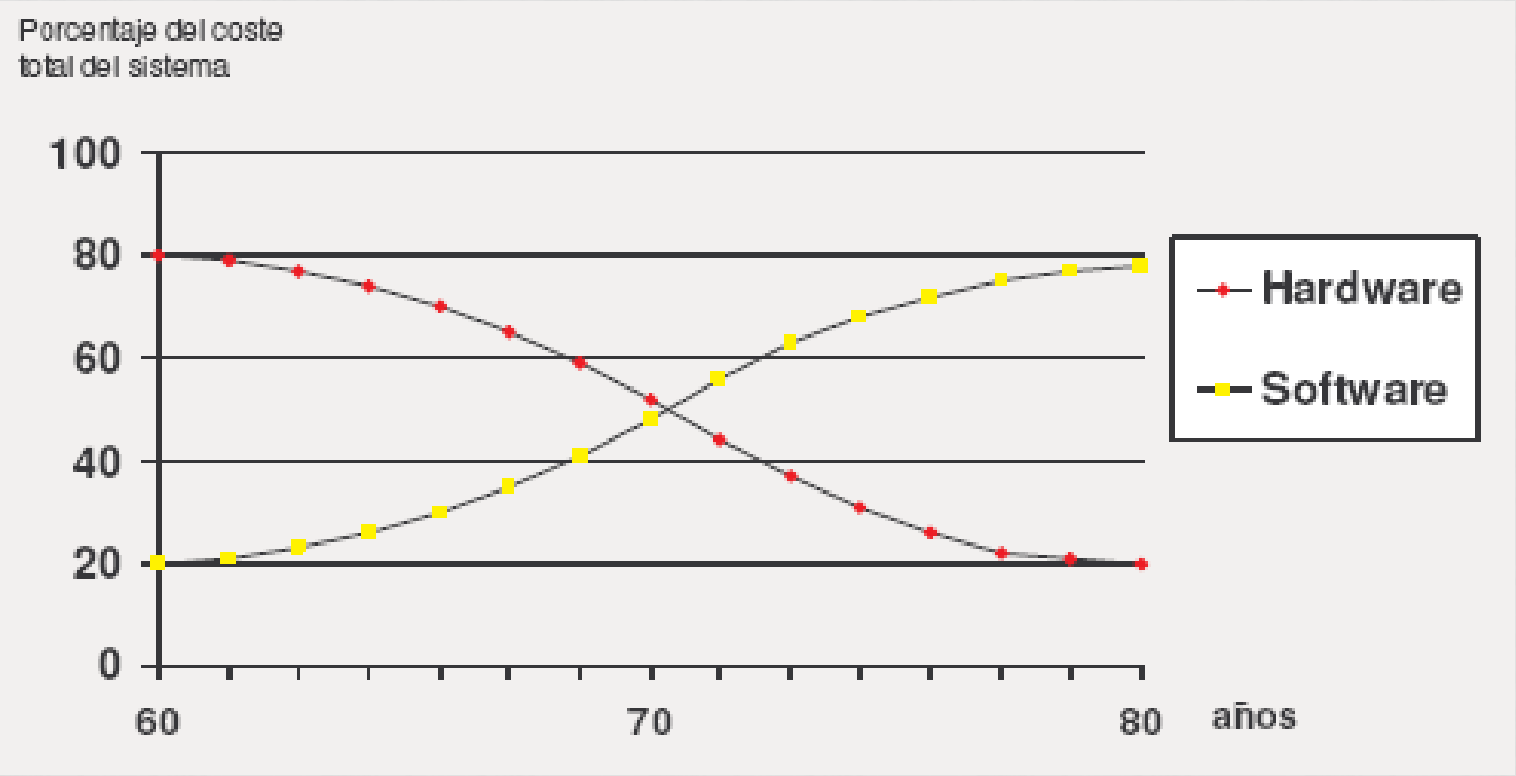
TIPOS DE SOFTWARE (PRESSMAN 2005)

- Por origen:
  - Software nuevo.
  - Software heredado.
  - Software de integración.

TIPOS DE SOFTWARE

- Por arquitectura
  - Monolítico.
  - Cliente/servidor.
  - Aplicaciones basadas en web.
  - Etc...

COSTOS HARD Y SOFT



NECESIDAD DE UN PROCESO

Un proceso de Desarrollo de SW que:

- Organiza y estructura las actividades.
- Contribuye a la calidad del software y a la velocidad con que se desarrolla.
- Define el enfoque que se adopta mientras el software está en desarrollo.
- Es parte de la Ingeniería de Software.

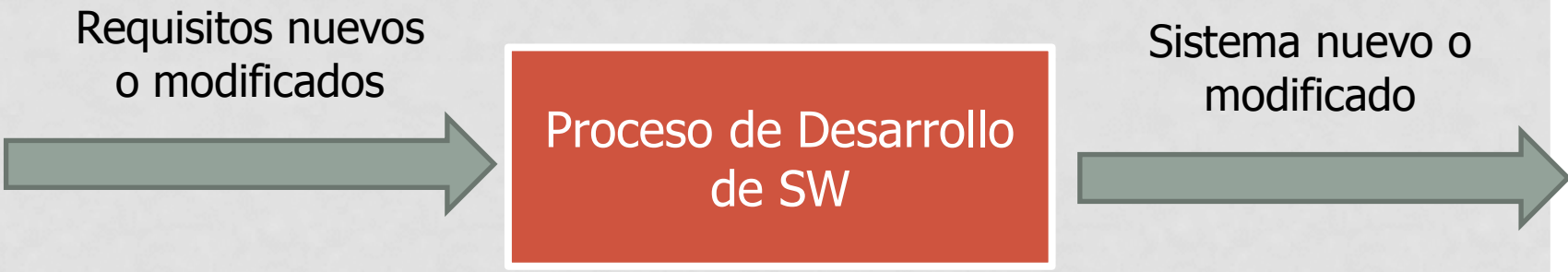
PROCESO DE DESARROLLO DE SW

- Define un marco de trabajo para realizar SW de alta calidad.
- Define un conjunto de actividades y resultados asociados que producen un "producto de SW".
- Nos brinda: estabilidad, organización y control.
- Define las interacciones:
  - Usuarios – Diseñadores.
  - Usuarios – Herramientas.
  - Diseñadores - Herramientas.

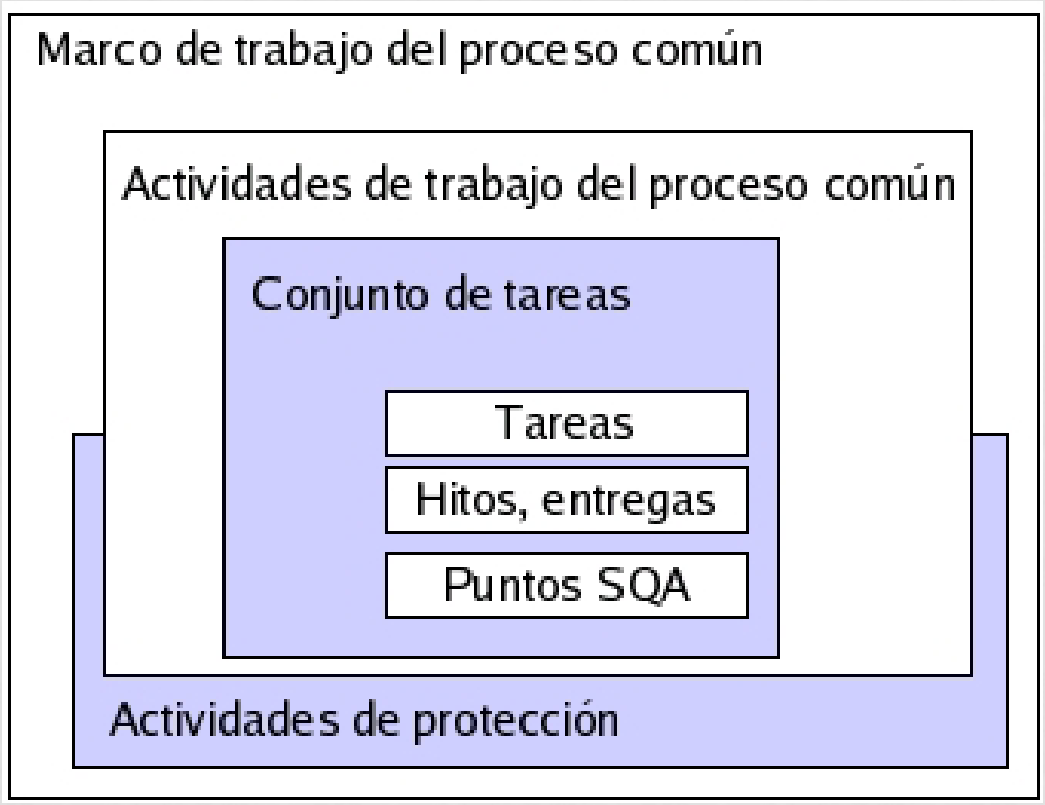
¿POR QUÉ SEGUIR UN PROCESO?

- Un proceso es un conjunto de procedimientos, organizado para construir productos que satisfacen una serie de objetivos y estándares.
- Los procesos son importantes porque imponen **consistencia** y **estructura** en un conjunto de actividades.
- Sabemos cómo hacer algo bien y queremos forzar que otros lo hagan de la misma forma.

PROCESO DE DESARROLLO DE SW



ELEMENTOS DEL PROCESO



PROCESO DE SW

- Distintos procesos de software organizan las actividades de diferentes formas, y las describen con diferente nivel de detalle.
- El tiempo de cada actividad varía, así como los resultados.
- Organizaciones diferentes usan procesos diferentes para producir el mismo producto.
- Para algunos tipos de aplicación, algunos procesos son más convenientes que otros.

CARACTERÍSTICAS DEL PROCESO DE DESARROLLO

- Entendible
  - ¿Se encuentra el proceso bien definido y es entendible?.
- Visible
  - ¿El proceso es visible al exterior?.
- Soportable
  - ¿Puede el proceso ser soportado por herramientas CASE? ¿Puede ser implementado según las condiciones existentes?
- Aceptable
  - ¿El proceso es aceptado por los involucrados en él?
- Confiable
  - ¿Los errores del proceso son descubiertos antes de que se conviertan en errores del producto?.
- Mantenible
  - ¿Puede el proceso evolucionar para cumplir con los objetivos organizacionales?.
- Rapidez
  - ¿Cuánto más rápido puede producirse el producto (SFW)?

MODELOS DEL PROCESO – CICLO DE VIDA

- Se debe escoger una estrategia de desarrollo, llamada:
  - modelo de proceso, o
  - ciclo de vida.
- Se selecciona de acuerdo a:
  - naturaleza del proyecto y aplicación,
  - controles y entregas requeridas,
  - características del equipo u organización de desarrollo, entre otros.

Ninguno es mejor que otro, cada proyecto tiene el más apropiado

CICLO DE VIDA

Sucesión de etapas por las que atraviesa un producto de software a lo largo de su existencia.

"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 12207-1

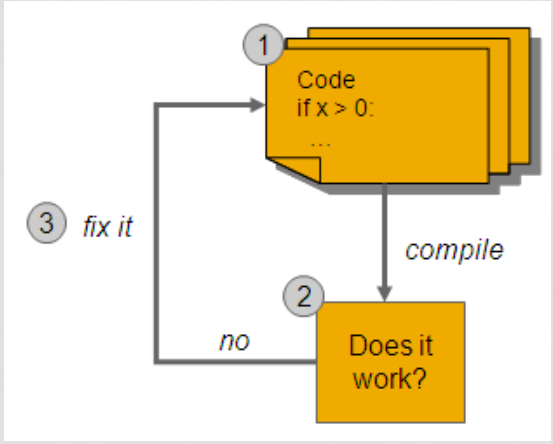
MODELOS PRESCRIPTIVOS DE PROCESOS

Las principales diferencias entre distintos modelos de ciclo de vida están divididas en tres grandes visiones:

- El alcance del ciclo de vida (hasta dónde se desea llegar con el proyecto): sólo saber si es viable el desarrollo de un producto, el desarrollo completo o el desarrollo completo más actualizaciones y mantenimiento.
- La cualidad y cantidad de las etapas en que será dividido el ciclo de vida: según el que sea adoptado y el proyecto para el cual sea adoptado.
- La estructura y la sucesión de las etapas, si hay realimentación entre ellas y si hay libertad de repetirlas (iteración).

CODIFICAR Y CORREGIR (CODE-AND-FIX)

- Este es el modelo básico utilizado en los inicios del desarrollo de software. Contiene dos pasos:
  - Escribir código.
  - Corregir problemas en el código.
- Se trata de primero implementar algo de código y luego pensar acerca de requisitos, diseño, validación, y mantenimiento.

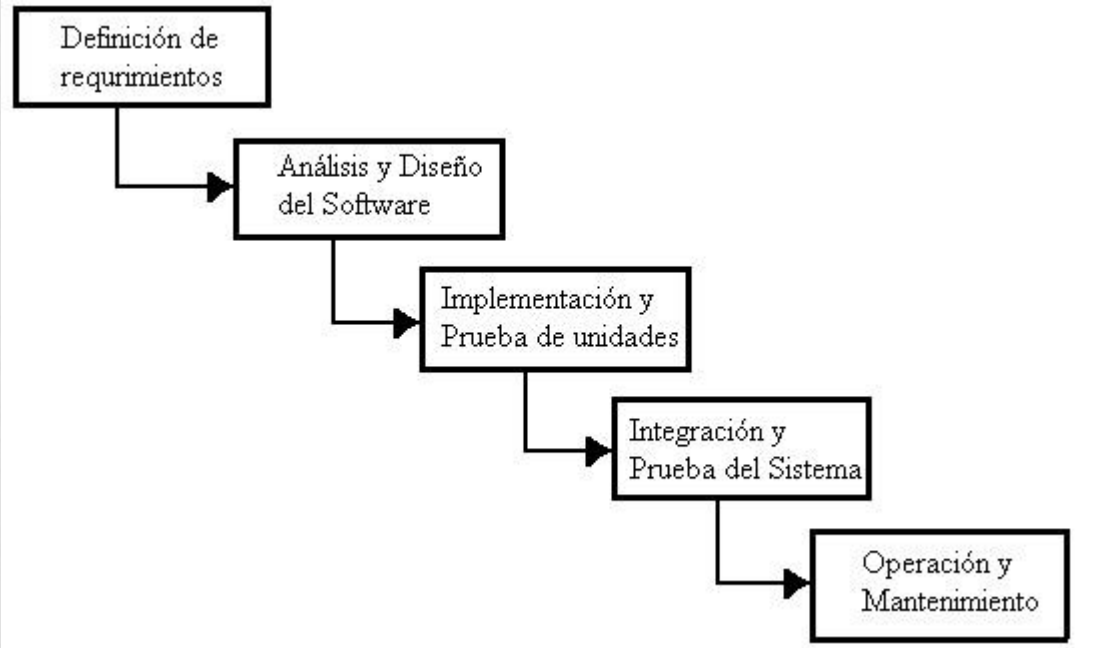


PROBLEMAS DE CODE-AND-FIX

- ¿Qué problemas tiene este modelo? ¿Cuándo usar?… Pensemos…
  - Después de un número de correcciones, el código puede tener una muy mala estructura, lo que hace que los arreglos sean muy costosos.
  - Frecuentemente, aún el software bien diseñado, no se ajusta a las necesidades del usuario, por lo que es rechazado o su reconstrucción es muy cara.
  - El código es difícil de reparar por su pobre preparación para probar y modificar.

CASCADA

- El primer modelo de desarrollo de software que se publicó se derivó de otros procesos de ingeniería. Éste toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y las representa como fases separadas del proceso.



CASCADA

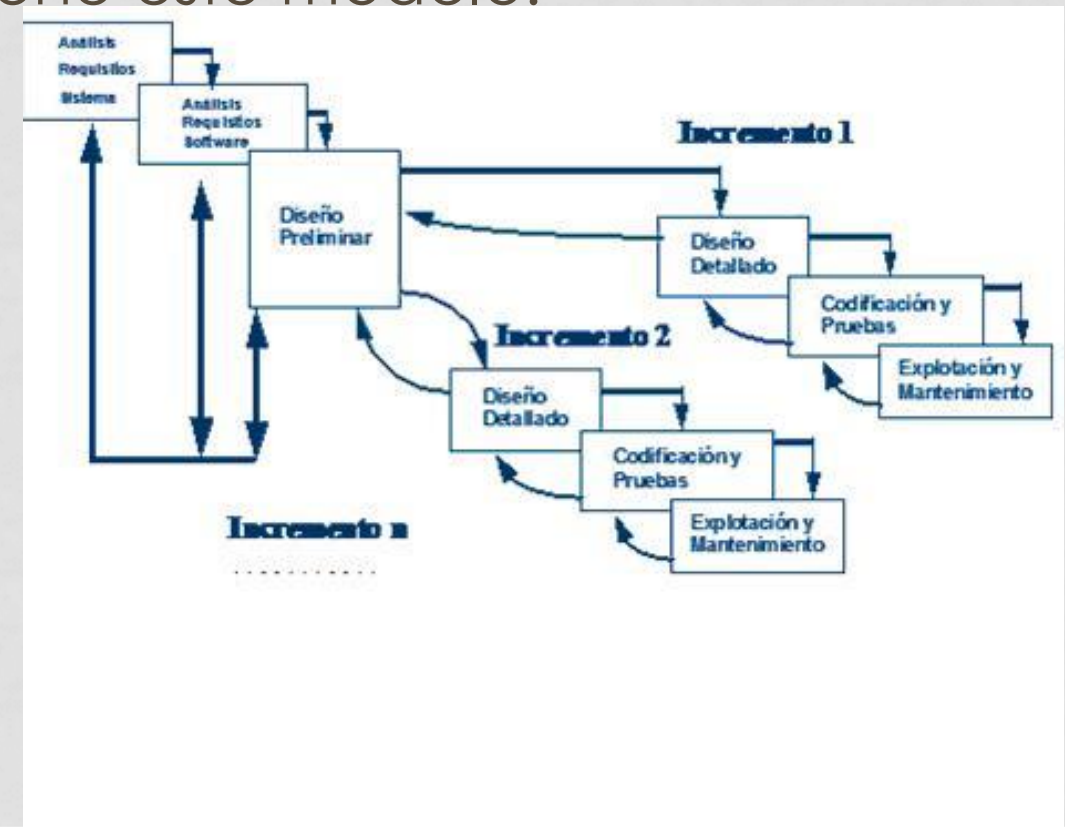
- Tiene las siguientes fases:
  - Definición de los requisitos: Los servicios, restricciones y objetivos son establecidos con los usuarios del sistema.
  - Diseño de software: Se particiona el sistema en sistemas de software o hardware. Se establece la arquitectura total del sistema. Se identifican y describen las abstracciones y relaciones de los componentes del sistema.
  - Implementación y pruebas unitarias: Construcción de los módulos y unidades de software. Se realizan pruebas de cada unidad.
  - Integración y pruebas del sistema: Se integran todas las unidades. Se prueban en conjunto. Se entrega el conjunto probado al cliente.
  - Operación y mantenimiento: Generalmente es la fase más larga. El sistema es puesto en marcha y se realiza la corrección de errores descubiertos. Se realizan mejoras de implementación. Se identifican nuevos requisitos.

CASCADA: VENTAJAS Y DESVENTAJAS

- Pensemos ¿Ventajas? ¿Desventajas? ¿Cuándo usar?...
  - Cada fase genera entradas y documentación a la siguiente.
  - Se considera el modelo "clásico" del desarrollo de software: mas antiguo y mas usado.
  - Enfoque sistemático secuencial.
- Problemas:
- Los proyectos reales raramente pueden seguir el flujo secuencial que se propone.
- Problemas con requerimientos o diseño -> Se resuelven luego de la implementación (con costos inmensamente superiores a su resolución en etapas tempranas).
- Alta dependencia entre fases ->Se tarda mucho tiempo en pasar por todo el ciclo.
- Producto operativo al final.

CASCADA INCREMENTAL

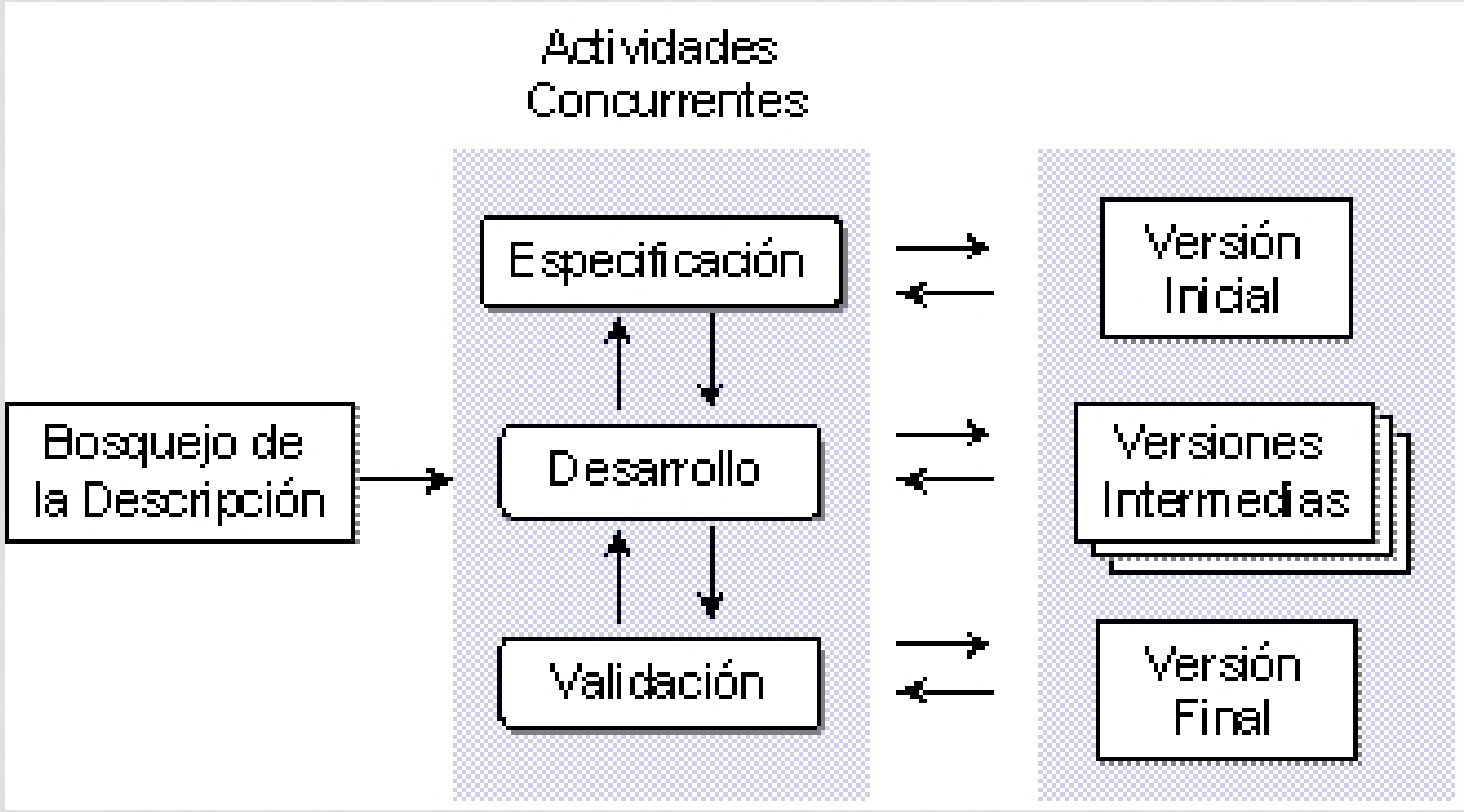
Como una mejora a los problemas planteados por el modelo anterior, y para proyectos mas importantes se propone este modelo:



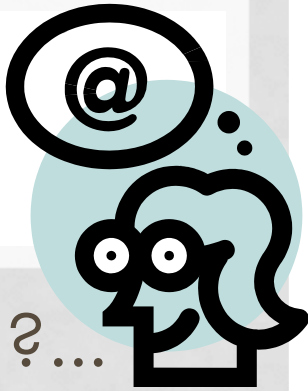
MODELOS INCREMENTALES

- Incrementos: proporcionan un subconjunto de funcionalidad : se entrega "*algo de valor*" al cliente con cierta frecuencia.
- El cliente se involucra más, se usan los incrementos como prototipos y generan experiencias de validación.
- 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.

MODELOS INCREMENTALES



MODELOS INCREMENTALES



¿Cuándo usaríamos un modelo incremental?...

- Cuando no se dispone del personal para una instalación completa.
- Cuando se pueden definir incrementos acotados en tamaño, con funcionalidades bien definidas.
- Cuando no se está seguro de cumplir con plazos de tiempo o se tiene una fecha imposible de cambiar.
- **No es** recomendable para atributos de calidad críticos.

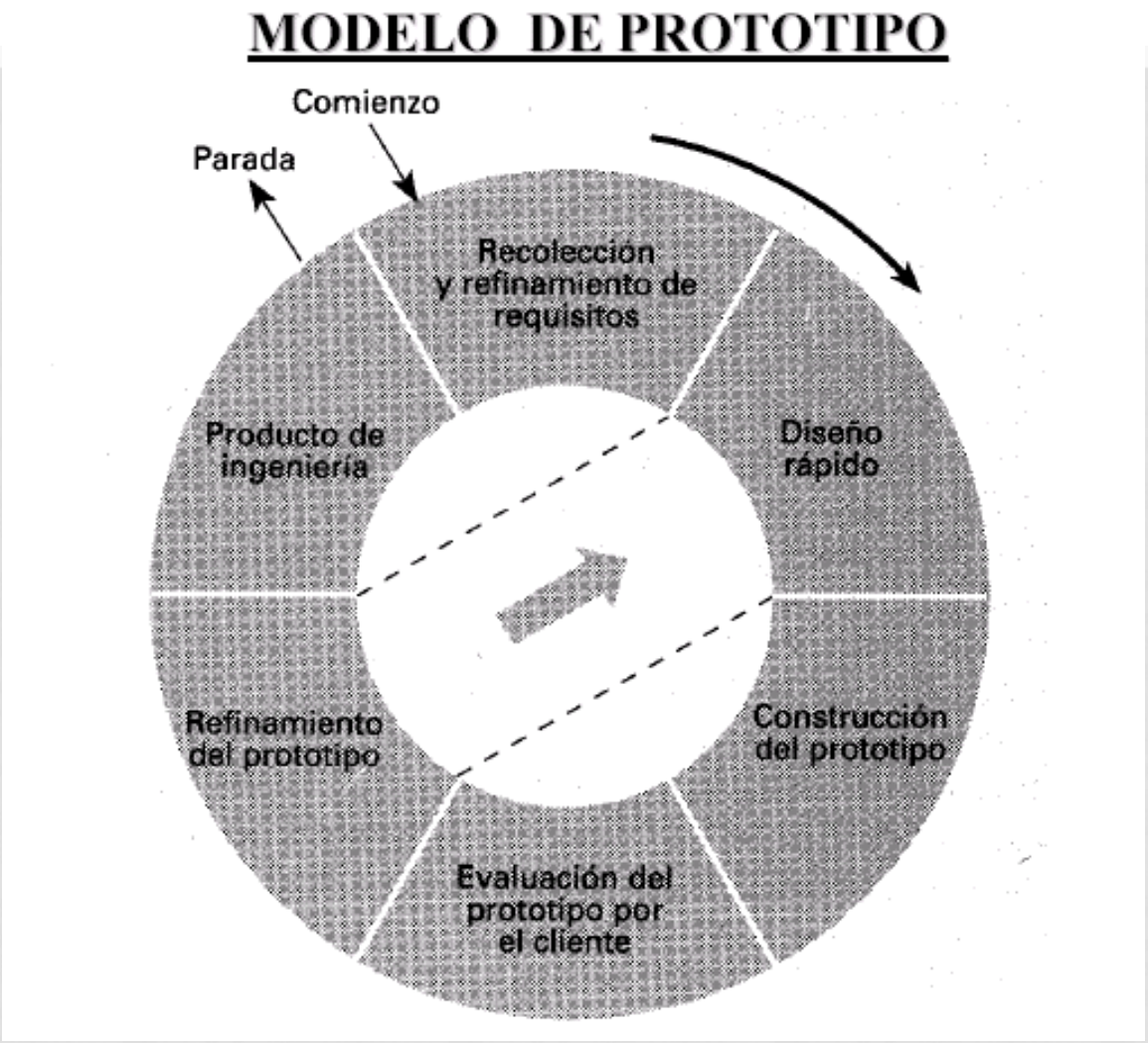
MODELOS EVOLUTIVOS

- La idea detrás de este modelo es el desarrollo de una implantación del sistema inicial, exponerla a los comentarios del usuario, refinarla en N versiones hasta que se desarrolle el sistema adecuado.
- Se adaptan más fácilmente a los cambios introducidos a lo largo del desarrollo.
- Iterativos
- En cada iteración se obtienen versiones más completas del software.

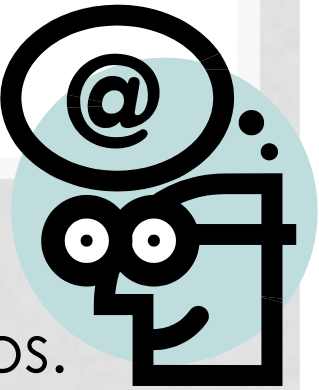
CONSTRUCCIÓN DE PROTOTIPOS

- No están claros los requerimientos al inicio.
  - Sistemas nuevos, poco conocimiento del dominio
    - Útil cuando hay inseguridades del lado cliente y/o desarrollador.
- Reduce el riesgo.
- Especificación: desarrollo creciente.
- No modifica el flujo del ciclo de vida.
- Una vez identificados los requerimientos, se construye el producto.

CONSTRUCCIÓN DE PROTOTIPOS



CONSTRUCCIÓN DE PROTOTIPOS



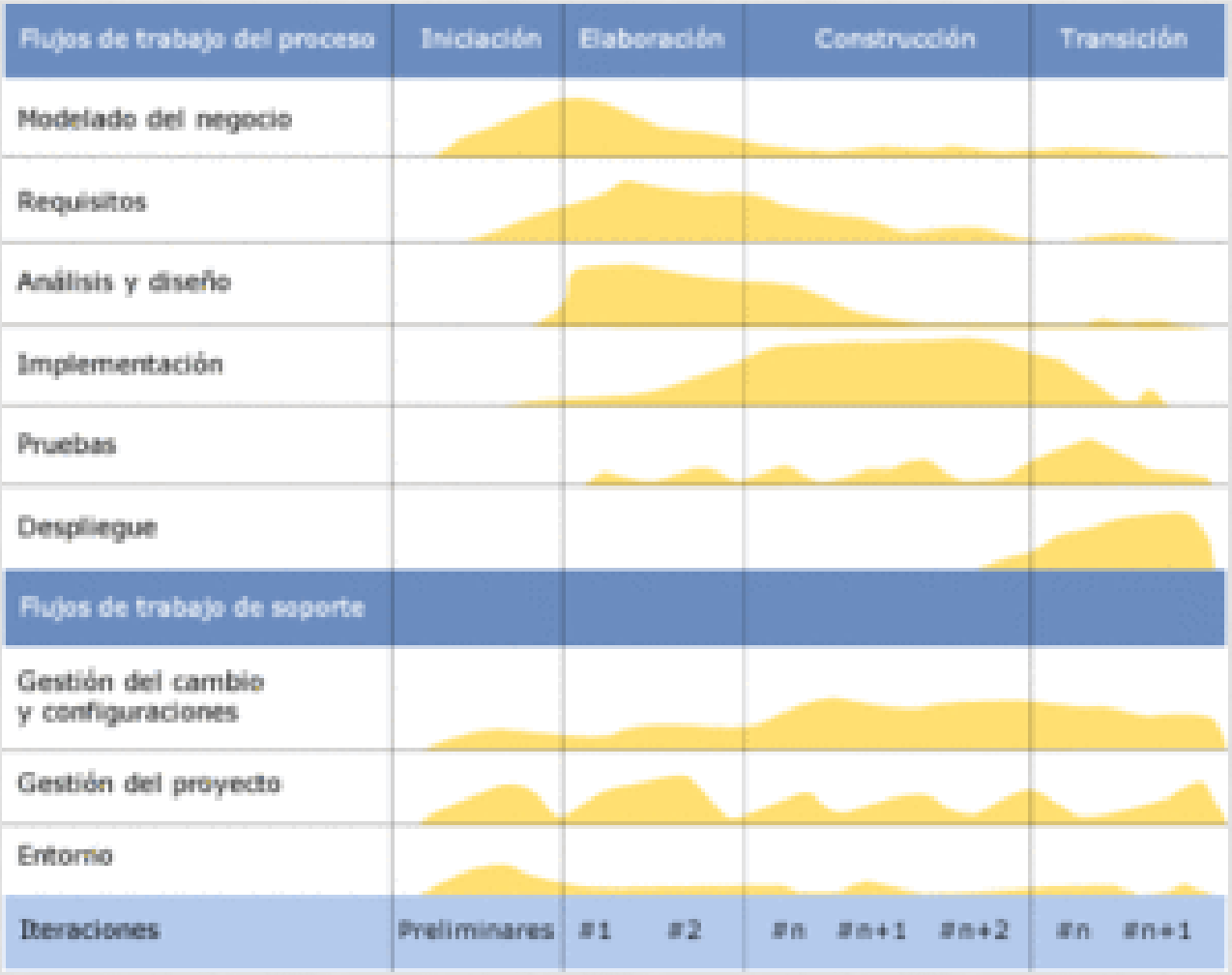
- Pensemos las ventajas y desventajas...
  - Se usa cuando no están claros los requisitos.
  - El cliente puede pensar que es el sistema.
  - ¿Cuándo usar? Debe estar claro esto, cuando es conveniente su uso.

"El prototipado es un medio excelente para recoger el 'feedback' (realimentación) del usuario final"

RUP

- El Rational Unified Process (RUP) (Krutchen, 2003) es un ejemplo de un moderno modelo de proceso derivado del trabajo sobre UML.
- Es un buen ejemplo de un modelo de proceso híbrido:
  - ilustra las buenas prácticas en la especificación y el diseño
  - es compatible con prototipos
  - Y permite la entrega incremental
- RUP se describe normalmente a partir de tres perspectivas:
  - Una perspectiva dinámica, que muestra las fases del modelo a través del tiempo.
  - Un punto de vista estático, que muestra las actividades de los procesos que están vigentes.
  - Una perspectiva de la práctica que sugiere las buenas prácticas que se utilizarán durante el proceso.

PROCESO UNIFICADO DE DESARROLLO



PROCESO UNIFICADO DE DESARROLLO

- Es un modelo por etapas que identifica cuatro fases diferenciadas en el proceso de software.
- Estas fases son:
  - Inicio: visión aproximada, análisis del quehacer de la empresa cliente ("el negocio"), alcance del proyecto, estimaciones (imprecisas) de plazos y costos. Se define la viabilidad del proyecto.
  - Elaboración: visión refinada, implementación iterativa del núcleo central de la aplicación, resolución de los riesgos más altos, identificación de nuevos requisitos y nuevos alcances, estimaciones más ajustadas.
  - Construcción: implementación iterativa del resto de los requisitos de menor riesgo y elementos más sencillos, preparación para el despliegue (entrega, instalación y configuración).
  - Transición: pruebas beta, despliegue.

RUP

- RUP describe las buenas prácticas en el desarrollo de sistemas informáticos.
  - Desarrollar software de forma iterativa: incrementos sobre la base de establecer prioridades y desarrollar las características del sistema de más alta prioridad al principio del proceso de desarrollo.
  - Gestionar los requerimientos: documentar los requisitos del cliente y hacer un seguimiento de los cambios en estos requisitos. Analizar el impacto de los cambios en la el sistema antes de aceptarlos.
  - Uso de componentes basados en la arquitectura.
  - Utilización de modelos UML para representar las vistas estáticas y dinámicas del sistema.
  - Verificar la calidad del software. El software debe cumplir con las normas de calidad de la organización.
  - Controlar los cambios mediante un software de gestión de los cambios y herramientas de administración y configuración.

INGENIERÍA DEL SOFTWARE

… según la IEEE [1990]…

"… La ingeniería de software es la aplicación de un enfoque sistemático, disciplinado, cuantificable para el desarrollo, operación y mantenimiento de software…"

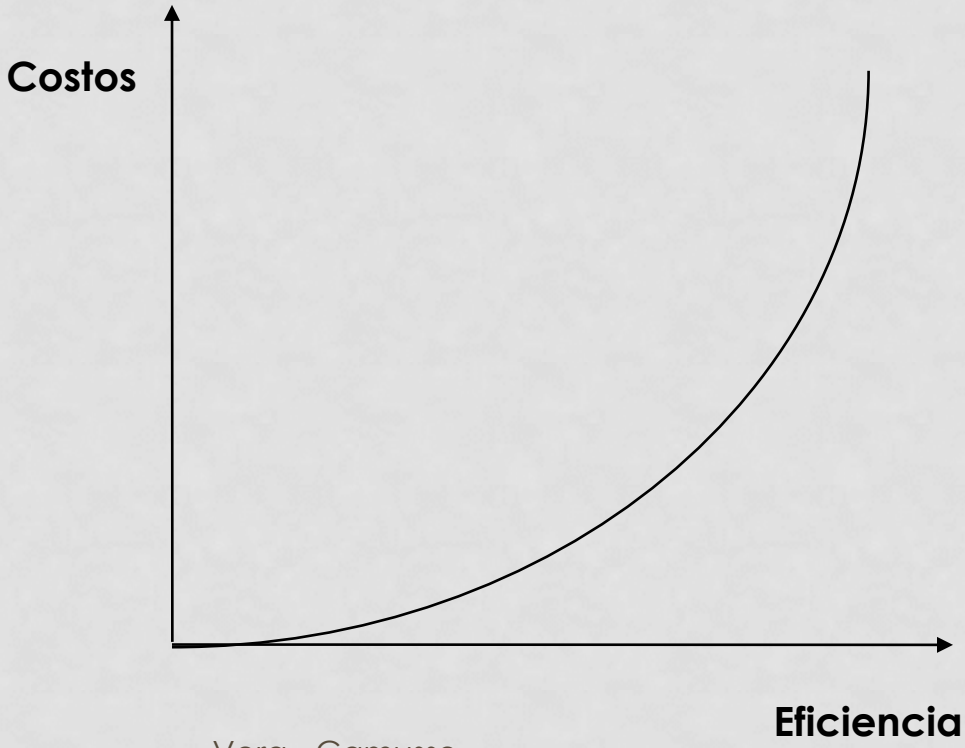
¿POR QUÉ ES DIFÍCIL?

- El **software** es uno de los objetos de mayor complejidad hecho por humanos
  - Desarrollar software es resolver un juego de restricciones de naturaleza técnica, económica y humana.
- Una disciplina joven víctima de su propio éxito
  - La teoría sobre la que se debe apoyar la ingeniería no está terminada.
  - La tecnología y la capacidad de construir sistemas complejos crece rápidamente exigiendo más a una disciplina ingenieril que está madurando e introduciendo nuevas problemáticas.

COSTOS DEL SOFTWARE

- Los costos del software a menudo dominan al costo del sistema. El costo del software en un PC es a menudo más caro que la PC.
- Cuesta más mantener el software que desarrollarlo. Para sistemas con una larga vida, este costo se multiplica.
- La Ingeniería de Software concierne a un desarrollo efectivo en cuanto a costos del software.

COSTO DE LA EFICIENCIA



SITUACIÓN ACTUAL

- Los cambios en hardware han sido enormes.
- Los cambios en software también:
  - Internet y aplicaciones relacionadas.
  - Enorme variedad de tecnologías para construir aplicaciones, que pueden ser desplegadas mucho más rápidamente que en el pasado.
- **Sin embargo, más allá de las tecnologías, si miramos los procesos de ingeniería del software ➔ muchas cosas permanecen igual.**

SITUACIÓN ACTUAL

- El modelo en cascada sigue siendo utilizado por más del 40% de las empresas a pesar de que sus serios problemas fueron identificados hace 20 años.
- Todavía muchos proyectos terminan tarde, exceden el presupuesto o no entregan el software que esperaban los clientes.
- En muchas áreas sigue sin existir un conjunto de estándares que se use ampliamente:
  - La disciplina no es todavía madura.
  - Es necesario mayor esfuerzo en educación en ISW.

SITUACIÓN ACTUAL

- Tres problemas esenciales al comienzo del siglo XXI [Sommerville]:
  - *El reto de lo heredado*: mantener y actualizar software antiguo con funciones críticas, sin parar el negocio, evitando costos excesivos.
  - *El reto de la heterogeneidad*: desarrollar sistemas flexibles, multiplataforma, ...
  - *El reto de la entrega*: reducir tiempo de entrega sin reducir la calidad del sistema.

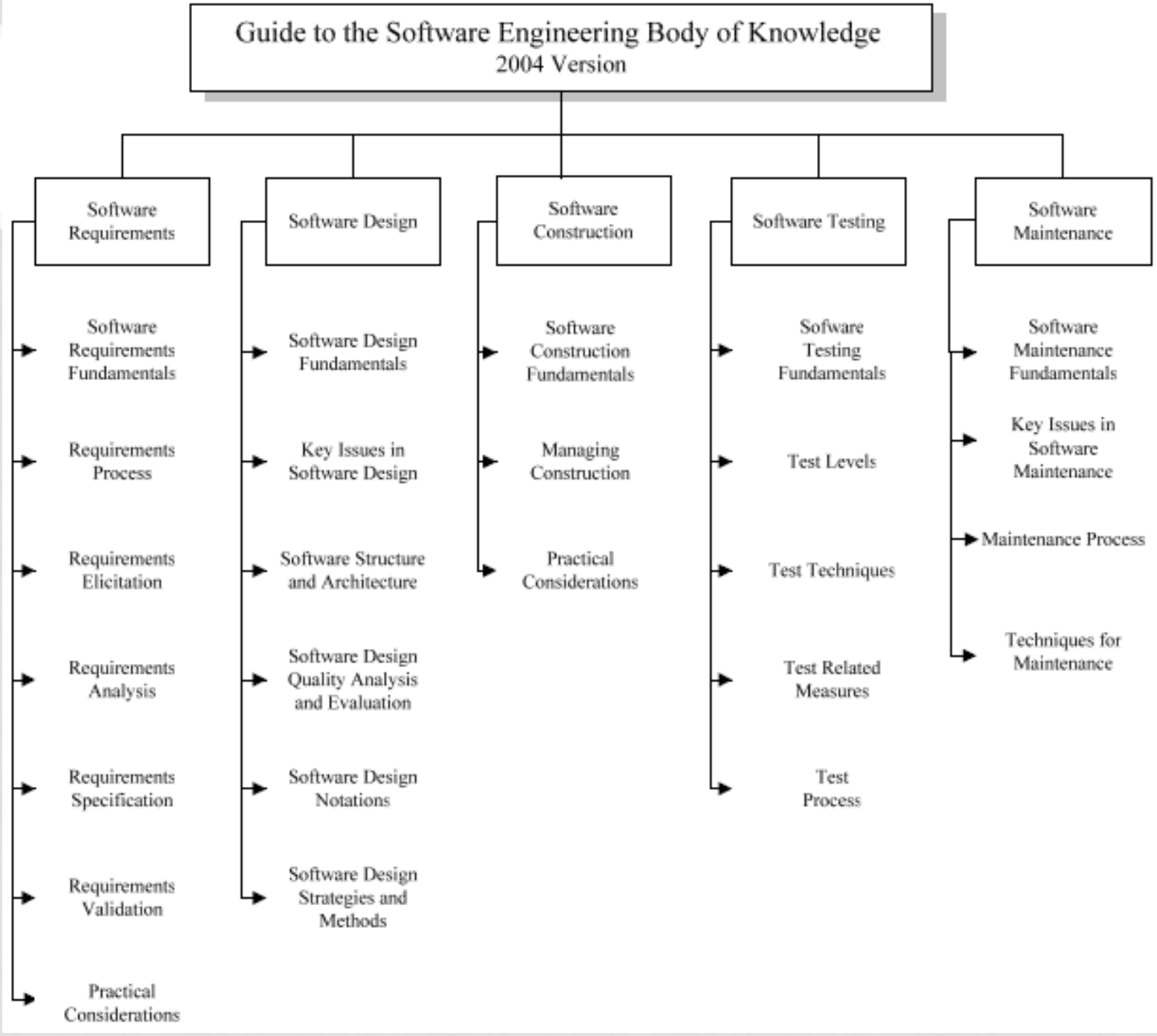
DEMANDAS COMERCIALES

- Sistemas que nunca deben fallar y que siempre deben alcanzar sus plazos de entrega. (Ejemplo: sistemas de control de aviones).

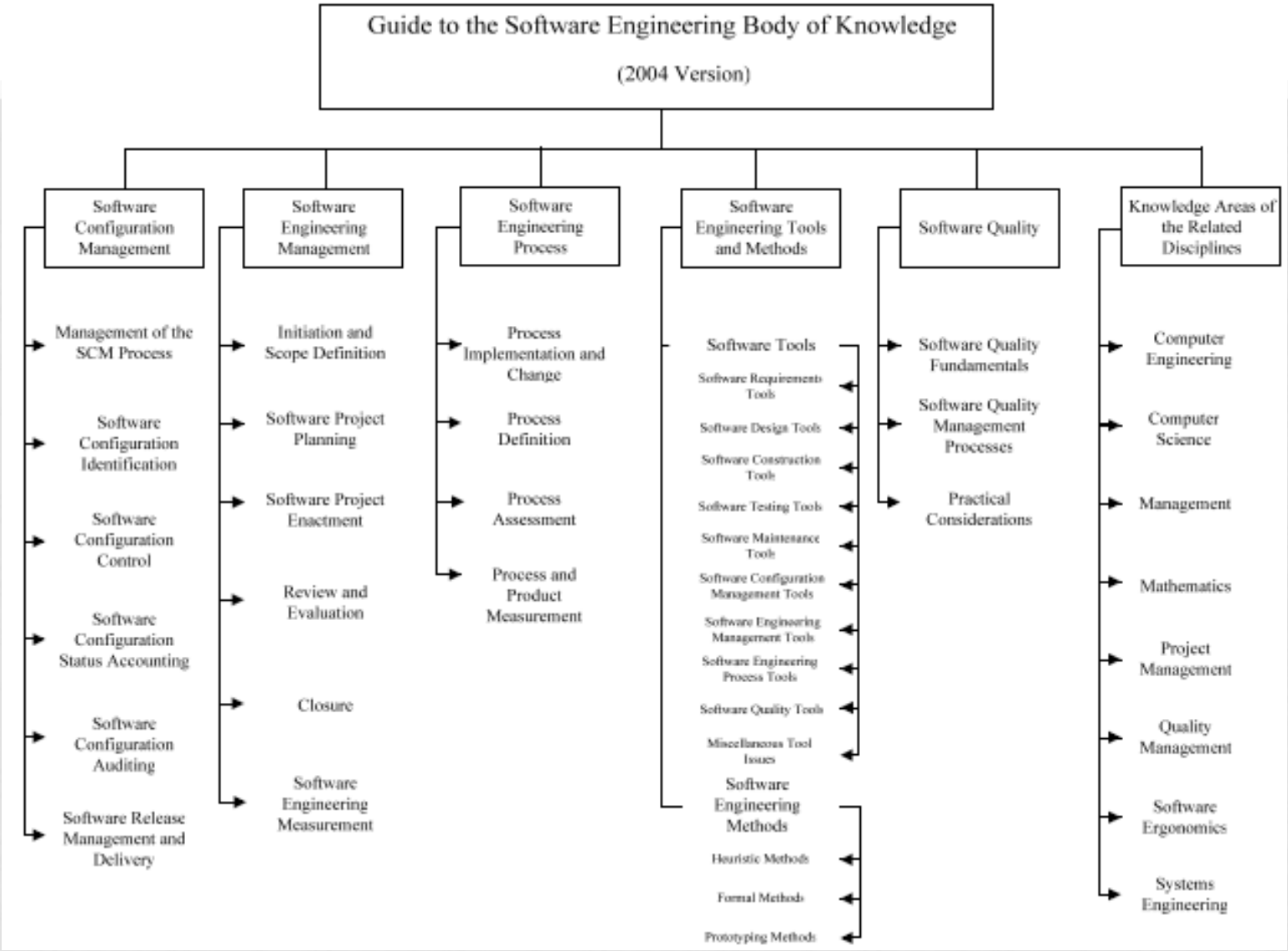
- Sistemas que deben ser seguros, confiables, livianos y extensibles. (Ejemplo: Sistemas de Tarjetas de Crédito o compras on-line).

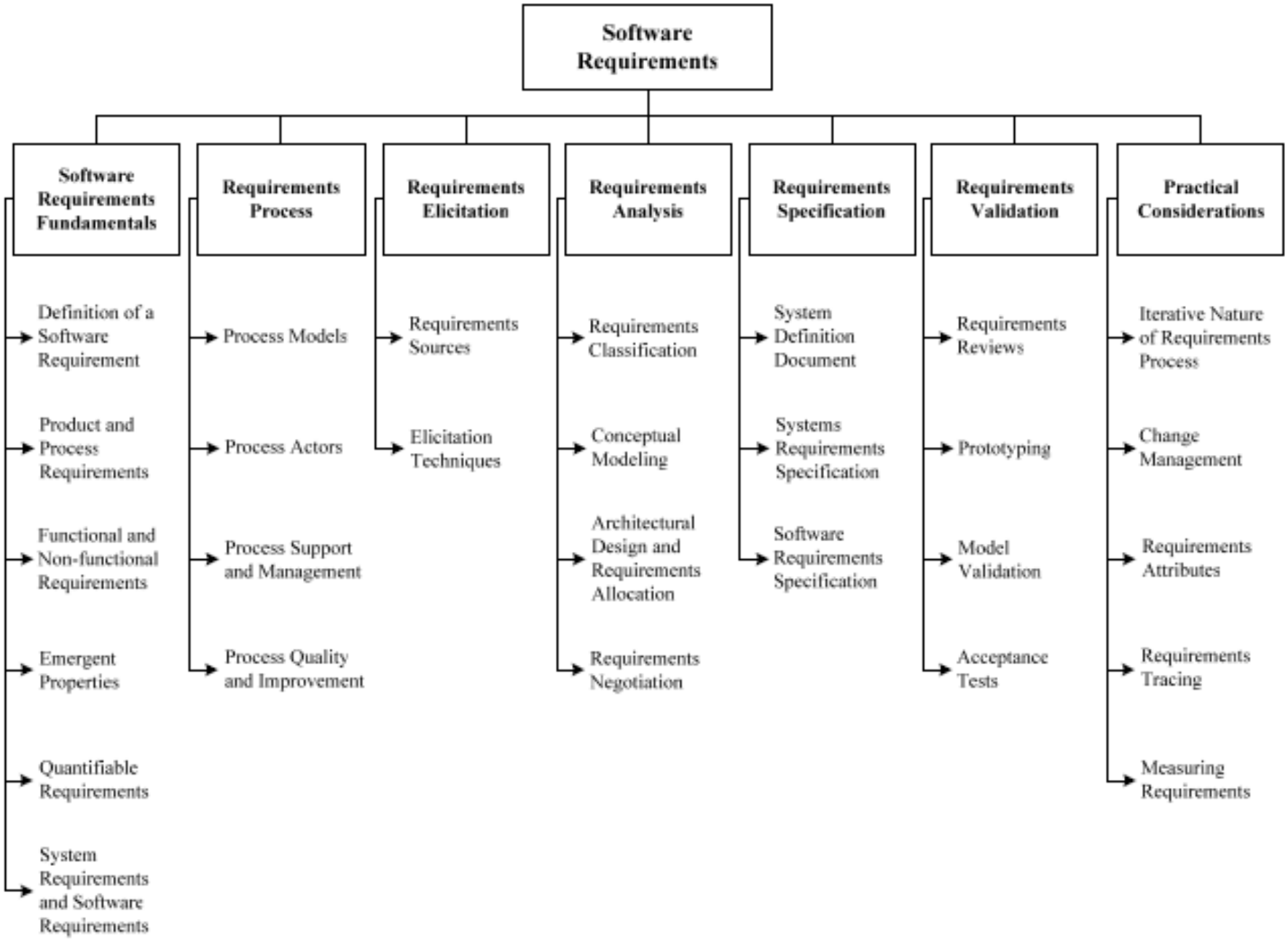
- Sistemas de redes abiertas de performance crítica, y que son muy costosos de apagar.

IS ¿SABER TODO?



IS ¿SABER TODO?




Software Requirements
Software Requirements Fundamentals
Definition of a Software Requirement
Product and Process Requirements
Functional and Non-functional Requirements
Emergent Properties
Quantifiable Requirements
System Requirements and Software Requirements
Requirements Process
Process Models
Process Actors
Process Support and Management
Process Quality and Improvement
Requirements Elicitation
Requirements Sources
Elicitation Techniques
Requirements Analysis
Requirements Classification
Conceptual Modeling
Architectural Design and Requirements Allocation
Requirements Negotiation
Requirements Specification
System Definition Document
Systems Requirements Specification
Software Requirements Specification
Requirements Validation
Requirements Reviews
Prototyping
Model Validation
Acceptance Tests
Practical Considerations
Iterative Nature of Requirements Process
Change Management
Requirements Attributes
Requirements Tracing
Measuring Requirements

IS: PROBLEMAS PRINCIPALES

- La planificación y la estimación de costos son frecuentemente imprecisas.
- La productividad no se corresponde con la demanda.
- La calidad del software no llega a ser a veces ni aceptable.

CONCLUSIONES

- Trabajamos con algo muy complejo.
- Que trata con una problemática muy amplia.
- Que tiene dificultades esenciales.
- Y que además está evolucionando…
- Hay muchos temas por cubrir.

Nos vamos a concentrar en aquellos que más afectan la construcción de software…

CONCLUSIONES

… No existe un único enfoque para mejorar el mal del software. Sin embargo, mediante la combinación de:

- métodos completos para todas las fases del desarrollo del software,
- mejores herramientas para automatizar estos métodos,
- bloques de construcción más potentes para la implementación del software,
- mejores técnicas para garantizar la calidad del software,
- y una filosofía predominante para coordinación, control y gestión

➔*podemos conseguir una disciplina para el desarrollo del software: "Ingeniería del Software" …*

REFLEXIÓN

…Hacer software sería perfecto

si no existieran, ni hardware, ni usuarios… *(Boria, 1987)*

Boria, Jorge. "Ingeniería de Software", (I EBAI), Kapelusz S.A., Bs.As. Argentina, 1987.

BIBLIOGRAFÍA BÁSICA

- Pressman, Roger S. "Ingeniería del Software: un Enfoque Práctico". Sexta Edición. Mc Graw-Hill. 2005.
- Sommerville, Ian. "Ingeniería de Software". Novena Edición. Pearson Educación. 2010.

BIBLIOGRAFÍA ADICIONAL

- [Brooks, 1987] Brooks, F.P. "No Silver Bullet - Essence and Accidents in Software Engineering". IEEE Computer, 20(4): 10-19, 1987.
- [IEEE,1990] IEEE Standard 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology.
- [IEEE, 2004]. SWEBOK – Guide to the Software Engineering Body of Knowledge. IEEE.
- [Mahoney, 2004] Mahoney, M.S. "Finding a History for Software Engineering". IEEE Annals of the History of Computing. Pp. 8-19.

OBJETIVOS DE LA CLASE

- ✓ Productos de Software.
- ✓ Características del Software.
- ✓ Tipos de Software.
- ✓ Proceso de Desarrollo.
- ✓ Ciclo de Vida.
- ✓ IS: una solución?

¿Dudas, consultas?