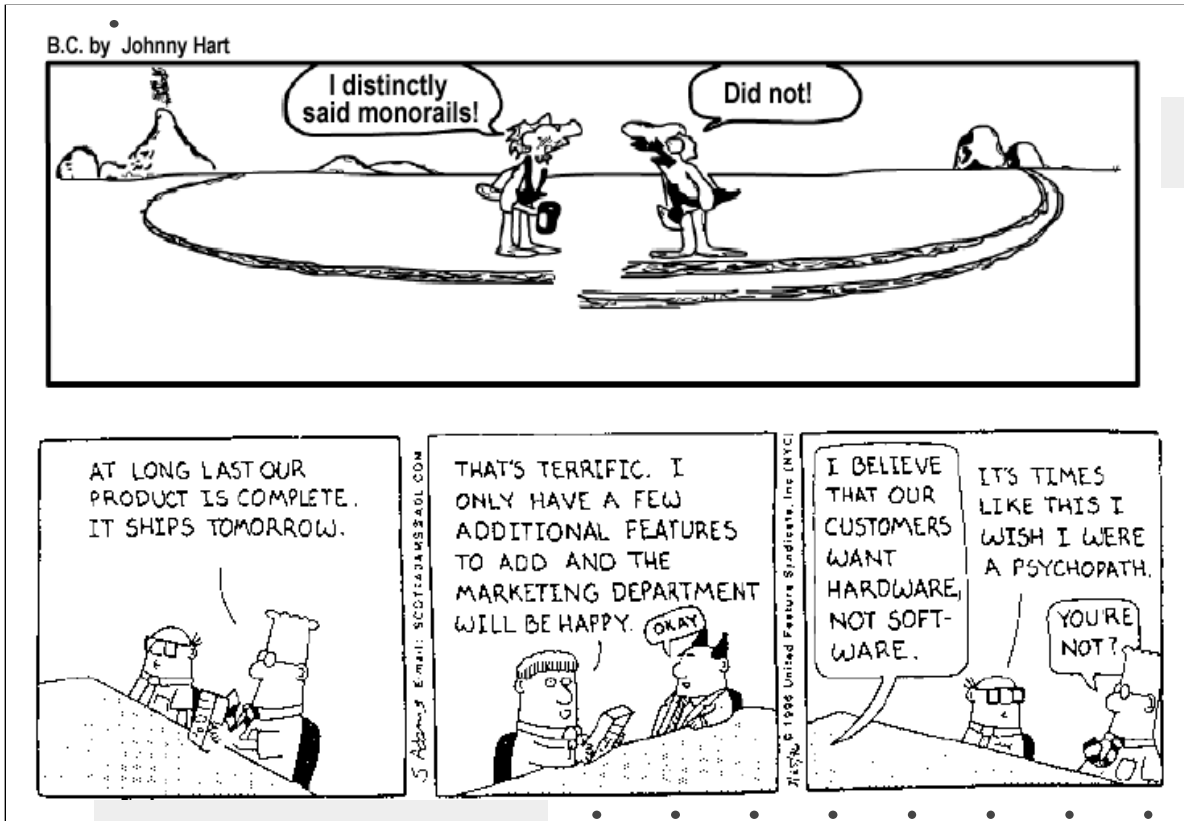


# Ingeniería de Requisitos

## Introducción

Andrés Silva - UDIS - FI - UPM  
(asilva@fi.upm.es)



•  
•  
•

## Ejemplo

- 18.1.2 Guiado: Cuando el piloto mueva la palanca hacia la derecha, el alerón derecho se elevará adecuadamente y el alerón izquierdo bajará adecuadamente. Igualmente, cuando el piloto mueva la palanca hacia la izquierda, el alerón derecho bajará adecuadamente y el alerón izquierdo se elevará adecuadamente.

3

- 
- 
- 

## Refinado

- 18.1.2 Guiado
- 18.1.2.1 Dirección: Cuando el piloto mueva la palanca hacia la derecha, el alerón derecho se elevará y el alerón izquierdo bajará. Igualmente, cuando el piloto mueva la palanca hacia la izquierda, el alerón derecho bajará y el alerón izquierdo se elevará.
- 18.1.2.2 Graduación: Movimientos incrementales de la palanca corresponderán a movimientos incrementales de los alerones.

4

•  
•  
•

## Refinado

Resulta que hoy, gracias al software se puede proteger la integridad estructural del avión:

- 18.1.2.2 Graduación: Movimientos incrementales de la palanca corresponderán a movimientos incrementales de los alerones, sin superar nunca la integridad estructural del avión.

5

•  
•  
•

## Y seguimos refinando

- 18.1.2.2.1 Seguridad: Si el piloto mueve bruscamente la palanca, el avión ejecutará el mayor giro posible que no exceda la tolerancia estructural del avión.
- 18.1.2.2.1.1 El sistema monitorizará la velocidad del aire
- 18.1.2.2.1.2 El sistema monitorizará ...
- ...
- 18.1.2.2.1.n El sistema calculará el mayor ángulo posible compatible con las restricciones estructurales del avión.

6

• • • • • • • •

⋮

## Ingeniero Jefe Boeing 777

“El software no es el problema. Los requisitos son el problema... En otras ingenierías las leyes naturales limitan el crecimiento de los requisitos, pero el software no tiene esas leyes... Nuevos requisitos crecen como un tumor hasta que se pierde el control del proyecto”

7

•  
•  
•

## Contenido (según SWEBOK)

- I. Introducción
- II. El proceso de Requisitos
- III. Educación de Reqs.
- IV. Análisis de Reqs.
- V. Documentación de Reqs.
- VI. Validación de Reqs
- VII. Gestión de Reqs.
- VIII. La IR hoy

8

La organización de estas clases se basa en la organización que, para el área de Ingeniería de Requisitos, es promovida por el proyecto “SWEBOK: Software Engineering Body of Knowledge”. Este proyecto pretende definir un Curriculum básico universal para establecer los programas de formación de los futuros Ingenieros de Software, así como una posible certificación.



⋮

## Material que se proporciona

- Esta presentación (requisitos.pdf)
- IEEE std. 1362 y std. 830 (ieeeCO.pdf, ieee.pdf)
- Ejemplos (ejemplo\_URD.rtf, ejemplo\_ERS.rtf)
- Herramientas de ayuda para Ing. Req (REtools.pdf)

9

•  
•  
•

## Lista de lecturas recomendadas

- Leffingwell D., Widrig D. “Managing Software Requirements: A Use Case Approach”, 2nd Edition. Addison-Wesley, 2003.
- Ian K. Bray. “An Introduction to Requirements Engineering”. Addison-Wesley, 2002.
- Alexander I., Stevens R., “Writing better requirements”. Pearson Education, 2002.
- K. Wiegers “Software Requirements”, Microsoft Press, 1999
- **B. L. Kovitz “Practical Software Requirements: A Manual of Content and Style”, Manning, 1999.**
- **G. Kotonya, I. Sommerville. “Requirements Engineering. Processes and Techniques”. Wiley, 1998.**
- **I. Sommerville, P. Sawyer “Requirements Engineering. A Good Practice Guide”, Wiley, 1998.**
- M. Jackson “Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices” Addison-Wesley, 1995.
- A. Davis. “Software Requirements: Objects, Functions and States” Prentice-Hall, 1993.
- **D. C. Gause, G. M. Weinberg “Exploring Requirements: Quality Before Design” Dorset House, 1989.**
- INCOSE tools survey: <http://www.paper-review.com/tools/rms/read.php>

10

• • • • • • • •

⋮

## PARTE I

### Introducción

Lo más difícil en la construcción de un sistema software es decidir precisamente qué construir... No existe tarea con mayor capacidad de lesionar al sistema, cuando se hace mal... Ninguna otra tarea es tan difícil de rectificar a posteriori...

F. P. Brooks, 1987

11

F. P. Brooks es el autor de “The Mythical Man-Month”, quizá el único libro “clásico” de la Ingeniería del Software (Addison-Wesley, 1975). Brooks fue jefe de proyecto del OS/360, el sistema operativo del IBM/360. A lo largo de este enorme proyecto, Brooks padeció todos los males que constituyeren lo que habitualmente se conoce como “crisis del software”.

⋮

## La evidencia empírica demuestra que

- Los requisitos contienen demasiados errores
- Muchos de estos errores *no se detectan* al principio
- Muchos de estos errores *podrían ser* detectados al principio
- No detectar estos errores incrementará los costes (tiempo, dinero) de forma exponencial
- Además, los programadores obedecen los requisitos (cuando existen).

12

Fuente: Libro de Alan Davis que figura en la Bibliografía.

Experimento de Gause & Weinberg: A cinco equipos de programadores se les proporcionan una serie de requisitos pero a un grupo se le pide que minimice la memoria utilizada, a otro que elabore un código claro, a otro que optimice el tiempo de ejecución etc. El resto de los requisitos eran idénticos. Finalmente, cada equipo elaboró un producto que satisfacía los requisitos solicitados, demostrando que es falso eso de que “Los programadores no hacen lo que el usuario desea”. Lo que ocurre es que los programadores no pueden hacer lo que el usuario desea si desconocen esos deseos.

•  
•  
•

## Consecuencias

- El sistema resultante no satisfará a los usuarios
- Se producirán desacuerdos entre usuarios y desarrolladores
- Puede ser imposible demostrar si el software cumple, o no, los requisitos
- Se gastará tiempo y dinero en construir el sistema equivocado

13

•  
•  
•

## La complejidad

- OJO: No estamos hablando de sistemas de 15 o 30 requisitos. También hay
  - Sistemas de cientos de requisitos
  - Sistemas de miles de requisitos
  - Sistemas de 5.000 requisitos
  - Sistemas de más de 10.000
  - Sistemas de incluso 60.000 (!) requisitos

14

•  
•  
•

## 1995/2001: The Chaos Report

<http://www.standishgroup.com>

- Gasto anual en EEUU: \$250 mil millones en unos 175.000 proyectos.
- 31,1% (23%) son cancelados
- 52,7% (49%) cuestan un 190% más de lo estimado
- Un 16,2% (28%) será finalizado a tiempo y dentro del presupuesto, pero el producto final poseerá (aprox.) la mitad de los requisitos iniciales

15

• • • • • • • •

## La crisis del software y los requisitos

- Boehm, 1975: 45% de los errores tienen su origen en los requisitos y en el diseño preliminar.
- DeMarco, 1984: 56% de los errores que tienen lugar en un proyecto Sw. Se deben a una mala Especificación de Requisitos
- Chaos Report: Los factores principales que conducen al fracaso en los proyectos Sw. Son
  - Falta de comunicación con los usuarios
  - Requisitos incompletos
  - Cambios a los requisitos
- Hoy. Parece que se mejora, pero muy poco a poco. ☹

16

Otras historias de horror relacionadas con los requisitos:

- Uno de los estudios más conocidos es el de la General Accounting Office (GAO) de EEUU. Este estudio de 1979 reveló que 47% del dinero empleado en proyectos software se destinó a sistemas que no llegaron a utilizarse. Otro 29% se empleó en proyectos que no llegaron a finalizar. Otro 19% se empleó en software que tuvo que ser profundamente modificado tras su entrega. Finalmente, tan sólo un 2% del dinero se empleó en proyectos software que sí cumplieron con sus requisitos, pero se trataba de proyectos más bien pequeños o de poca envergadura.
- En 1981, Victor Basili encontró cerca de 88 errores en una ERS de 400 páginas para el proyecto “A-7E Operational Flight Program”. Esta ERS había sido escrita por un grupo de expertos en especificación de requisitos.
- Recientemente, la NASA ha sufrido varios accidentes espectaculares cuyo origen se atribuye a problemas durante la definición de los requisitos.

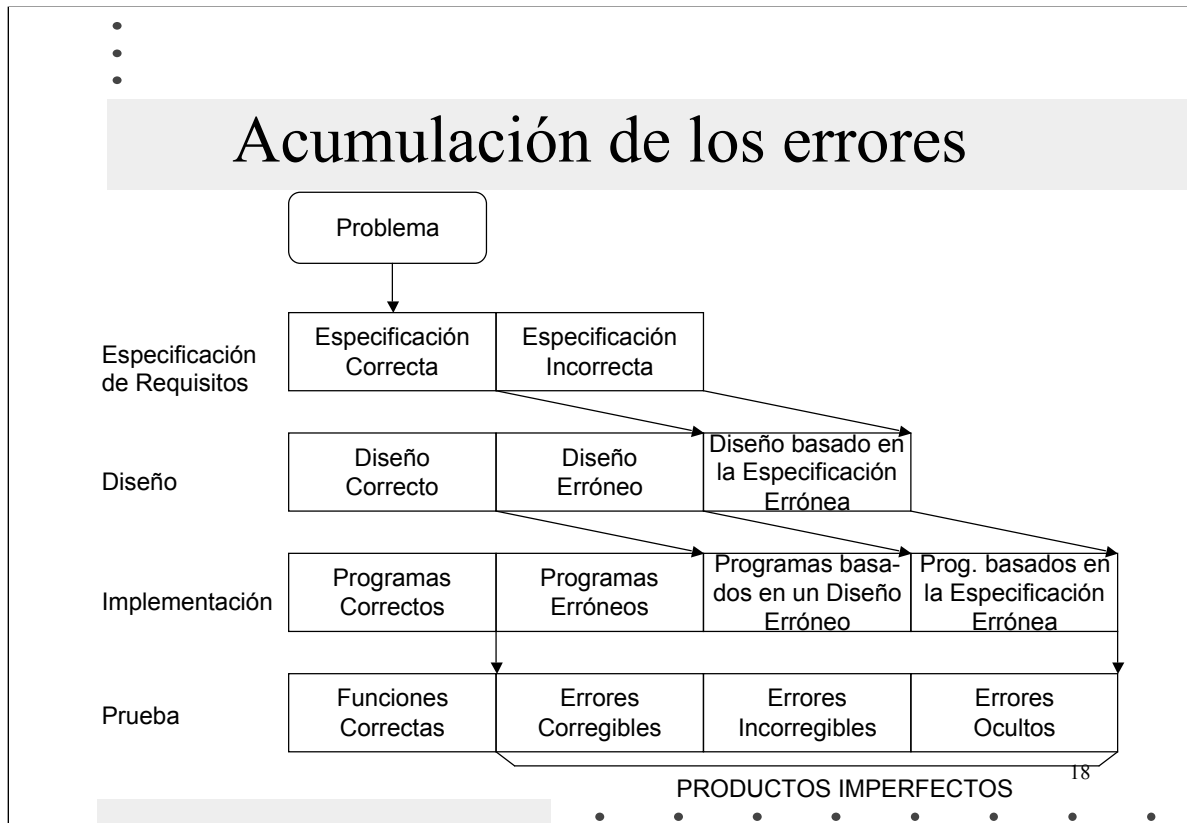


•  
•  
•

## Coste de los errores

<u><i>Etapa</i></u>	<u><i>Coste de la reparación</i></u>
Requisitos	1-2
Diseño	5
Codificación	10
Pruebas unitarias	20
Pruebas sistema	50
Explotación/Mtmtto.	200

17



Como se ve, los errores cometidos en los requisitos son los más peligrosos, pues sus consecuencias contaminan todas las restantes fases del ciclo de vida.

•  
•  
•

## ¿Qué podemos hacer?

- Tomar conciencia del problema. Estar a la defensiva.
- No conocemos todas las respuestas, pero conocemos muchas de las preguntas. Tenemos experiencia.
- Podemos minimizar el impacto de los errores en los requisitos
- Podemos tratar de organizar mejor las tareas relacionadas con los requisitos
- **MAS RECURSOS PARA LA FASE DE REQUISITOS !!!**

19

• • • • • • • •

⋮

## ¿Soluciones definitivas?

- No se ha encontrado solución universalmente válida
- Hay serias dudas acerca de si dicha solución existe
  - Nos movemos en la frontera socio-técnica de los sistemas: borrosa, voluble e inconsistente
  - Los requisitos es donde lo formal se encuentra con lo informal (M. Jackson)
  - Los requisitos estan vivos: emergen, interactúan, cambian, desaparecen...
- Desconfiad de quien ofrezca la solución definitiva a estos problemas !!!

20

• • • • • • • •

⋮

## Ingeniería de Requisitos

- Para remediar *en lo posible* esta situación, surge la Ingeniería de Requisitos (IR)
- **La IR trata de los principios, métodos, técnicas y herramientas que permiten descubrir, documentar y mantener los requisitos para sistemas basados en computadora, de forma *sistemática y repetible*.**
- La comparación del Chaos Report de 1995 con 2002 parece indicar que no vamos por el mal camino...

21

Esto no pretende ser una definición de la IR. De hecho, distintos autores proporcionan distintas definiciones, e incluso, el mismo autor, en distintas ocasiones, aporta definiciones distintas.

⋮

## Qué son los requisitos (Jackson & Zave)

- Todo problema sw. consiste en configurar una máquina para que ejerza unos efectos R en un dominio K. La conexión se realiza a través de una interfaz S.
  - Los efectos R son los requisitos: Necesidades, metas, objetivos
  - El conocimiento del dominio K describe el contexto.
  - La máquina (hw+sw) es la que realizará los requisitos R, gracias a S, que describe la conexión con el dominio K.
  - En la fase de requisitos tan sólo necesitamos K, S y R. No necesitamos describir el comportamiento interno de la máquina
- Idealmente:  $K \text{ y } S \Rightarrow R$

22

R: NO son realidades, pero serán realidades cuando construyamos el sistema.

K: Describe realidades. Afirmaciones cuya verdad es independiente de la existencia, o no, del sistema.

S: Describe comportamiento externo del sistema (interfaz)

Idealmente:

$K \text{ and } S \Rightarrow R$

•  
•  
•

## Ejemplo K,S y R

- Sistema de control de una caldera de vapor. Ejemplos:
  - K: “el agua entra en ebullición a 100 Grados Centígrados y a 1 atm. de presión”
  - R: “El sistema evitará que el agua entre en ebullición”
  - S: “El sistema leerá la temperatura del agua por medio del sensor S342”, “El sistema podrá subir la temperatura del agua por medio del regulador R452”
- Propiamente hablando tan sólo R son “requisitos”, pero a la Ingeniería de Requisitos le interesan también K y S, pues todas son necesarias.
- Pero K, S y R no son suficientes...

23

• • • • • • • •

•  
•  
•

## Otros contenidos relevantes

- Información acerca del problema a solucionar
- Propiedades y comportamiento del sistema
- Restricciones de diseño y fabricación del producto
- Descripciones acerca de cómo el futuro sistema ayudará a sus usuarios a realizar mejor sus tareas
- Restricciones acerca de la tecnología que será utilizada en la construcción del sistema (protocolos, SSOO, COTS, etc.)
- Restricciones acerca de las propiedades emergentes del sistema (requisitos no funcionales)
- Los requisitos NO son análisis top-down, ni son la arquitectura del sistema, ni son el “qué” frente al “cómo”.

24

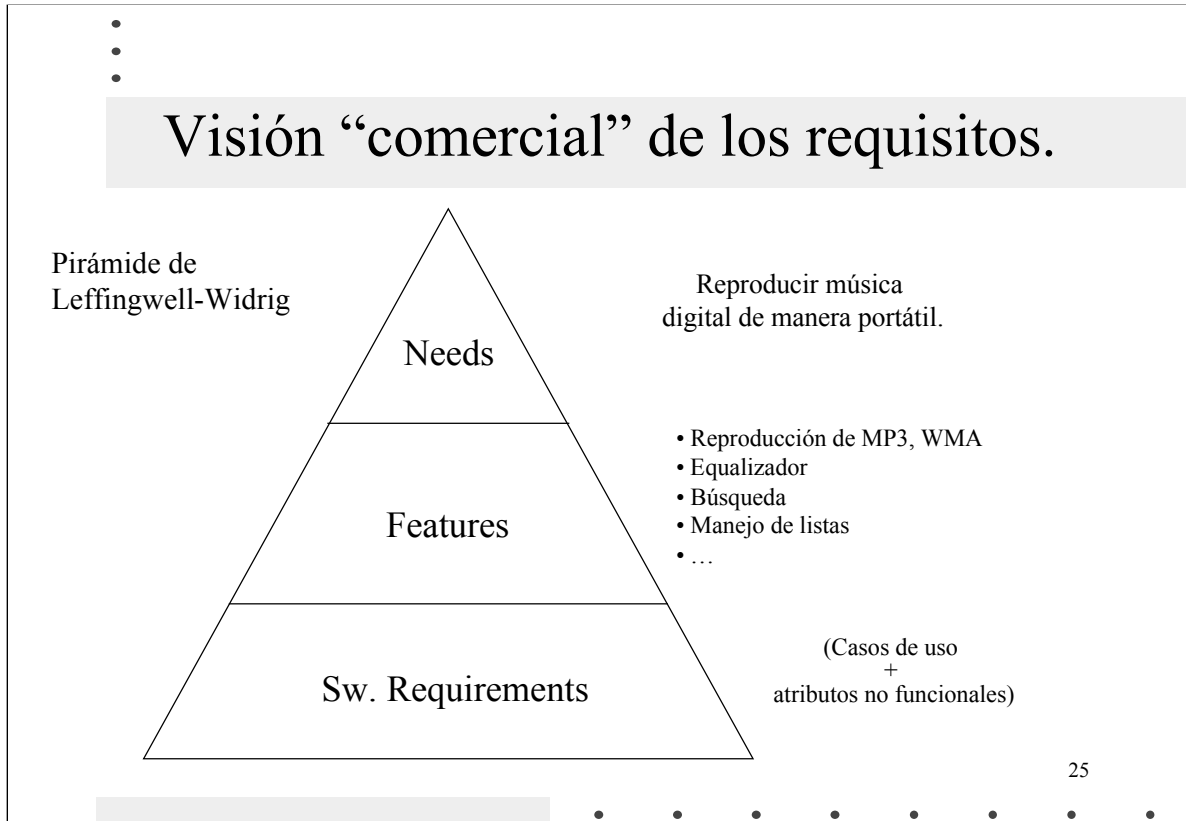
El “qué” de una persona es el “cómo” de otra. Ejemplo:

Qué: minimizar el número de personas que circulan por las escaleras

Cómo: Instalando un ascensor.

PERO para la empresa instaladora de ascensores, instalar un ascensor es un qué, no un cómo. El cómo del ascensorista es, ¿qué herramientas utilizo?, ¿qué proceso sigo? ¿qué medidas de seguridad aplicar?





⋮

## ¿Cómo escribir requisitos?

- La “mejor forma” de escribir requisitos no existe
- Lo más utilizado es el lenguaje natural. Cada requisito expresado en una frases corta (“el sistema hará X ...”, “se facilitará Y...”, etc)
- O lenguaje natural complementado con diagramas y/o notaciones formales
- La notación utilizada depende de quien lee o quien escribe los requisitos.

26

• • • • • • • •

•  
•  
•

### Ejemplo de lo que podemos encontrar en un documento de “requisitos”

1. El sistema mantendrá la temperatura de la caldera entre 70° y 80°
  2. El sistema mantendrá un registro de todos los materiales de la biblioteca, incluyendo libros, periódicos, revistas, videos y CDRoms
  3. El sistema permitirá a los usuarios realizar una búsqueda por título, autor o ISBN
  4. El interfaz de usuario se implementará sobre un navegador Web
  5. El sistema deberá soportar al menos 20 transacciones por segundo
  6. El sistema permitirá que los nuevos usuario se familiaricen con su uso en menos de 15 minutos.
- OJO: Por “documento” no debemos entender sólo papel sino...

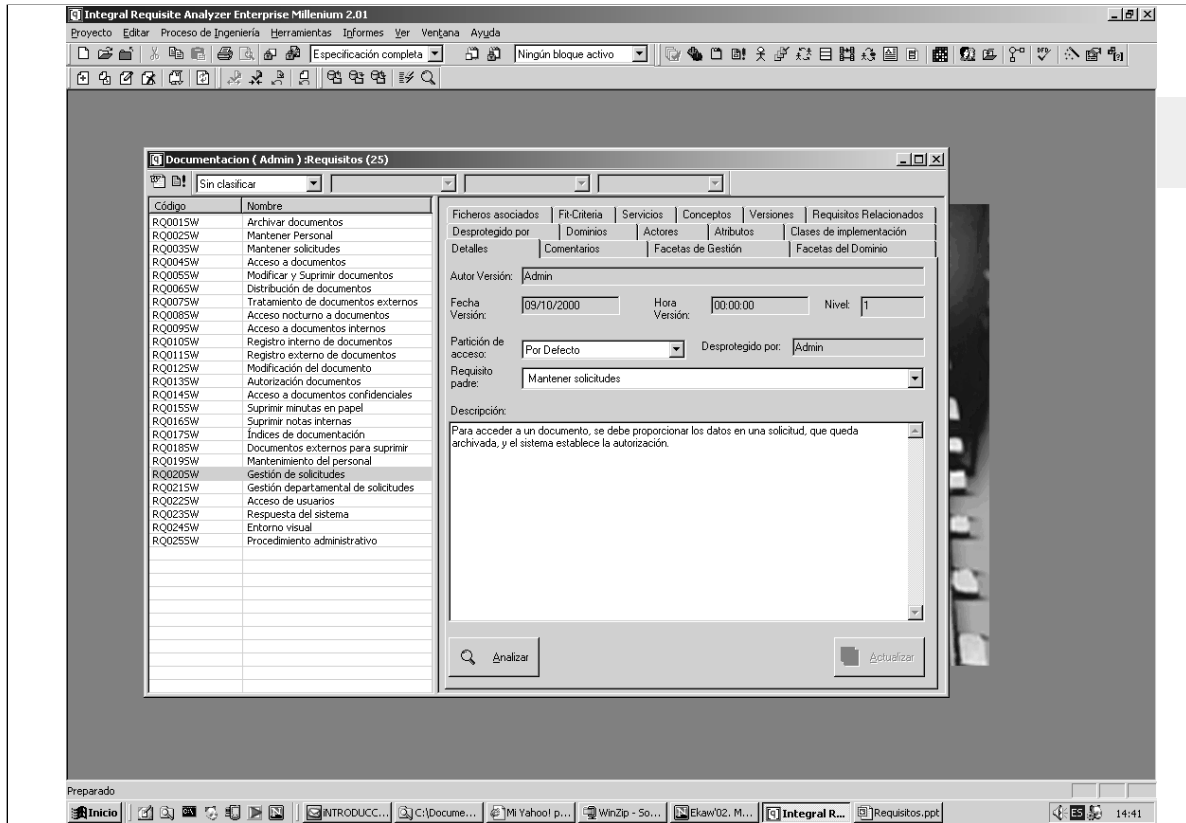
27

Aquí puede verse que los requisitos son de muy distintos tipos:

- Requisitos que definen efectos sobre el entorno (p.ej. El 1)
- Requisitos muy generales (p.ej, el 2)
- Requisitos funcionales (3)
- Requisitos de implementación (4)
- Requisitos de rendimiento (5)
- Requisitos de usabilidad (6)

Debido a que hay tantos tipos distintos de requisitos, no es posible establecer una forma estándar de escribirlos. Tampoco es posible decir cual es “la mejor forma” de especificarlos. Todo depende mucho de quien los escribe, quien los va a leer, el dominio de la aplicación, etc.

# Especificación de Requisitos



⋮

## Requisitos en negativo

- Es importante decir lo que el sistema NO debe hacer
- Estos requisitos “en negativo” limitan el ámbito del sistema.
- Dicen donde NO se deben emplear recursos
- Fundamental para sistemas críticos
  - Se debe mantener la distinción liveness/safety
    - Liveness: dicen lo que el sistema debe hacer
    - Safety: dicen lo que el sistema no debe hacer

29

Normalmente, se afronta la definición de los requisitos “en positivo”, tratando de recoger aquellas necesidades que se deberían materializar en el sistema. Pero establecer lo que el sistema NO debe hacer es casi tan importante como establecer lo que debe hacer.

⋮

## Reqs. funcionales y no funcionales

- Los requisitos funcionales describen los servicios (funciones) que se esperan del sistema
  - El sistema aceptará pagos con VISA
- Los requisitos no funcionales son restricciones sobre los requisitos funcionales
  - El sistema aceptará pagos con VISA
    - de forma segura y con un tiempo de respuesta menor de 5 segundos
- Pero esta distinción, muchas veces, resulta arbitraria.
  - El sistema aceptará pagos con VISA a través del protocolo SET

30

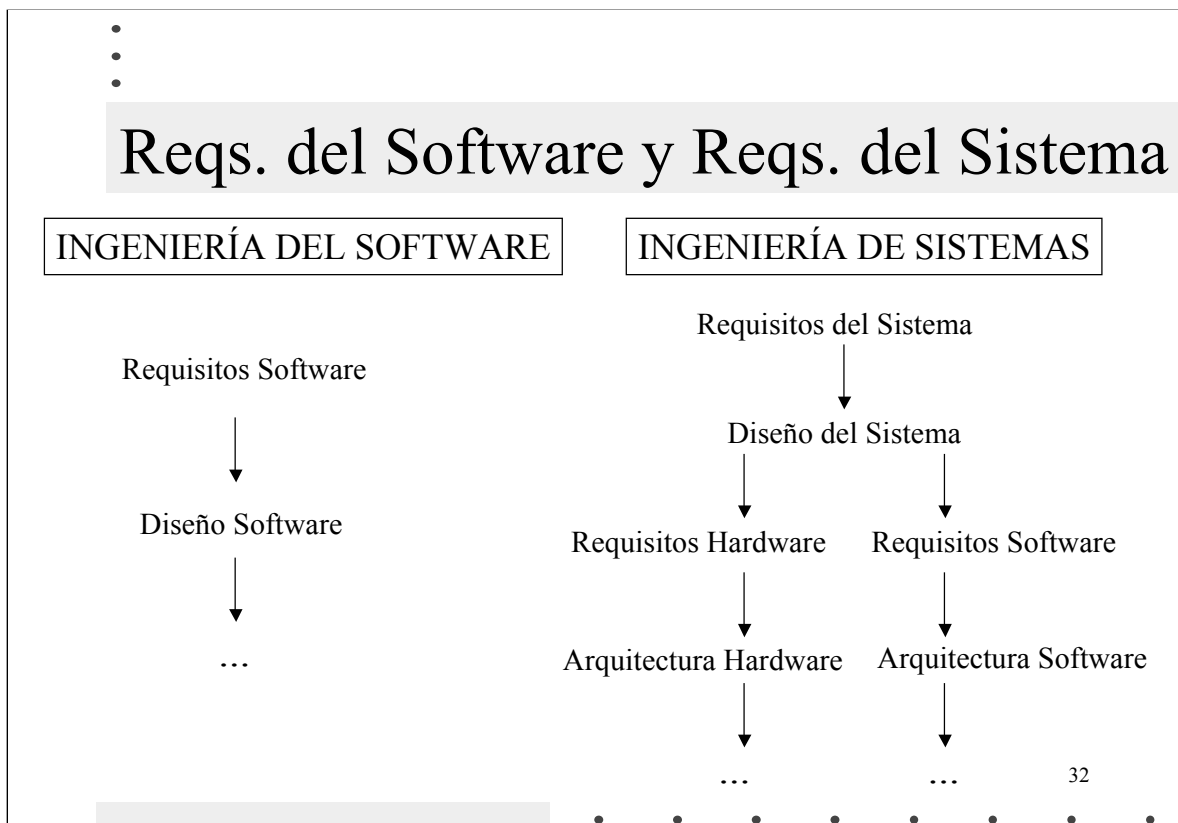
•  
•  
•

## Tipos de Reqs. No Funcionales

- Orientados al usuario:
  - Fiabilidad
  - Seguridad (“security”)
  - Seguridad (“safety”)
  - Usabilidad
  - Robustez
  - Disponibilidad
  - Rendimiento
    - Tiempo de respuesta
    - Capacidad (carga)
    - Throughput
- Orientados al desarrollador:
  - Disponibilidad
  - Portabilidad
  - Adaptabilidad
  - Testabilidad
  - Comprensibilidad
- Etc,...

31

• • • • • • • •



La figura de la izquierda refleja el ciclo de desarrollo para sistemas compuestos exclusivamente de software, como pueden ser los Sistemas de Información o un software de proceso de textos.

La figura de la derecha representa el ciclo de desarrollo para sistemas compuestos de Hardware y Software, es decir, todo tipo de sistemas empotrados y sistemas en los que el software interactúa fuertemente con el hardware (sistemas de control, sistemas de monitorización, electrónica de consumo, etc). En este caso, el software está supeditado a un diseño del sistema y a unos requisitos de nivel superior.

Uno de los problemas más difíciles en Ingeniería de Sistemas (Hw+Sw) es el problema de la asignación (“allocation”): Consiste en decidir qué partes del sistema (o qué tareas) van a ser ejecutadas por el Hw y cuales van a ser programadas en Sw.

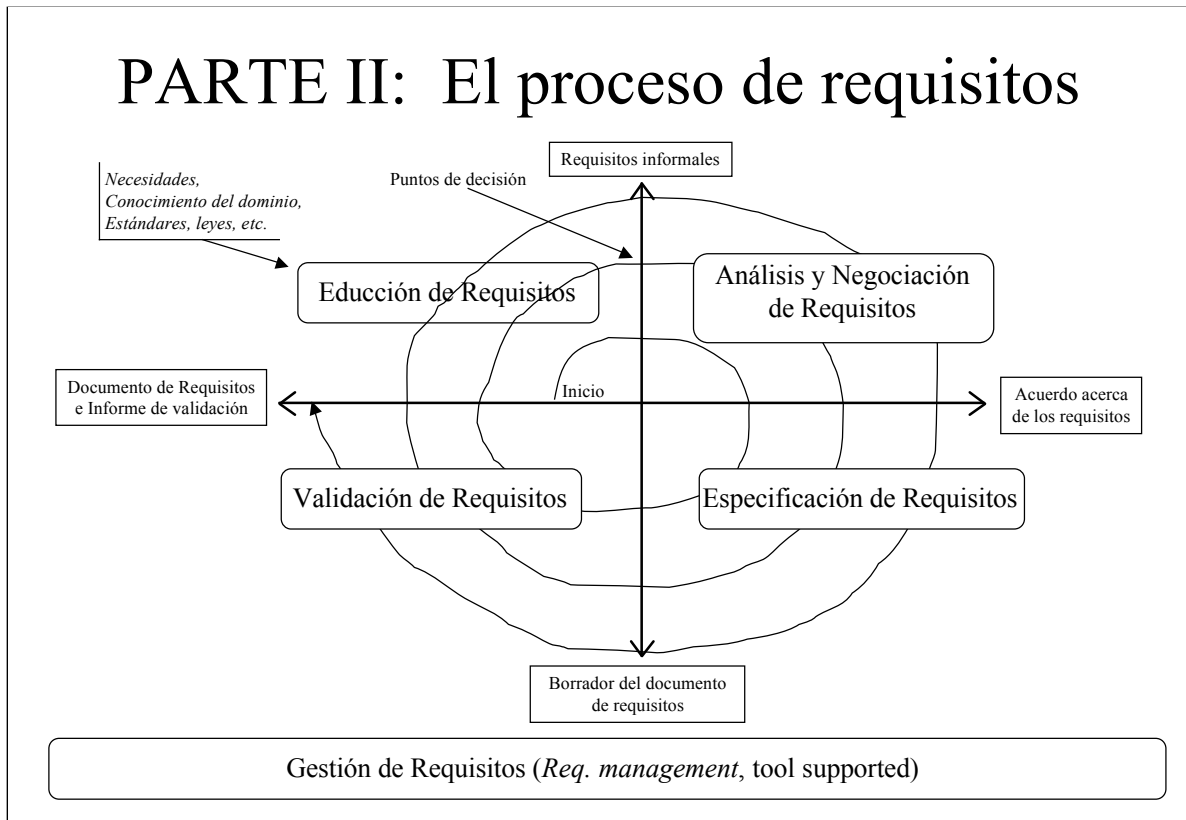


⋮

## ¿Relación Requisitos-Arquitectura?

- La elección de una determinada arquitectura sw. debe tener en cuenta los requisitos funcionales pero, sobre todo, los requisitos no funcionales
- No hay una regla definitiva para establecer, dados los requisitos, el tipo de arquitectura
- Tan sólo hay una serie de heurísticas para, dados unos requisitos, elegir la arquitectura.
- En la práctica, es difícil desarrollar uno sin el otro

33



El proceso de Ingeniería de Requisitos es un conjunto estructurado de actividades que sirven para derivar, validar y mantener los requisitos de un sistema (hardware, software o hardware+software).

En el dibujo, los cuadros redondeados son tareas. Los cuadrados son productos (inputs o outputs). En el resto de esta presentación se explicará cada componente de este proceso.

**La separación que aquí se ofrece es más conceptual que real.** Las distintas tareas que se ejecutan durante el proceso de requisitos suceden en paralelo y se solapan unas con otras. Por ejemplo, durante un proceso de educación de requisitos empleando prototipado, es inevitable realizar una pequeña validación de los requisitos que se van obteniendo, o incluso una pequeña negociación, si estamos tratando con varios usuarios a la vez.

•  
•  
•

## Variaciones en el proceso

- Según la naturaleza del proyecto
  - dirigido al mercado
  - a medida
- Según la naturaleza de la aplicación
  - Riesgo
  - Recursos
  - Incertidumbre
  - Sistemas empotrados

35

•  
•  
•

## Tenemos problemas en el proceso si:

- El proceso de requisitos es excesivamente largo y costoso
- Los implicados en el proceso se quejan de falta de tiempo u otros recursos
- Se reciben quejas acerca de la inteligibilidad del documento de requisitos
- Los implementadores se quejan del trabajo perdido por culpa de errores en los requisitos
- Los usuarios no utilizan muchas de las capacidades del sistema
- En cuanto el producto se entrega, se recibe una inmensa cantidad de solicitudes de cambios
- Lleva demasiado tiempo alcanzar un acuerdo cuando se proponen cambios a los requisitos

36

• • • • • • • •



## PARTE III

### Educción

- La educación de requisitos se refiere a la captura y descubrimiento de los requisitos.
- Es una actividad más “humana” que técnica
- Se identifica a los interesados (*stakeholders*) y se establecen las primeras relaciones entre ellos y el equipo de desarrollo

37

•  
•  
•

## Fuentes de requisitos

Los requisitos pueden proceder de:

- Metas: Factores críticos de éxito (de la empresa)
- Conocimiento del dominio de la aplicación
- Los interesados. Los afectados por el sistema.
- El entorno físico que rodea al sistema
- El entorno organizacional. Los procesos de negocio.

38

Los requisitos pueden proceder de:

- Metas: Factores críticos de éxito
- Conocimiento del dominio de la aplicación. Por ejemplo, si el analista tiene experiencia en realizar sistemas para compañías de seguros, puede sugerir requisitos que no son obvios para los usuarios, o puede deducir mejor las consecuencias de los requisitos propuestos por los usuarios. Por ejemplo, un usuario quiere consultar por pantalla todas las pólizas que venzan durante el mes. Para que ello sea posible, el software deberá obligar, cada vez que se crea una póliza, a que se introduzca su fecha de vencimiento. Esto puede resultar obvio para un informático, pero no lo es tanto para un usuario inexperto.
- Interesados. Afectados por el cambio
- El entorno que rodea al sistema
- La organización. Los procesos de negocio.

⋮

## Problemas de la educación

- Los usuarios no pueden/saben describir muchas de sus tareas
- Mucha información importante no llega a verbalizarse
- A veces hay que “inventar” los requisitos
  - sistemas orientados a miles de usuarios: web, mercado
- La educación no debería ser un proceso pasivo, sino cooperativo

39

- 
- 
- 

## Hacer preguntas no es suficiente.

### Ejemplo: “Diseñar un medio de transporte”

- Diseñador: ¿para cuando?
- Cliente: 18 meses
- Diseñador: ¿qué hay que transportar?
- Cliente: personas
- Diseñador: ¿Cuántas personas a la vez?
- Cliente: Una
- Diseñador: ¿Qué clase de energía lo mueve?
- Cliente: El pasajero
- Diseñador: ¿sobre que clase de superficie?
- Cliente: Una superficie plana y dura
- Diseñador: ¿Cuál es la distancia máxima a recorrer?
- Cliente: Unos 2 km.
- Diseñador: ¿Cuál es el coste?
- Cliente: Unos 300 francos suizos cada vez que se use.
- Etc.
- Finalmente: se fabrica un triciclo.
- El cliente (alcalde de Grindelwald) destaca su gran portabilidad (?).
- Dice: “Es muy portable pero ¿cómo se utiliza cuando llegue el momento de rescatar a un alpinista en la cara norte del monte Eiger?”

**LO QUE SE QUERÍA ERA  
UN DISPOSITIVO QUE AYUDE  
A RESCATAR ALPINISTAS !**

40

- 
- 
- 
- 
- 
- 
- 
-



•  
•  
•

## Técnicas de educación

- Preliminares: Utilizar preguntas libres de contexto
  - (ver RETools.pdf)
- Brainstorming
- Creatividad? (herramientas como ideafisher, curio, mindjet, etc.)
- Entrevistas: Es el método “tradicional”
- Observación y análisis de tareas
- Casos de uso / Escenarios: requisitos en contexto de uso
- Prototipado: Útiles cuando la incertidumbre es total acerca del futuro sistema. Hay dos tipos principales:
  - Evolutivo
  - De usar y tirar (prototipos en papel, mago de Oz, etc.)

41

Ahora no se dedicarán clases a los casos de uso/escenarios, porque se hablará de ellos dentro del tema de Orientación a Objetos.

•  
•  
•

## Entrevista

- Es la aproximación tradicional a la educación, pero
  - Debe usarse en complemento con otras técnicas
  - No debe ser el primer paso de la educación
- Es fundamental:
  - Entrevistar a la(s) persona(s) adecuadas
  - Preparar las preguntas con antelación
  - Utilizar diagramas, modelos, etc.

42

• • • • • • • •

•  
•  
•

## Brainstorming

- **Objetivo:** Generar un gran número de ideas
  - Seleccionar un grupo variado de participantes
  - Eliminar críticas, juicios y evaluaciones mientras los participantes sugieren ideas
  - Producir **muchas** ideas. “La calidad emerge de la cantidad”
  - Recogerlas todas por escrito
  - Otro día, en otra sesión, se evalúan las ideas
    - Por ejemplo, pueden puntuarse (de -3 a +3)

43

⋮

## Tarjetas de recogida de requisitos

#Requisito	Clasificación:	Tipo:
Descripción:		
Razón:		
Orígen:		
Prioridad:		
Dependencias:	Conflictos:	
Referencias:		
Historia:		

44

⋮

En esta transparencia se presenta una “tarjeta de requisitos” (inspirada en el libro de S. Robertson) que puede ser útil en proyectos reales. Dichas tarjetas sirven para recoger información relevante.

- #Requisito: Número que identifica a cada requisito
- Clasificación: En qué apartado del documento de requisitos debería figurar. En otras palabras, a qué parte del sistema afecta (por ejemplo, pedidos, contabilidad, ventas, etc.)
- Tipo: Funcional, no-funcional
- Descripción: Una frase que describe el requisito. Preferentemente de la forma “el sistema debe...”
- Razón: ¿Por qué es importante este requisito?
- Origen: ¿quién lo pide?
- Prioridad: Importancia del requisito. Puede valorarse de 0 a 3, por ejemplo.
- Dependencias: Otros requisitos de los que depende
- Conflictos: Requisitos que, de alguna forma, contradicen a éste
- Referencias: Qué otros materiales son necesarios para comprender el requisito (por ejemplo, documentos)
- Historia: Historia de los cambios del requisito

⋮

## Análisis de Tareas

- La gente tiene problemas para verbalizar lo que hace.
  - ¿Cómo cambia vd. de marchas en el coche?
- Por eso surgen los métodos “etnográficos”
  - Un observador estudia a los futuros usuarios en su entorno de trabajo. A veces se utiliza el video.
  - Anota todo aquello que es susceptible de mejora
  - Posteriormente, genera una serie de requisitos tentativos

45

## Uso de vídeo en IR

The screenshot displays two windows from the PRO-ART 2.0 software. The left window, 'PRO-ART 2.0 Whiteboard-Editor', shows a tree view of requirements under 'ADITEC-MEDIA', including 'ADITEC-RWE1' and its sub-items like 'schedule\_order', 'unknown\_interaction', 'report\_set-up', and 'retrieving\_orders'. Below this is a video player showing a person interacting with a computer screen. The right window, 'PRO-ART 2.0 Goal Editor - ADITEC-Production-Management (Version 1)', shows a goal hierarchy. The root goal is 'G1 Support information exchange for production management'. It branches into 'G1.1 Support production order transfer' and 'G1.2 Support production time reports'. 'G1.1' further branches into 'G1.1.1 Orders are scheduled', 'G1.1.2 Orders are distributed to production', and 'G1.1.3 Orders are retrieved by worker'. 'G1.2' branches into 'G1.2.1 Production times are measured' and 'G1.2.2 Production times are received by central eyes'. Each goal node includes a table with numerical values and a status icon. A large black arrow points from the video player to the 'G1.1.1' goal node.

P. Haumer, K. Pohl, K. Weidenhaupt. "Requirements elicitation and validation with real world scenes". *IEEE Transactions on Software Engineering (TSE)*, 24(12), Dec. 1998, pp. 1036-1054.

En este ejemplo el video se ha utilizado para derivar escenarios de uso de un sistema y analizarlos, con objeto de descubrir carencias en el sistema actual.

## Casos de uso. Ejemplo.

<b>Nombre del Caso de Uso</b>	Control de luces
<b>Requisito relacionado</b>	REQ-USR-232
<b>Descripción</b>	Este caso de uso muestra como las luces se apagan y encienden y como el usuario puede regular su nivel de iluminación.
<b>Flujo de eventos básico</b>	Comienza cuando el usuario presiona el botón ON/OFF/DIMM en el panel de control. Cuando el usuario deja de presionar, el sistema actualiza el estado de la luz, como sigue: -Si la luz está encendida, se apaga -Si la luz está apagada, se enciende con el nivel de intensidad memorizado la última vez.
<b>Flujo alternativo</b>	Si el usuario presiona el botón más de 3 segundos se inicia un ciclo de luminosidad, del nivel más bajo al más alto, hasta que el usuario deja de presionar.
<b>Precondiciones</b>	-El botón ON/OFF/DIMM está habilitado
<b>Postcondiciones</b>	- El sistema memoriza el nivel de luminosidad para ese botón
<b>Extensioes</b>	Ninguna

47

•  
•  
•

## Ejercicio

- Aplicar las “context-free questions” al siguiente problema:

**El propósito de nuestro producto es ayudar al usuario a moverse fácilmente por la ciudad, guiándole desde una dirección a otra.**

48

Preguntas libres de contexto (context-free questions). Ver archivo Retools.rtf

- 1 Respecto al proceso
  - ¿Quién es el cliente?
  - ¿Qué ventajas aportaría una solución para el cliente?
  - ¿Cuál es la verdadera razón para resolver el problema?
  - ¿Deberíamos utilizar a un grupo de personas o a varios?
  - ¿Quién debería formar parte de dichos grupos?
  - ¿Cuánto tiempo tenemos para el proyecto?
  - ¿Cuál es el balanceo tiempo - dinero?
  - ¿Hay algún lugar de donde se pueda obtener una solución al problema?
  - ¿Podemos imitar algo que ya existe?

- 2 Respecto al producto
  - ¿Qué problema resolverá el producto?
  - ¿Qué problemas creará el producto?
  - ¿En qué ambiente funcionará el producto?
  - ¿Cuál será la precisión deseada del producto?

- 3 Metapreguntas
  - ¿Estoy haciendo demasiadas preguntas?
  - ¿Cree que mis preguntas tocan aspectos relevantes?





## PARTE IV

### Análisis de Requisitos

- Consiste en detectar y resolver conflictos entre requisitos
- Se precisan los límites del sistema y la interacción con su entorno
- Se trasladan los requisitos de usuario a requisitos del software (implementables).
- Se realizan tres tareas fundamentales:
  - Clasificación
  - Modelización
  - Negociación

•  
•  
•

## Clasificación de los requisitos

- En el análisis de requisitos, éstos se clasifican
  - En funcionales vs. no funcionales (Capacidades vs. Restricciones)
  - Por prioridades
  - Por coste de implementación
  - Por niveles (alto nivel, bajo nivel)
  - Según su volatilidad/estabilidad
  - Si son requisitos sobre el proceso o sobre el producto
- En Ingeniería de Sistemas, además, se realiza la ubicación de requisitos (*Requirements Allocation*)

50

•  
•  
•

## Modelización conceptual

- Ciertos aspectos de los requisitos se expresan mediante modelos de datos, de control, de estados, de interacción, de objetos, etc.
- La meta es entender mejor el problema, más que iniciar el diseño de la solución (idealmente)
- El tipo de modelo elegido depende de
  - La naturaleza del problema
  - La experiencia del modelizador
  - La disponibilidad de herramientas
  - Por decreto. El cliente impone una notación

51

Tradicionalmente se entendía que “el análisis” se reducía a modelizar (DFDs, modelos de objetos, etc), pero el análisis de requisitos **NO** es exclusivamente un proceso de modelización, como ya se ha dicho.

Por otro lado, no existe “la mejor” forma de modelizar requisitos. En realidad, no hay evidencia empírica que demuestre, **en general**, la superioridad de unas notaciones de modelización frente a otras.

•  
•  
•

## Negociación de requisitos

- En todo proceso de IR intervienen distintos individuos con distintos y, a veces, enfrentados intereses
- Estos conflictos entre requisitos se descubren durante el análisis. Todo conflicto descubierto debería disparar un proceso de (re)negociación.
- Los conflictos NUNCA se deben resolver “por decreto”

52

En todo proceso de IR intervienen distintos individuos con distintos y, a veces, enfrentados intereses. Por ejemplo, distintos individuos de distintos departamentos pueden desear requisitos conflictivos entre sí, o conflictivos con los objetivos globales del sistema o de la organización. Incluso a veces, detrás de un requisito ambiguo lo que hay, en realidad, es un conflicto no resuelto.

Es durante el análisis cuando muchos de los conflictos entre requisitos son descubiertos. EL CONFLICTO NO ES RECHAZABLE y no debe resolverse por decreto, sino mediante un proceso de negociación. Desde este punto de vista, los conflictos son positivos, pues SON FUENTE DE NUEVOS REQUISITOS. Los acuerdos alcanzados deben ser convenientemente anotados, favoreciéndose así la trazabilidad de los requisitos a sus orígenes (“el requisito 987 surge como resultado de la reunión entre x, y z el día tal del tal...”)

Existe, incluso, un subcampo de la IR dedicada a este tipo de temas: la IR orientada a Puntos de Vista o “Viewpoint-Based Requirements Engineering”.



## PARTE V

### El Documento de Requisitos

- Es el modo habitual de guardar y comunicar requisitos.
- Es buena práctica utilizar, al menos, dos documentos, a distinto nivel de detalle
  - DRU = Documento de Requisitos de Usuario (en inglés, URD)
  - ERS = Especificación de Requisitos Software (en inglés, SRS)
- **OJO:** Con “Documento” nos referimos a cualquier medio electrónico de almacenamiento y distribución:
  - Procesador de textos
  - Base de Datos
  - Herramienta de Gestión de Requisitos

53

DRU/URD: Documento de Requisitos de Usuario/User Requirements Document.

ERS/SRS: Especificación de Requisitos Software/Software Requirements Specification.

La pregunta que surge es ¿en qué se diferencian los “requisitos de usuario” de los “requisitos del software”? A grandes rasgos se puede afirmar que:

- El DRU se escribe desde el punto de vista del usuario/cliente/interesado. Normalmente los requisitos de usuario, contenidos en la DRU, no poseen demasiado nivel de detalle. Se incluye la descripción del problema actual (razones por las que el sistema de trabajo actual es insatisfactorio) y las metas que se espera lograr con la construcción del nuevo sistema.
- La ERS desarrolla mucho más los contenidos de la DRU. Los requisitos del software contenidos en la ERS son, por tanto, más detallados. Contiene la respuesta a la pregunta ¿Qué características debe poseer un sistema que nos permita alcanzar los objetivos, y evitar los problemas, expuestos en la DRU?

OJO: La diferencia entre la DRU y la ERS **NO** es que la DRU emplee lenguaje natural y la ERS emplee modelos o notaciones formales. Tanto la DRU como la ERS pueden utilizar todo tipo de notaciones. La diferencia entre ambos documentos es más de nivel de detalle.

•  
•  
•

## Estándares

- IEEE Std. 1362 (ConOps)
- IEEE Std. 830 (SRS)
- PSS-05 de la Agencia Espacial Europea (ESA)
- Las herramientas de gestión de requisitos permiten generar documentación en los anteriores formatos, a partir de una base de datos de requisitos.

54

Los documentos de IEEE se proporcionarán en el web de la UDIS

⋮

## Esquema del estándar de IEEE 830

- Introducción
  - Propósito
  - Alcance
  - Definiciones
  - Referencias
  - Visión General
- Descripción General
  - Perspectiva del producto
  - Funciones del producto
  - Características del usuario
  - Restricciones
  - Suposiciones
- Requisitos específicos
- Apéndices
- Índice

55

• • • • • • • •

Los detalles de qué es lo que debe contener cada sección se pueden encontrar en el archivo [ieee.pdf](#) disponible en el Web de la UDIS

## Características deseables de una ERS

Una ERS *de calidad* debería poseer 24! características:

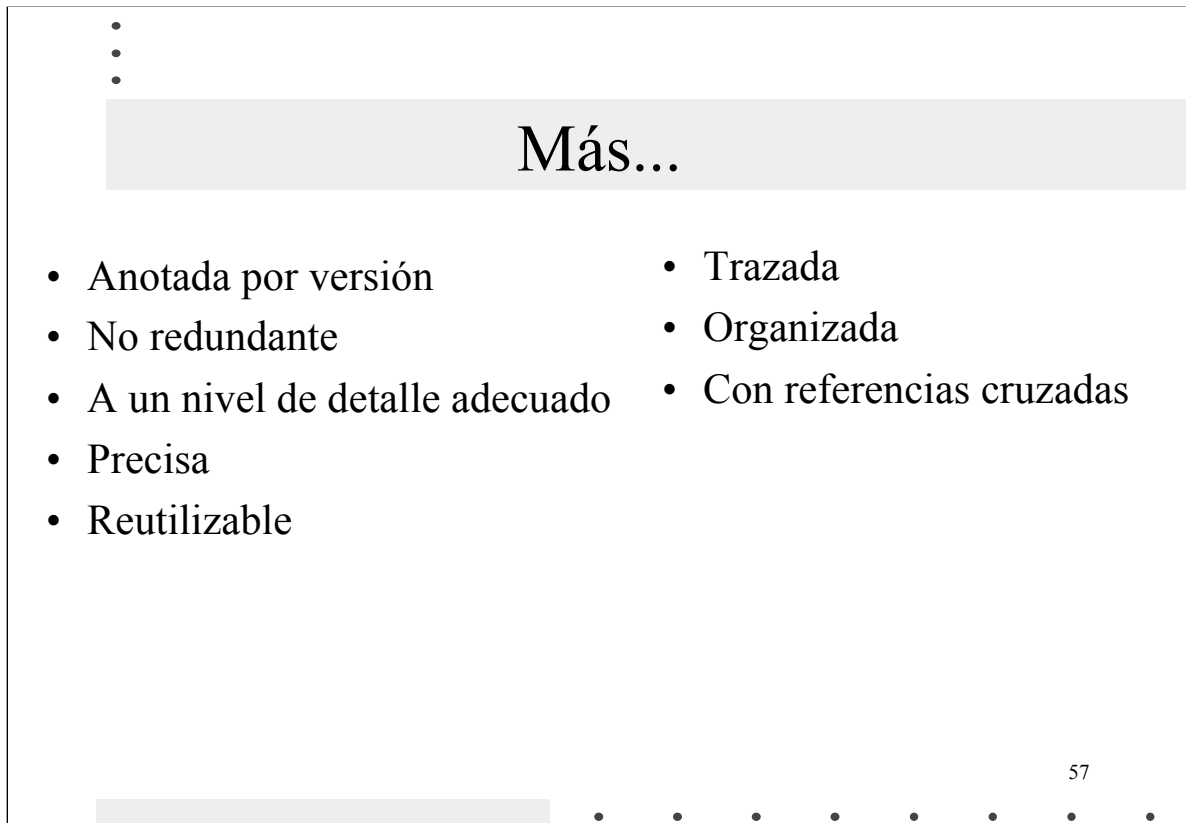
- No ambigua
- Completa
- Correcta
- Comprensible
- Verificable
- Internamente consistente
- Externamente consistente
- Realizable
- Concisa
- Independiente del diseño
- Trazable
- Modificable
- Electrónicamente almacenada
- Ejecutable/Interpretable
- Anotada por rel. importancia
- Anotada por rel. estabilidad

56

- No ambigua: La ERS es no ambigua si todo requisito posee una sólo interpretación
- Completa: Una ERS es completa si todo lo que se supone que el software debe hacer está incluido en la ERS. Por completitud, deberían describirse todas las posibles respuestas a todas las posibles entradas y en todas las situaciones posibles. Además, la ERS no contendrá secciones de tipo “por determinar...”
- Correcta: Todo requisito de la ERS contribuye a satisfacer una necesidad real.
- Comprensible: Todo tipo de lectores (clientes, usuarios, desarrolladores, equipo de pruebas, gestores, etc.) entienden la ERS.
- Verificable: Si para cada requisito expresado en la ERS existe un procedimiento de prueba finito y no costoso para demostrar que el futuro sistema lo satisface.
- Internamente Consistente: No existen subconjuntos de requisitos contradictorios.
- Externamente Consistente: Ninguno de los requisitos está en contradicción con lo expresado en documentos de nivel superior. Por ejemplo, en un sistema (hardware+software), los requisitos del software no pueden contradecir los requisitos del sistema.
- Realizable: Si, dados los actuales recursos, la ERS es implementable
- Concisa: La ERS debe ser lo más breve posible, sin que esto afecte al resto de atributos de calidad

(sigue...)





•  
•  
•

## Más...

- Anotada por versión
- No redundante
- A un nivel de detalle adecuado
- Precisa
- Reutilizable
- Trazada
- Organizada
- Con referencias cruzadas

57

• • • • • • • •

- Independiente del diseño: Existen más de un diseño e implementación que realizan la ERS. Para ello la ERS debería limitarse a describir el comportamiento externo del sistema.
- Trazable: Cada requisito se puede referenciar de forma unívoca. Es fundamental para precisar qué requisitos son implementados por qué componente del diseño, lo cual es imprescindible a la hora de realizar las pruebas de dicho componente
- Modificable: Los cambios son fáciles de introducir.
- Electrónicamente almacenada: Se encuentra en un archivo de texto, en una base de datos o, mejor aún, ha sido creada con una herramienta de gestión de requisitos (RequisitePro, Doors, etc.)
- Ejecutable/Interpretable/Prototipable/Animable: Si existe una herramienta software que, recibiendo como entrada la ERS, realice un modelo ejecutable de la misma. Aplicable tan sólo a ciertas notaciones como las notaciones formales o los diagramas de transición de estados.
- Anotada por importancia relativa: Si los requisitos se clasifican según su importancia. Como mínimo un requisito puede ser “Obligatorio, Deseable u Opcional”. Esto sirve para no asignar demasiados recursos a la implementación de requisitos no esenciales.

(Sigue...)

•  
•  
•

## La calidad como ideal:

- Una ERS perfecta es imposible.
- La calidad de la ERS es muy difícil de cuantificar.
- En general, una ERS de calidad NO garantiza la ausencia de problemas, pero una ERS pésima garantiza su presencia.

58

- Anotada por estabilidad relativa: Los requisitos son, en general, inestables y volátiles. A cada requisito se le asigna una probabilidad de cambio (p.ej. “Alta, Media o Baja”). Esto ayudará a los diseñadores a diferenciar los componentes más flexibles de los más estables.
- Anotada por versión: Si un lector de la ERS puede determinar qué requisitos serán satisfechos por qué versión del producto.
- No redundante: Cada requisito se expresa en un solo lugar de la ERS. La redundancia, de todas formas, no es del todo mala si aumenta la legibilidad.
- Al nivel adecuado de abstracción: Ni demasiado detallada ni demasiado vaga.
- Precisa: Una ERS es precisa si hace uso de valores numéricos para precisar las características del sistema. La precisión es aplicable, ante todo, a los requisitos no funcionales. Por ejemplo, no es útil decir “El tiempo de respuesta será más bien rápido”, sino “El tiempo de respuesta será menor que dos segundos”. OJO: en la práctica diaria, la precisión es difícilísima de conseguir pues es fuertemente dependiente de la tecnología disponible, la cual no siempre se conoce al principio de un proyecto.
- Reutilizable: Si ciertas secciones de la ERS se pueden reutilizar
- Trazada: Si está claro el origen de cada requisito (quién o qué lo pide)
- Organizada: Si el lector puede fácilmente encontrar la información buscada.
- Con referencias cruzadas: Si se utilizan referencias entre requisitos relacionados (trazabilidad intra-ERS) o entre secciones relacionadas.



## PARTE VI

### Validación de Requisitos

- Objetivo: Descubrir problemas en el Documento de Requisitos antes de comprometer recursos a su implementación.
- Resultados: se produce una *línea-base* (baseline)
- El documento debe revisarse para
  - descubrir omisiones
  - conflictos
  - Ambigüedades
  - Comprobar la calidad del documento y su grado de adhesión a estándares

59

En el contexto de la Ingeniería de Requisitos, una línea base es un conjunto de requisitos que han sido formalmente aceptados por todas las personas implicadas en el proyecto. Una vez que se establece una línea base, futuros cambios a tales requisitos sólo podrán realizarse por medio de un proceso formal de gestión y aprobación de cambios.

⋮

## Revisiones (Reviews)

- Es la fórmula más empleada para validación
- Un grupo de personas (incluyendo usuarios) se ocupan de revisar el documento de requisitos.
- Tres fases: Búsqueda de problemas, reunión y acuerdos.
- Como guía para identificar problemas habituales, se pueden utilizar listas de comprobación (“checklists”).
- Hay “checklists” adaptadas a distintos tipos de sistemas

60

El 33% de los errores de requisitos en la especificación del sistema A-7E fueron detectados mediante revisión manual. El resto se descubrieron en posteriores fases, con el consiguiente incremento en el coste.

Curiosamente, las revisiones parecen funcionar también con el código ejecutable: se descubren más errores inspeccionando el código fuente que ejecutando el programa. ¿Radica aquí el éxito de los desarrollos Open Source?

Cada organización, según su experiencia y según el dominio de las aplicaciones que desarrolle, debería desarrollar su lista de comprobación o “checklist” particular. Un ejemplo de cuestiones que deberían figurar en una lista de comprobación podría ser esta:

- ¿Están todos los requisitos convenientemente numerados?
- ¿El mismo servicio es solicitado en distintos requisitos? ¿Existen contradicciones entre ellos?
- ¿Los requisitos relacionados se encuentran agrupados en el documento?
- ¿Los requisitos son fácilmente comprensibles? ¿Por todo tipo de lectores?

En general, una lista de comprobación debería girar alrededor de los 24 atributos de calidad que debería poseer una ERS. Para cada atributo de calidad, se pueden plantear una serie de cuestiones que sirvan para confeccionar la lista de comprobación.

•  
•  
•

## Otros métodos de validación

- Prototipado: Permite descubrir con rapidez si el usuario se encuentra satisfecho, o no, con los requisitos
  - Uso de escenarios/casos de uso
- Validación de modelos: Cuando los requisitos se expresan por medio de modelos (de objetos, DFDs, etc.)
- Validación de su “testabilidad”. El equipo de pruebas debe revisar los requisitos.

61

• • • • • • • •

•  
•  
•

## PARTE VII

# Gestión de Requisitos

(Requirements Management, RM)

- Consiste, básicamente, en gestionar *los cambios* a los requisitos.
- Asegura la consistencia ente los requisitos y el sistema construido (o en construcción)
- Consume grandes cantidades de tiempo y esfuerzo
- Abarca todo el ciclo de vida del producto

62

• • • • • • • •

⋮

## ¿Por qué es necesaria?

- Los requisitos son volátiles
- El entorno físico cambia
  - Trasladar un sistema de un entorno a otro requiere modificaciones
- El entorno organizacional cambia
  - Las políticas cambian
  - Cambios en las reglas y en los procesos del negocio provocan cambios en el sistema
- La propia existencia del sistema va a generar nuevos requisitos por parte de los usuarios

63

⋮

## La Gestión de Requisitos implica

- Definir procedimientos de cambios: definen los pasos y los análisis que se realizarán antes de aceptar los cambios propuestos (una Software Change Request (SCR) de la NASA ocupa más de 500 páginas !!!).
- Cambiar los atributos de los requisitos afectados
- Mantener la trazabilidad: hacia atrás, hacia delante y entre requisitos (P.ej. Proyectos para la FDA)
- Control de versiones del documento de requisitos

64



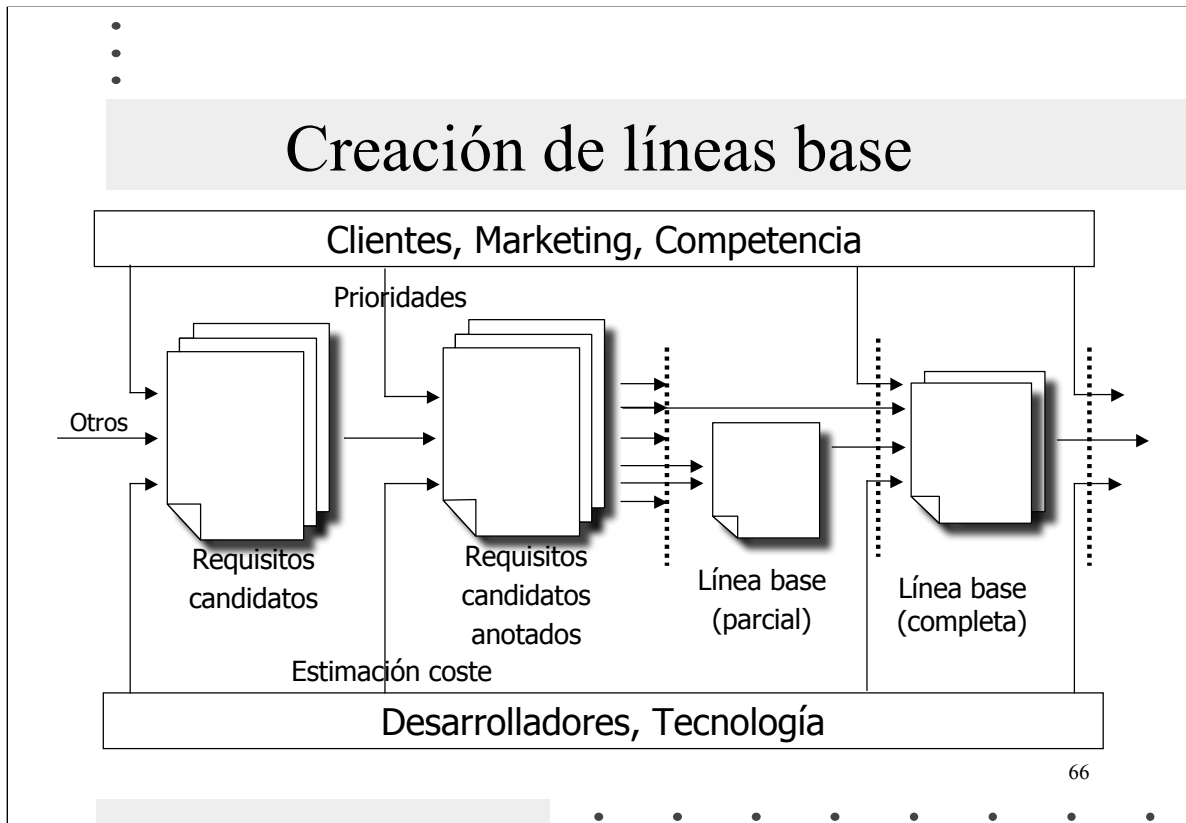
⋮

## Herramientas de Gestión de Requisitos

- Facilitan las tareas relacionadas con la escritura, trazabilidad y gestión de cambios.
- Organizan los requisitos en bases de datos.
- Gestión (incremental) de *lineas-base*
- Todas permiten añadir atributos a los requisitos
- Ejemplos:
  - DOORS (de QSS/Telelogic),
  - RequisitePro (de Rational),
  - IRqA, icConcept, etc.

65

Se ofrecerá una demo de DOORS como complemento a estas clases.



Una línea base (*baseline*) es un conjunto de requisitos que, mediante acuerdo entre las partes implicadas, se ha decidido no modificar. Son aquellos requisitos más importantes que el desarrollador se compromete a implementar. A lo largo de un proceso de requisitos pueden crearse (incrementalmente) varias líneas base.

Las herramientas de gestión de requisitos permiten gestionar fácilmente las líneas base. Por ejemplo, los requisitos que forman parte de una línea base no podrán ser modificados, salvo por usuarios privilegiados.



## PARTE VIII

### Hoy: La IR en la práctica

- Hoy el software es parte casi imprescindible de todo tipo de dispositivos
- Industrias tradicionalmente alejadas del software hoy tienen problemas con los requisitos
  - Cambios de paradigma
  - P. Ej. Industria Automóviles: el 30% del precio final corresponde al software
- Introducir el software en estas industrias está produciendo choques culturales.
- Prevalece lo NO funcional sobre lo funcional

67

Hoy el software es parte casi imprescindible de todo tipo de dispositivos. El software se utiliza para proporcionar valor añadido a productos tradicionales (teléfonos, refrigeradores, coches) y para diferenciarse de los productos de la competencia. Basta con decir que, hoy día, el componente más complejo de un coche es el software (por ejemplo, modelos de Mercedes contienen sistemas empotrados que suman casi 5 Mbytes de código ejecutable),

Industrias tradicionalmente alejadas del software hoy tienen problemas de requisitos. Introducir componentes software es distinto a introducir nuevos componentes físicos. La complejidad que añade el software es de varios órdenes de magnitud respecto a la complejidad que añadiría un nuevo componente físico.

Introducir el software en industrias tradicionales está produciendo choques culturales. Estas industrias poseen tradiciones, normas y procedimientos que no son aplicables al desarrollo de software. El mismo concepto de “prototipo” posee significados muy distintos en Ingeniería Aeronáutica, por ejemplo, y en Ingeniería del Software.

Muchas veces hay que desarrollar un software cuando no se conocen todos los detalles del hardware u otros subsistemas. Además, los plazos de entrega normalmente son cortos e innegociables. Frecuentemente, deberá trabajarse con una arquitectura software lo suficientemente flexible (aún a costa de su eficiencia) que permita incorporar futuros aspectos: patrones que permitan un futuro plug-in de componentes, desarrollos iterativos y/o incrementales, etc. [IEEE Computer, Nov. 2001].

(sigue ...)

•  
•  
•

## Hoy: La IR en la práctica (2/2)

- Costes totales de fabricación:
  - Hoy, el coste del software (desarrollo y mantenimiento) es igual o mayor que el coste del hardware
- Los productos se deben optimizar para:
  - desarrollo, mantenimiento, evolución y mejora del software contenido en el producto.
- Muchas veces hay que desarrollar un software cuando no se conocen los detalles del hardware y en plazos de entrega cortos e innegociables.
  - Subsistemas => Subcontratas
  - Solución? : Arquitectura flexible, desarrollos incrementales.
    - Líneas de producto

68

(...)

Actualmente, los requisitos no funcionales (seguridad, fiabilidad, tiempo de respuesta, disponibilidad, etc.) ocupan más espacio en el documento de requisitos que los requisitos puramente funcionales.

Los requisitos ya no son un problema exclusivo de la industria del software y, desde luego, no son un problema exclusivo de los Sistemas de Información tradicionales. Pero es el software, precisamente, lo que provoca los problemas de requisitos, debido al gran potencial que posee para aumentar la complejidad de un sistema.

Más información en: *Juristo, N.; Moreno, A.M.; Silva, A. Is the European industry moving toward solving requirements engineering problems?* IEEE Software, 19(6), Nov/Dec. 2002, page(s): 70- 77.

## Epílogo

- Sólo habrá una verdadera Ingeniería de Requisitos cuando
  - Todos los posibles sistemas software hayan sido construidos al menos una vez (fin de las novedades) ☺...
  - Los sistemas se instalen en entornos fijos e inamovibles, como sucede con los puentes o los edificios ☺☺...
  - La tecnología deje de evolucionar ☺☺☺...
  - Se pueda predecir el futuro con exactitud ☺☺☺☺...
- Mientras tanto, tenemos la **obligación** de minimizar los daños causados por una pobre gestión de los requisitos.

**FIN**