Fundamentos Ingeniería de Software

TALLER: Desarrollo de Software: Casos Prácticos con Mysql y PHP

UNIVERSIDAD CATOLICA DEL CIBAO (UCATECI)
Micro Unidad I – FUNDAMENTOS DE LA INGENIERIA DE SOFTWARE

Ing. Francisco Santana, MTI
fsantana@uce.edu.do

Introduccion a la Ingenieria de Software

Objetivos:

Comprender la importancia de la Ingeniería de Software en la ciencias de la informática y sus aspectos generales.

Contenido de la Unidad

- Que es la Ingeniería de Software
- Importancia de la Ingeniería de de Software
- El objetivo de la Ingeniería de Software
- Ética de la Ingeniería de Software
- Ciclo de Vida del Software
- Procesos y Actividades
- Introduccion a los Modelos

Que es la Ingenieria de Software?

- «La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software».

- "La ingeniería de software es una disciplina de ingeniería que se interesa por todos los aspectos de la producción de software, desde las primeras etapas de la especificación del sistema hasta el mantenimiento del sistema después de que se pone en operación»

Importancia de la Ingenieria de Software

- Cada vez con mayor frecuencia, los individuos y la sociedad se apoyan en los avanzados sistemas de software. Por ende, se requiere producir económica y rápidamente sistemas confiables.

- A menudo resulta más barato a largo plazo usar métodos y técnicas de ingeniería de software para los sistemas de software, que sólo diseñar los programas como si fuera un proyecto de programación personal. Para muchos tipos de sistemas, la mayoría de los costos consisten en cambiar el software después de ponerlo en operación

Etica de la Ingenieria de Software

- **Confidencialidad:** Por lo general, debe respetar la confidencialidad de sus empleadores o clientes sin importar si se firmó o no un acuerdo formal sobre la misma.
- **Competencia:** No debe desvirtuar su nivel de competencia. Es decir, no hay que aceptar de manera intencional trabajo que esté fuera de su competencia.
- **Derechos de propiedad intelectual:** Tiene que conocer las leyes locales que rigen el uso de la propiedad intelectual, como las patentes y el copyright. Debe ser cuidadoso para garantizar que se protege la propiedad intelectual de empleadores y clientes.
- **Mal uso de computadoras:** No debe emplear sus habilidades técnicas para usar incorrectamente las computadoras de otros individuos.

Antecedentes de la Ingenieria de Software

- El concepto “ingeniería de software” se propuso originalmente en 1968, en una conferencia realizada para discutir lo que entonces se llamaba la “crisis del software” (Naur y Randell, 1969). Se volvió claro que los enfoques individuales al desarrollo de programas no escalaban hacia los grandes y complejos sistemas de software. Éstos no eran confiables, costaban más de lo esperado y se distribuían con demora.

- A lo largo de las décadas de 1970 y 1980 se desarrolló una variedad de nuevas técnicas y métodos de ingeniería de software, tales como la programación estructurada, el encubrimiento de información y el desarrollo orientado a objetos. Se perfeccionaron herramientas y notaciones estándar y ahora se usan de manera extensa.

Antecedentes de la Ingenieria de Software

Décadas de los 40 y 50:

- En estas décadas el coste del hardware era tremendamente superior al del software.
- Se consideraba además que el software se podía desarrollar de la misma forma que se desarrolla el hardware.

Antecedentes de la Ingenieria de Software

Décadas de los 60:

- Éxito de la NASA en misiones espaciales.
- En el NASA/IEEE Software Engineering Workshop de 1966; y las conferencias de la OTAN en 1968 y 1969, se analizó la “crisis del software”, y se plantearon ideas fundamentales como “reutilización” o “arquitectura software”.

- Es la época de los famosos “códigos espagueti” (muy difíciles de entender incluso por quien lo escribía) y la aparición de “héroes” que después de varias noches sin dormir conseguían arreglar a último minuto el software para cumplir los plazos marcados.
- Congreso IFIP se cita por primera vez el concepto de “factoría o fábrica de software”

Antecedentes de la Ingenieria de Software

Décadas de los 70:

- En esta década las organizaciones empezaron a comprobar que los costes del software superaban a los del hardware.
- Parnas propone la descomposición modular y el concepto de ocultamiento de información (information hiding), Chen el modelo E/R y Royce el modelo de ciclo de vida en cascada
- La formación de los profesionales de la Ingeniería del Software se centra entonces en las metodologías estructuradas, que supusieron un avance importante en el análisis y diseño de software.

Antecedentes de la Ingenieria de Software

Décadas de los 80:

- Los problemas de no conformidad de proceso se intentaron resolver con estándares como el DoD-STD-2167 o el MILSTD- 1521B por parte del Departamento de Defensa de EEUU.
- Creado Software Engineering Institute (SEI) de la Universidad Carnegie Mellon, un modelo de madurez de la capacidad software (SW-CMM) que desarrollaría Watts Humphrey.
- La formación de los profesionales del software requiere entonces el manejo de las herramientas CASE, comprender el gran cambio de paradigma

Antecedentes de la Ingenieria de Software

Décadas de los 90:

- Se desarrollan los modelos relacionados con la mejora de procesos software, como Ideal, TSP o PSP, y las normas y estándares de calidad como la ISO 9126, ISO 12207, ISO 9000-3, etc.
- Se consolida la orientación a objetos (OO) como aproximación para el desarrollo de sistemas informáticos, apareciendo más de cien metodologías, que terminan dando lugar a la aparición del Lenguaje de Modelado Unificado (UML) y el Proceso Unificado (UP).
- Multitud de técnicas y conocimientos sobre la construcción de sistemas orientados a objetos: patrones, heurísticas, refactorizaciones,

Antecedentes de la Ingenieria de Software

Décadas del 2000:

- Se firma el “Manifiesto Ágil” como intento de simplificar la complejidad de las metodologías existentes y en respuesta a los modelos “pesados” tipo CMM.
- Se difunden el Desarrollo Software Dirigido por Modelos (DSDM) y las líneas o familias de productos software, que suponen un esfuerzo al Ingeniero del Software al trabajar con modelos de alto nivel como elemento principal del desarrollo y mantenimiento de software. .
- Desarrollo Distribuido de Software
- Ingeniería del Software Empírica (ESE) y la Ingeniería del Software Basada en Evidencias (EBSE

Antecedentes de la Ingenieria de Software

Décadas del 2010:

- Además de afianzarse las líneas descritas en las décadas anteriores, estamos asistiendo a una mayor integración entre la Ingeniería del Software y la Ingeniería de Sistemas -destacando el papel de los requisitos no funcionales y, sobre todo, de la seguridad.

- La importancia de la “Ciencia, Gestión e Ingeniería de los Servicios” que requiere un enfoque interdisciplinar (informática, marketing, gestión empresarial, ciencias cognitivas, derecho, etc.)

- Estamos viendo ya la implantación de la “Ingeniería del Software Continua”, y su correspondiente tecnología y “filosofía” “DevOps”, que logran reducir el tiempo entre que se compromete un cambio en el sistema y que se ponga en producción normal.

Antecedentes de la Ingenieria de Software

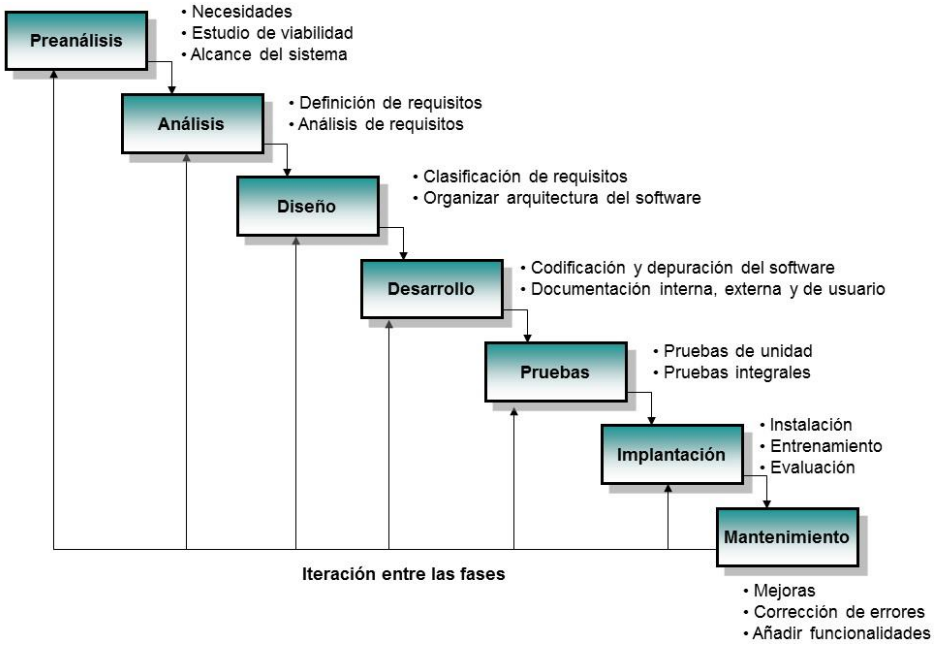
Décadas del 2010 al 2020:

- Aplicaciones en nube y Software como servicios SasS
- Virtualidad en un 50%
- Aplicaciones Moviles y Dispositivos
- Y los waerables (Accesorios)
- Internet de las Cosas

Décadas del 2021 al 2030:

- Internet del comportamiento
- Virtualidad en un 100%
- Avance importante y de gran impacto de la IA

Ciclo de Vida del Software



Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software. IEEE 1074.

Ciclo de Vida del Software

Una aproximación lógica a la adquisición, el suministro, el desarrollo, la explotación y el mantenimiento del software. IEEE 1074.

Ciclo de Vida del Software

Etapas	Disciplina (Roles)
Gestión de Necesidades	Ingeniero de Requisitos o requerimientos
Analisis	Analista de Sistemas, Analista de Software
Diseño	Diseñador o Arquitecto de Software
Codificación	Desarrollador (Junior o Senior), Programador, Lider de desarrollo, Gte de Desarrollo de software.
Validación	Software QA
Pruebas	Software Testing
Documentación	Software Doc Experto
Mantenimiento y Evolución	Implementador, Equipo de Mantenimiento

Que son los Procesos en Ing. De Software

- Un proceso es un conjunto de pasos definidos para resolver un problema.

- Un proceso definido es aquel que esta descrito en tal detalle que permite que los ingenieros lo ocupen constantemente. Los procesos definidos ayudan durante la planeación y el desarrollo del trabajo. [HUMPHREY 2001]

- Un proceso de software es una serie de actividades relacionadas que conduce a la elaboración de un producto de software. Estas actividades pueden incluir el desarrollo de software desde cero en un lenguaje de programación estándar como C#, Java, C, PHP etc

Que son los Modelos

Un modelo de proceso de software es una representación simplificada de este proceso. Cada modelo del proceso representa a otro desde una particular perspectiva y, por lo tanto, ofrece sólo información parcial acerca de dicho proceso.

> **Por ejemplo**, un modelo de actividad del proceso muestra las actividades y su secuencia, pero quizá sin presentar los roles de las personas que intervienen en esas actividades

Tales modelos genéricos no son descripciones definitivas de los procesos de software.

Algunos Modelos

- **El modelo en cascada (waterfall)**
- **Desarrollo incremental:**
- **Ingeniería de software orientada a la reutilización:**
- **Espiral de Boehm**

Modelos: El modelo en cascada (waterfall)

El modelo en cascada (waterfall) :

Éste toma las actividades fundamentales del proceso de especificación, desarrollo, validación y evolución y, luego, los representa como fases separadas del proceso, tal como especificación de requerimientos, diseño de software, implementación, pruebas, etcétera.

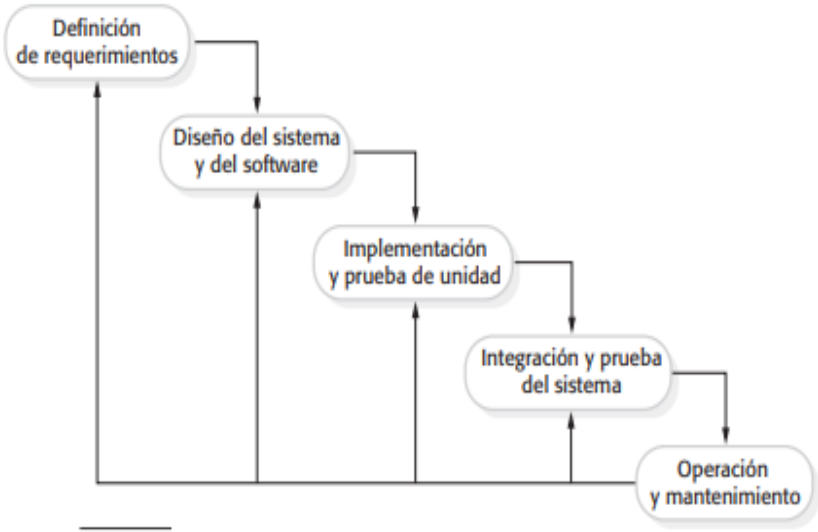


Figura 2.1 El modelo en cascada

Modelos: El modelo en cascada (waterfall) :

Las principales etapas del modelo en cascada reflejan directamente las actividades fundamentales del desarrollo:

- **Análisis y definición de requerimientos:** Los servicios, las restricciones y las metas del sistema se establecen mediante consulta a los usuarios del sistema.

- **Diseño del sistema y del software:** El proceso de diseño de sistemas asigna los requerimientos, para sistemas de hardware o de software, al establecer una arquitectura de sistema global. El diseño del software implica identificar y describir las abstracciones fundamentales del sistema de software y sus relaciones.

- **Implementación y prueba de unidad:** Durante esta etapa, el diseño de software se realiza como un conjunto de programas o unidades del programa. La prueba de unidad consiste en verificar que cada unidad cumpla con su especificación.

Modelos: El modelo en cascada (waterfall) :

- **Integración y prueba de sistema:** Las unidades del programa o los programas individuales se integran y prueban como un sistema completo para asegurarse de que se cumplan los requerimientos de software.

- **Operación y mantenimiento:** Por lo general (aunque no necesariamente), ésta es la fase más larga del ciclo de vida, donde el sistema se instala y se pone en práctica. El mantenimiento incluye corregir los errores que no se detectaron en etapas anteriores del ciclo de vida, mejorar la implementación de las unidades del sistema e incrementar los servicios del sistema conforme se descubren nuevos requerimientos.

Modelos: Desarrollo incremental

Desarrollo incremental:

Este enfoque vincula las actividades de especificación, desarrollo y validación. El sistema se desarrolla como una serie de versiones (incrementos), y cada versión añade funcionalidad a la versión anterior

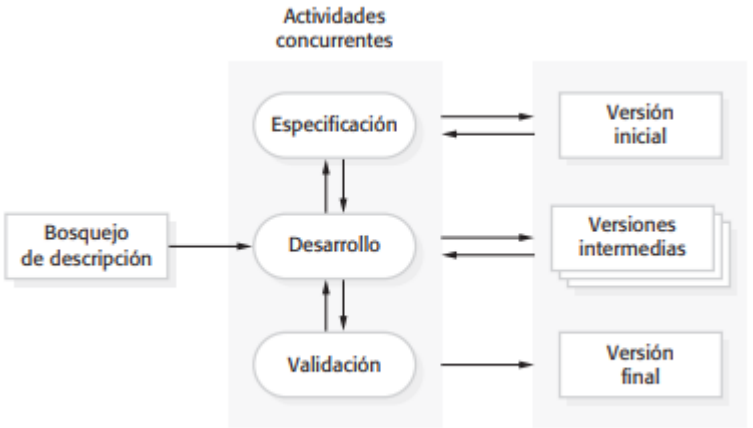


Figura 2.2 Desarrollo incremental

Modelos: Desarrollo incremental

Comparado con el modelo en cascada, el desarrollo incremental tiene tres beneficios importantes:

1. Se reduce el costo de adaptar los requerimientos cambiantes del cliente. La cantidad de análisis y la documentación que tiene que reelaborarse son mucho menores de lo requerido con el modelo en cascada.

2. Es más sencillo obtener retroalimentación del cliente sobre el trabajo de desarrollo que se realizó. Los clientes pueden comentar las demostraciones del software y darse cuenta de cuánto se ha implementado. Los clientes encuentran difícil juzgar el avance a partir de documentos de diseño de software.

3. Es posible que sea más rápida la entrega e implementación de software útil al cliente, aun si no se ha incluido toda la funcionalidad. Los clientes tienen posibilidad de usar y ganar valor del software más temprano de lo que sería posible con un proceso en cascada.

Modelos: Ingeniería de software orientada a la reutilización

Ingeniería de software orientada a la reutilización:

Este enfoque se basa en la existencia de un número significativo de componentes reutilizables. El proceso de desarrollo del sistema se enfoca en la integración de estos componentes en un sistema, en vez de desarrollarlo desde cero.

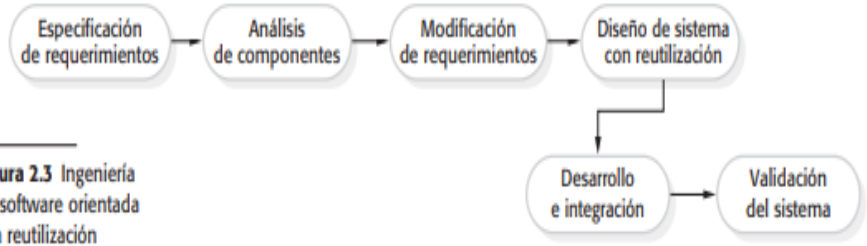


Figura 2.3 Ingeniería de software orientada a la reutilización

Modelos Ingeniería de software orientada a la reutilización

Existen tres tipos de componentes de software que pueden usarse en un proceso orientado a la reutilización:

1. Servicios Web que se desarrollan en concordancia para atender servicios estándares y que están disponibles para la invocación remota.

2. Colecciones de objetos que se desarrollan como un paquete para su integración con un marco de componentes como .NET o J2EE.

3. 3. Sistemas de software independientes que se configuran para usar en un entorno particular. La ingeniería

Modelo: Espiral de Boehm

Boehm (1988) propuso un marco del proceso de software dirigido por el riesgo (el modelo en espiral), que se muestra en la figura 2.11.

Aquí, el proceso de software se representa como una espiral, y no como una secuencia de actividades con cierto retroceso de una actividad a otra. Cada ciclo en la espiral representa una fase del proceso de software. Por ende, el ciclo más interno puede relacionarse con la factibilidad del sistema, el siguiente ciclo con la definición de requerimientos, el ciclo que sigue con el diseño del sistema, etcétera.

El modelo en espiral combina el evitar el cambio con la tolerancia al cambio. Lo anterior supone que los cambios son resultado de riesgos del proyecto e incluye actividades de gestión de riesgos explícitas para reducir tales riesgos.

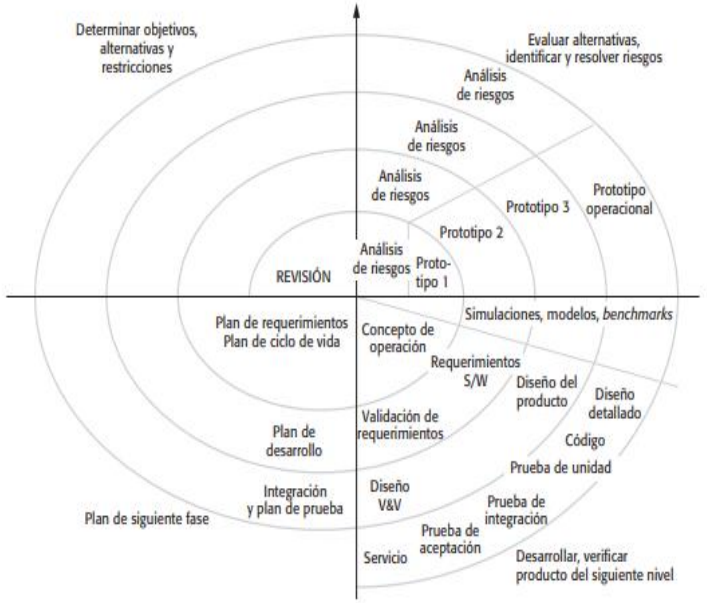


Figura 2.11 Modelo en espiral de Boehm del proceso de software (© IEEE, 1988)

SOCIALIZACION – PREGUNTAS?

Preguntas Y/O Comentario?

Bibliografia

- Tom ReWsta Técnica de la Empresa de Telecomunicadones de Cuba S.A.
- La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software**,** InterSedes: Revista de las Sedes Regionales ISSN: 2215-2458 intersed@cariari.ucr.ac.cr Universidad de Costa Rica Costa Rica
- Ingeniería de Requisitos: de la especificación de requisitos de software al aseguramiento de la calidad. Entre Ciencia e Ingeniería, ISSN 1909-8367 Año 10 No. 20 - Segundo Semestre de 2016, página 117 – 123
- Evolución de la Ingeniería del Software y la formación de profesionales, REVISTA INSTITUCIONAL DE LA FACULTAD DE INFORMÁTICA | UNLP, Mario Piattini
- INGENIERÍA DE SOFTWARE, 9 EDICION, IAM SUMMERVILLE 2011

Fundamentos de Ingeniería de Requisitos

TALLER: Desarrollo de Software: Casos Prácticos con MySQL y PHP

Ing. Francisco Santana, MTI
fsantana@uce.edu.do

Resumen de Ingenieria de Requisitos

Objetivos:

Fortalecer los conceptos, fundamentos y metodología de la ingeniería de requisitos y aplicarlos en casos prácticos de la vida real.

Contenido de la Unidad

- Ingeniería de Requisitos
- Importancia de la Ingeniería de Requisitos
- Porque la Ingeniería de Requisitos
- Famosa ilustracion de la Importancia IR
- Factores de fracaso de proyectos de Software
- Ingeniería de Requisitos (primer paso)
- Proceso consensuado de la Ingeniería de Requisitos.
- Técnicas de Educción de Requisitos.
- Métodos débiles del Analisis

Ingenieria de Requisitos

Que es la Ingenieria de Requisitos?

- La Ingeniería de Requerimientos (IR) es definida como: "La disciplina para desarrollar una especificación completa, consistente y no ambigua, la cual servirá como base para acuerdos comunes entre todas las partes involucradas y en donde se describen las funciones que realizará el sistema"'.

- "Un enfoque sistémico para recolectar, organizar y documentar los requerimientos del sistema; es también el proceso que establece y mantiene acuerdos sobre los cambios de requerimientos, entre los clientes y el equipo del proyecto"

El porque de la Ingenieria de Requisitos

- La Ingeniería de Requerimientos (IR) cumple un papel primordial en el proceso de producción de software, ya que se enfoca un área fundamental: la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, las necesidades de los usuarios o clientes; de esta manera, se pretende minimizar los problemas relacionados por la mala gestión de los requerimientos en el desarrollo de sistemas.

- Estudios realizados muestran que más del 53 % de los proyectos de software fracasan por no realizarse un estudio previo de requisitos. Otros factores como la falta de participación del usuario, los requerimientos incompletos y el cambio a los requerimientos, ocupan sitiales altos en los motivos de fracasos.

Importancia de la Ingenieria de Requisitos

Según la autora Lizka Johany Herrera en su documento de la ingeniería de requerimientos, los principales beneficios que se obtienen de la Ingeniería de Requerimientos son (2003: 3):

- Permite gestionar las necesidades del proyecto en forma estructurada: Cada actividad de la IR consiste de una serie de pasos organizados y bien definidos.

- Mejora la capacidad de predecir cronogramas de proyectos, así como sus resultados: La IR proporciona un punto de partida para controles subsecuentes y actividades de mantenimiento, tales como estimación de costos, tiempo y recursos necesarios.

Importancia de la Ingenieria de Requisitos cont..

- Disminuye los costos y retrasos del proyecto: es sabido que reparar errores por un mal desarrollo no descubierto a tiempo, es sumamente caro; especialmente aquellas decisiones tomadas durante la IR, ya que es una de las etapas de mayor importancia en el ciclo de desarrollo de software y de las primeras en llevarse a cabo.

- Mejora la calidad del software: La calidad en el software tiene que ver con cumplir un conjunto de requerimientos (funcionalidad, facilidad de uso, confiabilidad, desempeño, etc.).

Importancia de la Ingenieria de Requisitos cont..

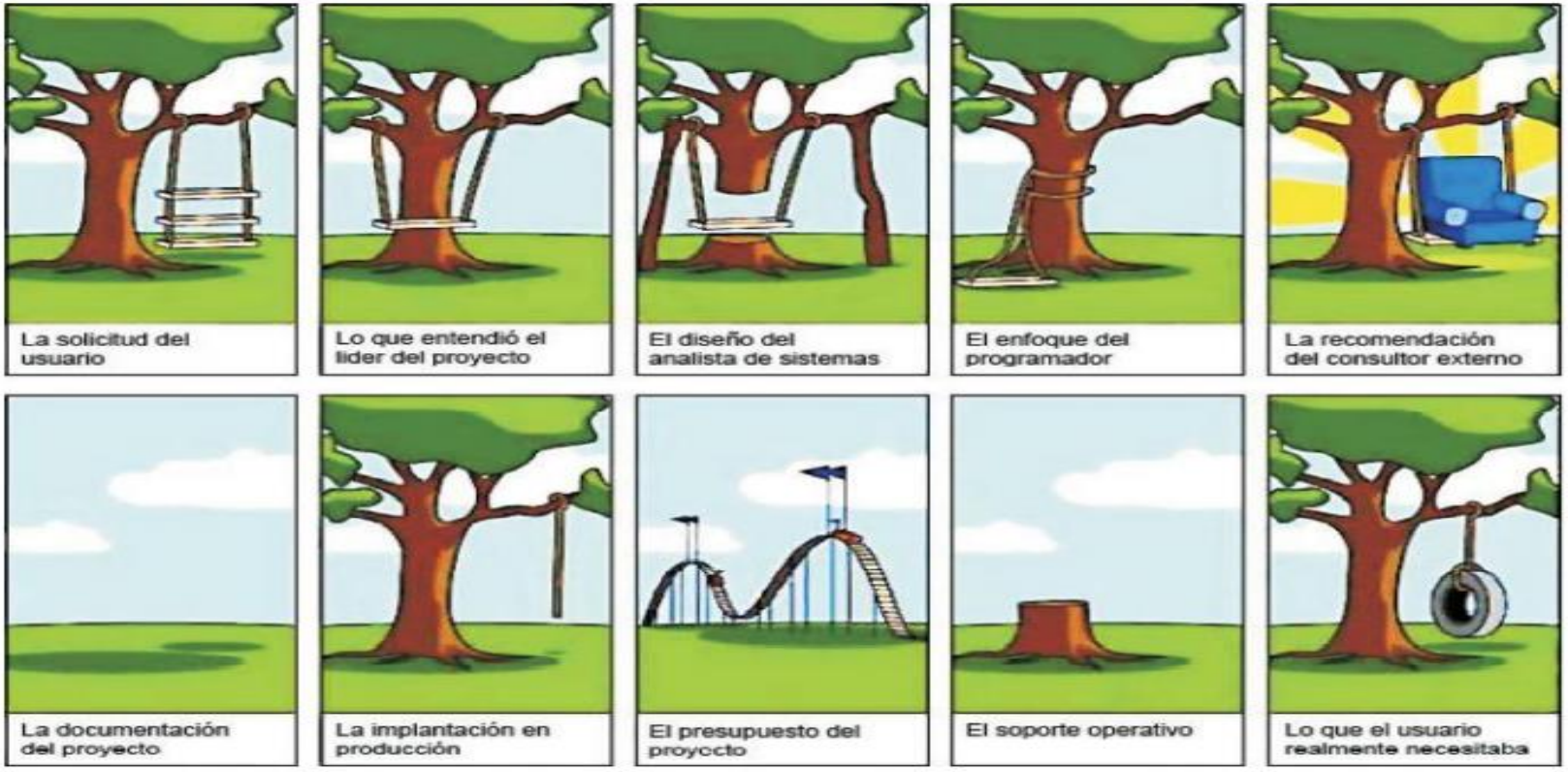
- Mejora la comunicación entre equipos: La especificación de requerimientos representa una forma de consenso entre clientes y desarrolladores. Si este consenso no ocurre, el proyecto no será exitoso.

- Evita rechazos de usuarios finales: La ingeniería de requerimientos obliga al cliente a considerar sus requerimientos cuidadosamente y revisarlos dentro del marco del problema, por lo que se le involucra durante todo el desarrollo del proyecto.

Importancia de la Ingenieria de Requisitos cont..

- Típicos casos de la vida real. Analicemos la imagen a la derecha y compartamos que entendemos.

Famosa Ilustracion de Importancia de la IR



Factores de Fracaso de los Proyectos

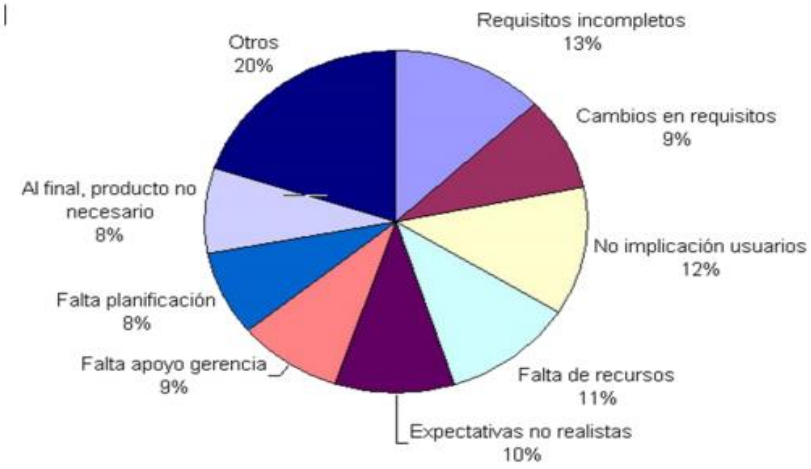
Los principales factores de fracaso de los proyectos software

Factores de Daño o cancelación	%
Requerimientos incompletos	13.1
Deficiencia en el involucramiento del usuario	12.4
Deficiencia de recursos	10.6
Expectativas no realistas	9.9
Deficiencia en soporte ejecutivo	9.3
Cambios en los requerimientos y especificaciones	8.7
Deficiencia en la planeación	8.1
Ya no se necesita más	7.5
Deficiencia en administración de TI	6.2
Desconocimiento en tecnología	4.3
Otros	9.9

Fig. 1. Factores de falla o cancelación en los proyectos [5]

Factores de Fracaso de los Proyectos

Otro estudio nos muestra, estos factores de fracaso en proyectos en base a la mala especificación de requisitos.



- Requisitos incompletos: 13%
- Cambios en requisitos: 9%
- No implicación de usuarios: 12%
- Expectativas no realistas: 10%
- Producto no necesario: 8%

- **TOTAL: 52%**

Factores de Fracaso de los Proyectos

- **Estimación:** No es posible estimar con rigor los costes y recursos necesarios para desarrollar algo que no se conoce.
- **Planificacion:** No se puede confiar en la planificacion para desarrollo de algo que no se sabe bien como es.
- **Diseno**: Los errores en los requisitos, las modificaciones frecuentes las deficiencias en restricciones o futuras evoluciones, producen arquitecturas que mas tarde se confirmaran con erroneas y seran modificados.
- **Construccion:** La deficiencias en los requisitos obligan a programar un ciclo de prueba error que derrocha horas y paciencia de programacion sobe patrones de recodificacion continua y “programacion heroica”.
- **Validacion y Verificacion:** Terminado el desarrollo de sistemas, si las especificacion tienen errores de bulto, o por aun, no estan reflejadas en una especificacion de requisitos, no sera posble validar

Beneficios del Levantamiento de Requisitos

Acuerdo entre desarrolladores, clientes y usuarios sobre el trabajo que debe realizarse. Unos requisitos bien elaborados y validados con el cliente evitan descubrir al terminar el proyecto que el sistema no era lo que se pedía.

 Acuerdo entre desarrolladores, clientes y usuarios sobre los criterios que se emplearán para su validación. Resulta muy difícil demostrar al cliente que el producto desarrollado hace lo que el pidió si su petición no está documentada y validada por él.

 Base objetiva para la estimación de recursos (coste, personal en número y competencias, equipos y tiempo) Si los requisitos no comprenden necesidades reales, las estimaciones no dejan de ser meras apuestas. Las estimaciones en el fondo son cálculos de probabilidad que siempre implican un margen de error; por esta razón disponer de la mayor información posible reduce el error.

Beneficios del Levantamiento de Requisitos

 Concreción de los atributos de calidad (ergonomía, mantenibilidad, etc.) Más allá de funcionalidades precisas, los requisitos recogen atributos de calidad necesarios que en ocasiones son desconocidos por los desarrolladores, produciendo finalmente sistemas sobredimensionados o con serias deficiencias de rendimiento.

 Eficiencia en el consumo de recursos: reducción de la re-codificación, reducción de omisiones y malentendidos. Tener un conocimiento preciso de lo que hay que hacer evita la prueba y error, repetición de partes ya desarrolladas, etc.

Documentos de la IR

LIMITACIONES DE LA ING. DE REQUISITOS

Existe un limite fundamental, y sobre todo inherente a la construcción de software, que afecta a la Ingeniería de Requerimientos. Este limite puede expresarse de la siguiente forma: es virtualmente imposible iniciar la construcción de un sistema de software teniendo la lista completa y consistente de todos los requerimientos, en un tiempo razonable.

Entre las razones que justifican esta hipótesis tenemos las siguientes:

- Usualmente se requiere que los grandes sistemas de software constituyan una superación del estado del arte en cierto dominio de aplicación.
- Los grandes sistemas normalmente son utilizados por una comunidad de usuarios muy diversos. Esto implica que cada grupo de usuarios planteara expectativas y requerimientos diferentes sobre el sistema.
- En general la gente que paga por el sistema y quienes lo usan no son los mismos, lo que genera conflictos entre lo que se puede pagar y lo que se necesita.

LIMITACIONES DE LA ING. DE REQUISITOS

- La introducción de un sistema de software importante en una organización es tan disruptiva que suele producir cambios igualmente importantes en la organización lo que, a su vez, genera nuevos requerimientos sobre el sistema.

- El entorno en el cual se instala y opera el sistema cambia por lo que el sistema debe cambiar también.

- El negocio de la empresa productora del software suele cambiar por lo que sus ejecutivos solicitan nuevas funciones o cualidades al sistema.

Proceso de Ingeniería de Requisitos Consensuado

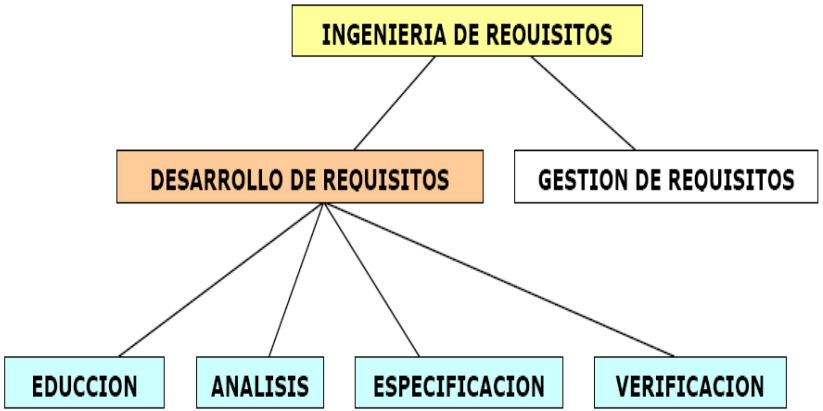


Figura 7. Ingeniería de Requisitos [SWEBOK 2001]

Educción: Se refiere al proceso de Captura y Descubrimiento de los Requisitos.

Análisis: Consiste en detectar y ver conflictos entre los requisitos

Especificación: Plasma en un documento formal las características deseadas:
STD 830 IEEE

Verificación: Pretende descubrir problemas en el documento de especificación de requisitos antes de la Implementación.

Gestión: Consiste en gestionar los cambios de los requisitos.

Cuantos tipos de Requisitos de usuario hay?

- Funcionales: Se refieren a tares o funciones primordiales que se espera realize el sistema.
  - *El sistema permitira aceptar pagos con tarjeta de credito y de débito.*
  - *El sistema gerenera dos tipos de facturas a credito para cobro futuro y al contado para pago directo seleccionadas por el usuario.*

- No funcionales: Se refieren a los atributos de calidad o cualidades del sistema (restricciones de seguridad, acceso, control etc) que el sistema debe cumplir.
  - *El sistema solicitara el nombre de usuario y contraseña para validar el acceso al sistema.*
  - *El sistema permitira llevar un historial de acceso de los usuarios al sistema para fines de auditoria.*
  - *El sistema sera multiplataforma o sea se ejecutara en diferentes sistemas operativos.*

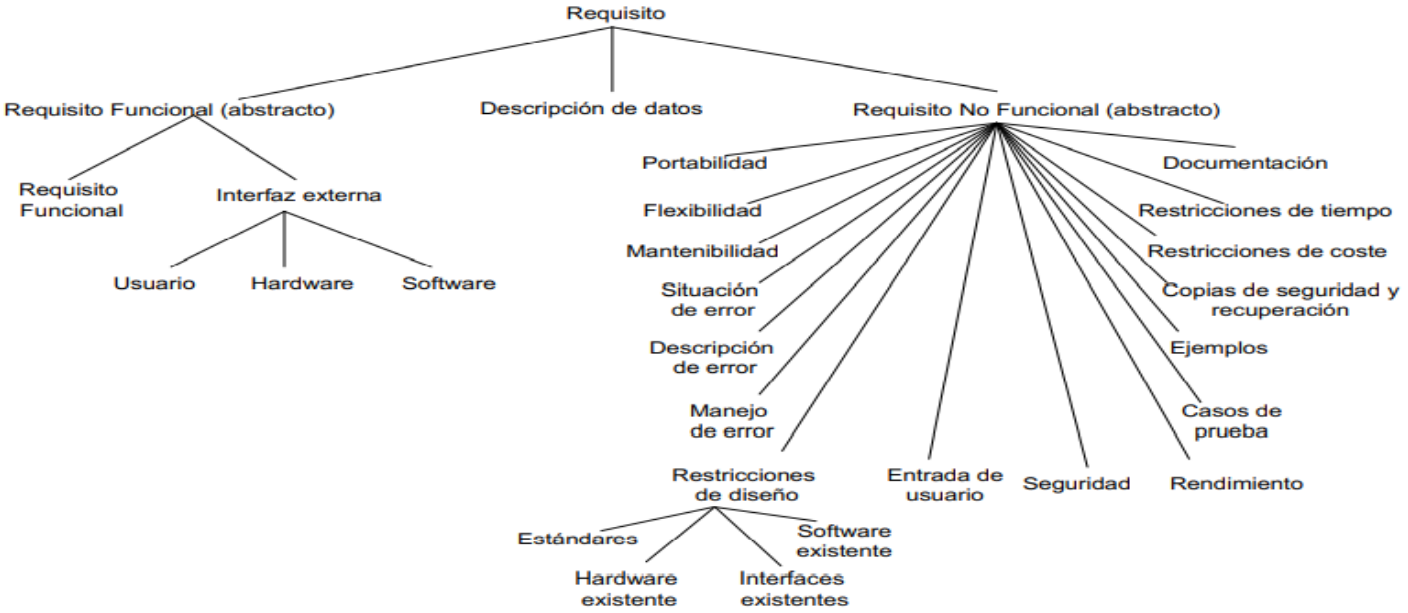
Elicitación de los Requisitos

La **Elicitación de Requisitos –ER–** es la piedra angular en el desarrollo de proyectos software y tiene un impacto muy alto en el diseño y en las demás fases del ciclo de vida del producto.

Si se realiza apropiadamente, puede ayudar a reducir los cambios y las correcciones en los requisitos. Además, la calidad de la elicitación determina la exactitud de la retroalimentación al cliente acerca de la integridad y validez de los requisitos.

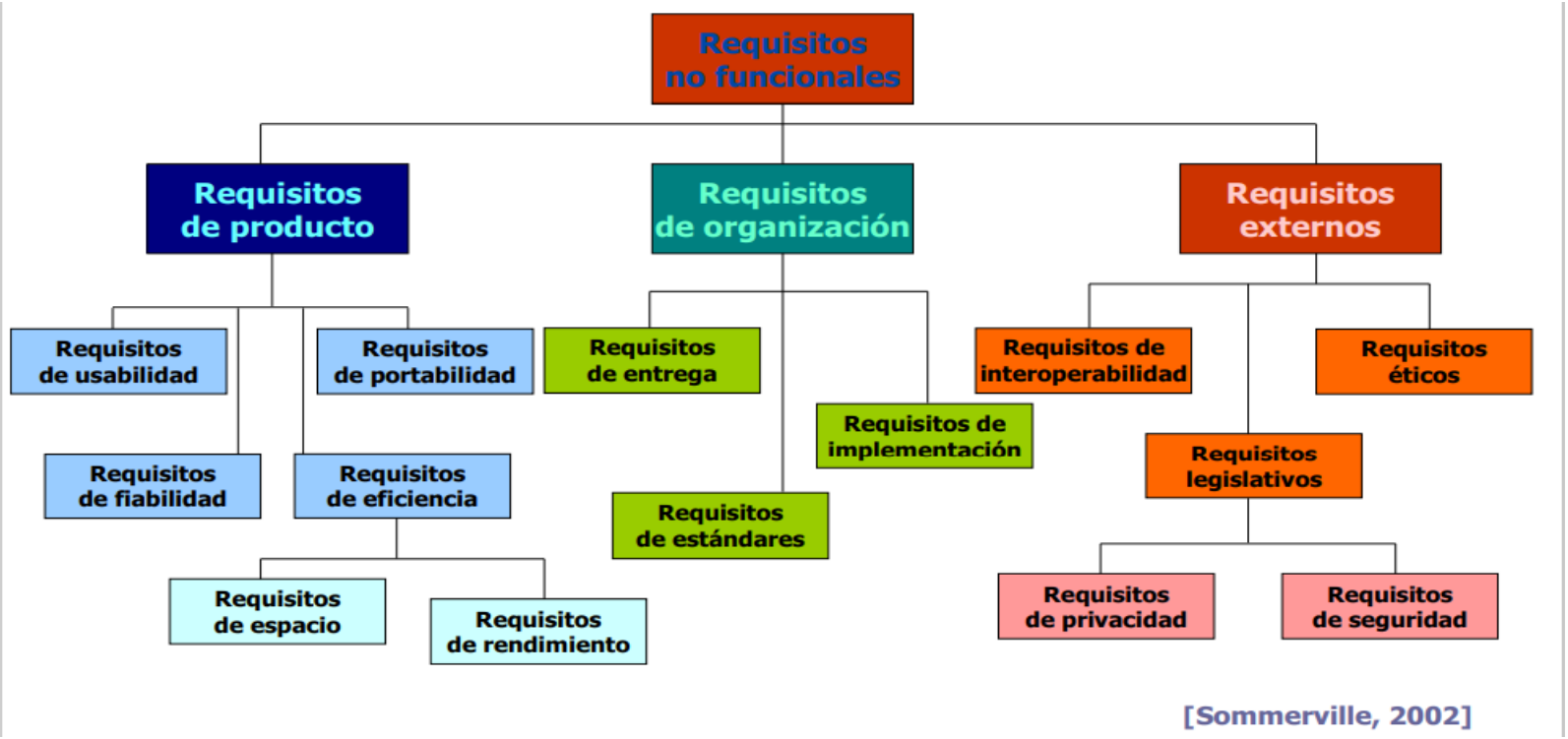
Debido a que esta fase es crítica y de alto impacto en el proyecto, es muy importante que la labor de elicitar se realice lo más cercano posible a la “perfección”.

Dimensiones de los Requisitos



Jerarquía de especialización de RSM – adaptado de [Pohl, 1997]

Requisitos no funcionales



Elicitacion de los Requisitos

- Educción de Requerimientos
- Redacción de Requerimientos
- Técnicas de Educción de Requisitos

Gestión de los Requisitos

- Son los fundamentos funcionales que sustentan la construcción de un sistema software.

- Es una especificación formal de que hara el sistema luego que el sistema este creado.

  El sistema debera …

  El sistema Controlara…

  El sistema generara …

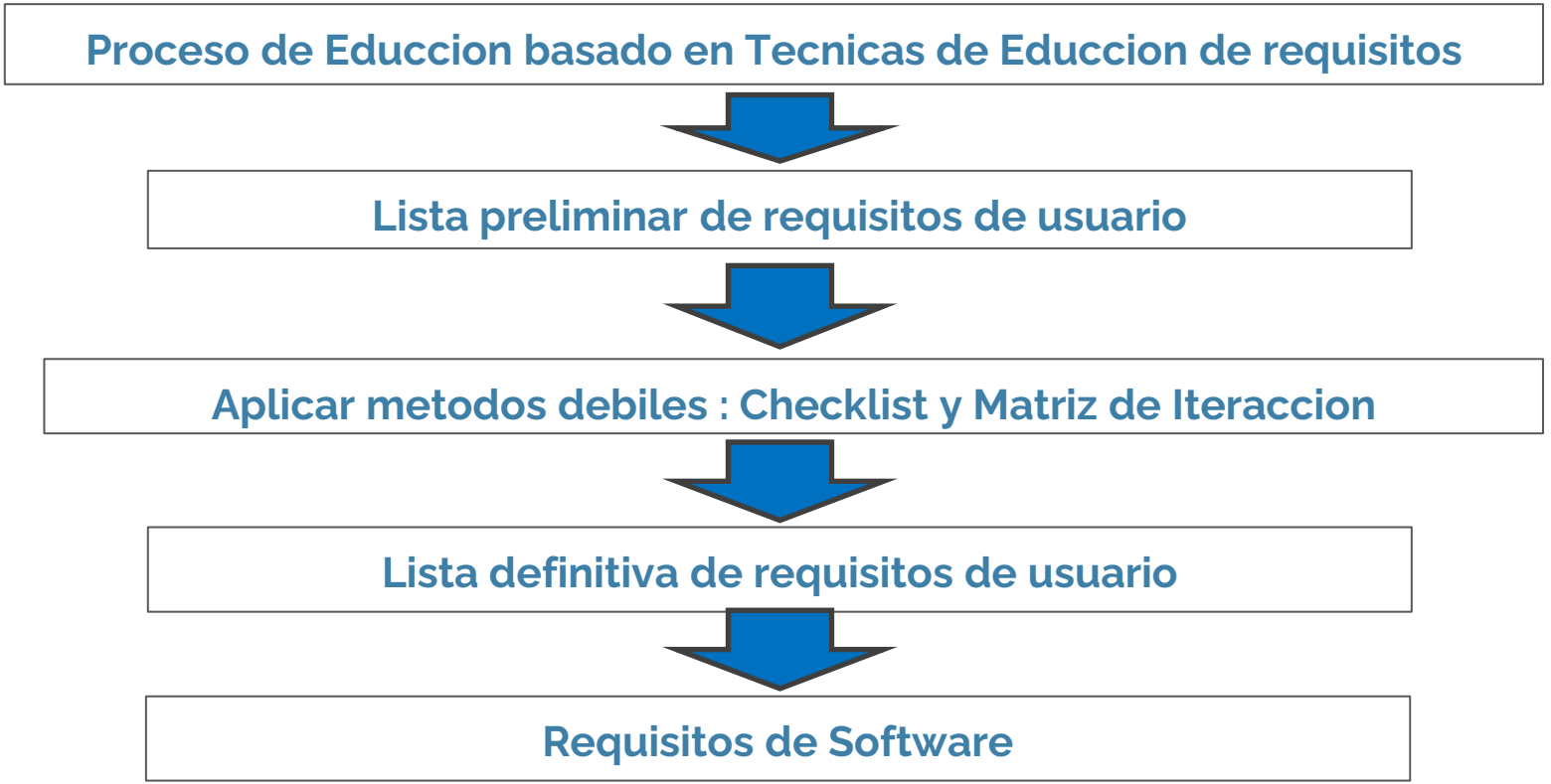
Gestion de los Requisitos

Usuario:

RU- 1 El sistema calculara el salario de los empleados de la empresa (Funcional)

- 1.1 Integrar los incentivos
- 1.2 Calcula sueldo bruto (HT* PH)+Incentivos
- 1.3 Calculo la retencion de TSS
- 1.4 Calculo la Retencion de ISR
- 1.5 Agrego otros descuestos
- 1.6 Calculo el SN basado SB menos deducciones

Lista preliminar de Requisitos



Lista preliminar de Requisitos

1. El Sistema procesara…
2. El Sistema Controlara…
3. El sistema Registara …
4. El sistema solicitara usuario y contrasena para el control de acceso
5. …
6. N

Requisitos de Software

RU-1- El sistema registrara los datos generales de los estudiantes (Entrevista 1,1)

1.1 Registrar datos generales estudiantes
1.2 Consultar datos generales estudiantes
1.3 Modificar datos generales estudiantes
1.4 Reportar Datos generales estudiantes

Objetivos vs Requisitos

Objetivos	Requisitos usuarios
Monitorear los avances de las enfermedades perinatales en niños menores de 12 años	1. El sistema registrara los datos generales de los ninos menores de 12 anos (Entrevista 1,1) 2. El sistema registrara las enfermedades que padecen lo ninos menores de 12 anos 3. El sistema registrara los sectores y municipios donde se localizan los mayor cantidad de ninos enfermos
Monitorear los avances de las enfermedades perinatales en niños menores de 12 años	1. Req Usuario 1 2. Req Usuario 2

Validacion de un Proyecto de Software

1	2	3	4	5	6	7
Tema del Proyecto	Objetivo General Alcance Descripcion	1. Ob. Esp 2. Ob. Esp 3. Ob. Esp 4. … 5. Ob. Esp N	Obj Esp 1	1. Req Usuario 1 2. Req Usuario 2 3. Req Usuario 3 4. .. 5. Req Usuario N	Req Usuario 1	1. Req Soft 1 2. Req Soft 2 3. … 4. Req Soft N

Tecnicas de Educcion de Requisitos

Técnicas de Educción de Requisitos

Tecnicas de Educcion de Requisitos

- Cuando los investigadores tratan el tema de educción, se refieren a una gran cantidad de métodos o TÉCNICAS de representación o modelación de requisitos [Kovitz 1998] pero no a la forma como se obtienen esos requisitos.

- A pesar de que existe una gran cantidad de técnicas de educción [Goguen and Linde 1993], en muchos casos, se utiliza simplemente las entrevistas para la captura de información.

- Sin embargo, existe evidencia que las entrevistas tradicionales y otras técnicas de inspección son inadecuadas para el desarrollo de sistemas actuales [Beyer and Holtzblatt 1995].

Técnicas de Educción de Requisitos

Algunas de las tecnicas de Educcion de Requisitos mas usuadas son:

- Entrevistas Estructuradas
- Entrevistas No Estructuradas
- Cuestinarios
- Observaciones de Tareas Habituales
- Documentación y Procedimientos
- Brain Storming (lluvia de Ideas)
- Analisis de Tareas
- Veinte Preguntas
- Entrevistas de Grupo
- Metodo Delphi
- Prototipado

Tecnicas de Educcion de Requisitos

Entrevistas

- La entrevista estructurada se caracteriza porque se realiza a partir de un cuestionario previamente elaborado que se aplica de forma sistemática. Tiene la ventaja de disminuir los sesgos del entrevistador

- Entrevistas Estructuradas
- Entrevistas No Estructuradas
- Cuestinarios

Tipos de Preguntas

- Abiertas
- Cerradas

Preguntas Iniciales de una entrevista

1. ¿Qué objetivos persigue con el desarrollo del <Sistema>?
2. ¿En que tipos de tareas/actividades de la organización debe ayudar el <Sistema>?
3. ¿Qué beneficios espera obtener <Sistema>?
4. ¿Qué características desea que posea/tareas debe realizar el nuevo <Sistema> ?
5. ¿Qué usuarios deberán utilizar el nuevo?
6. ¿Qué individuos, además de los usuarios, se verán afectados por la implantación del <Sistema>?
7. ¿Debe el nuevo interactuar con otros sistemas software preexistentes/que se implementarán en el futuro <Sistema>?
8. ¿Debe el nuevo interactuar con otros sistemas hardware. Bases de datos, etc. preexistentes/que se implementarán en el futuro <Sistema>?
9. ¿Existe alguna restricción temporal para el desarrollo del <Sistema>?
10. ¿Existe alguna restricción (estándar, regulación, etc.) que afecte al desarrollo del <Sistema>?

Preguntas independientes del contexto. Este tipo de preguntas son adecuadas para los niveles organizativos superiores.

Preguntas Iniciales de una entrevista

1. ¿Conoce los planes de la dirección para desarrollar un sistema que realice ?
2. Estamos estudiando el modo de desarrollar el sistema, y necesitamos de su colaboración. ¿Podría ayudarnos?
3. ¿Cómo cree que podría ayudarle el nuevo sistema en sus tareas?
4. Una vez identificadas alguna tarea, preguntar:
   - ¿Qué recibe Ud. para realizar la ?
   - ¿De dónde?
   - ¿Qué genera Ud. al finalizar la ?
   - ¿A quién se lo envía?
   - ¿Archiva Ud. algo?
   - ¿Interviene alguien más en la , aunque sea esporádicamente?

4. ¿Cree Ud. que es necesario hablar con otra persona para averiguar más cosas acerca de ?

Preguntas independientes del contexto. Este tipo de preguntas son adecuadas para los niveles organizativos inferiores.

Redacción de los Requisitos

Los requisitos son la base de la comunicación entre los distintos participantes de la actividad de requisitos. Por ello, es fundamental que los requisitos estén expresados, de la forma lo más precisa posible, en algún tipo de lenguaje entendido por todas las partes. A priori, cualquier tipo de formalismo seria valido:

1. Lenguajes formales, tales como el Z o VDM
2. Lenguajes semi-formales y modelos gráficos, tales como los DFD, casos de uso,etc.
3. Lenguaje natural

No obstante, y dado que los principales participantes del proceso de requisitos son el analista y los clientes/usuarios, el formalismo de expresión más frecuente para los requisitos es el lenguaje natural, dada la facilidad que supone para clientes/usuarios su utilización. Aunque el lenguaje natural sea el formalismo de expresión más utilizado para los requisitos, es necesario indicar que los requisitos no se pueden expresar de cualquier forma. Existen una serie de recomendaciones para redactar los requisitos del software:

Redaccion de los requisitos

La forma más usual de redactar los requisitos del software es utilizar frases cortas, las cuales típicamente comienzan por “El sistema permitirá…” o alguna variante.

1. El sistema permitirá registrar los datos identificativos y contables de los clientes

2. El sistema deberá generar un listado de facturas impagadas por un periodo mayor de 15 días. El listado deberá imprimirse los lunes a las 7:00

3. La consulta de un artículo en stock debe proporcionar los datos requeridos en un tiempo menor que 0.5 segundos

4. El sistema deberá funcionar en entornos windows (todas versiones posteriores a 1999) y Unix System V

5. El acceso a los datos personales deberá cumplir las normas especificadas en la LOPD

Metodos de Validacion de Requisitos

Metodos de Validación de Requisitos

Metodos Debiles del Analisis

Existen, principalmente, dos métodos débiles de análisis: el checklist de análisis y la matriz de interacción. La denominación de métodos débiles surge del hecho de que, al contrario que los modelos conceptuales, aquellos son válidos en todo tipo de sistema software y configuración de proyecto.

Esto es, los métodos débiles pueden usarse siempre, mientras que los modelos conceptuales, dependiendo del tipo de sistema, pueden ser más o menos útiles.

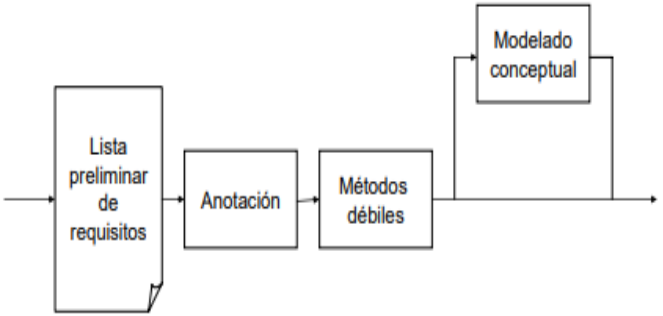


Figura 1. Proceso de análisis

Ejercicio de Educcion de Requisitos

Ejercicio práctico: 10 Minutos:

Se Requiere:

Se desea obtener la lista de requerimientos para el desarrollo de una sistema de Mensajería como Whatsapp.

Checklist

Un checklist de análisis es, simplemente, *un conjunto de preguntas* que el analista debe considerar *para cada requisito individual*. Estas preguntas están relacionadas con alguno de los siguientes atributos de calidad:

- Independencia del diseño
- Concisión (Interna)
- Realizabilidad
- Externamente consistente
- Ambigüedad
- Verificabilidad.

Matriz de Iteracion Ejem:

Atributo de calidad a considerar	Pregunta
Independencia del diseño	• ¿La lista de requisitos anticipa el diseño o incluye información de implementación?
Concisión	• ¿Cada requisito es simple o, por el contrario, podría dividirse en varios requisitos? • ¿Existe algún requisito que no parezca añadir ninguna información útil acerca del sistema a desarrollar?
Realizabilidad	• ¿Es realizable el requisito en la plataforma de implementación? • ¿Existe algún requisito irrealizable con la tecnología actual? **NOTA**: Para responder a esta pregunta, es necesario conocer los aspectos técnicos de la plataforma donde se implementará el sistema.
Consistencia externa	• ¿Existe algún requisito que contradiga requisitos organizativos explícitamente formulados?
Ambigüedad	• ¿Es posible interpretar de varias formas un requisito?
Verificabilidad	• ¿Es posible idear algún caso de prueba que permita establecer que el requisito NO SE CUMPLE?

Matriz de Iteraccion

Una matriz de interacción es, simplemente, una matriz de doble entrada donde se *cruzan todos los requisitos entre sí*, tal y como muestra la Tabla 3. Por cruzar, debe entenderse que para cualesquiera dos requisitos r_1 y r_2, se debe comprobar si:

REQUISITOS	1	2	3	4	5	6	7	8	9	10	.	.	N
1													
2													
3													
4			S										
5													
6													
7							C						
8						R							
9									SD				
10													
11						R							
.													
.													
N													

Tabla 3. Matriz de interacción

Checklist

Un checklist de análisis es, simplemente, un conjunto de preguntas que el analista debe considerar para cada requisito individual. Estas preguntas están relacionadas con alguno de los siguientes atributos de calidad:

- Independencia del diseño
- Concisión
- Realizabilidad
- Externamente consistente
- Ambigüedad
- Verificabilidad

El propósito del checklist es asegurar que el analista ha descrito adecuadamente cada requisito de la lista preliminar de requisitos. En caso de que el analista detectase alguna carencia, éste debería corregir el requisito defectuoso de forma inmediata.

Caracteristica de un buen requisito

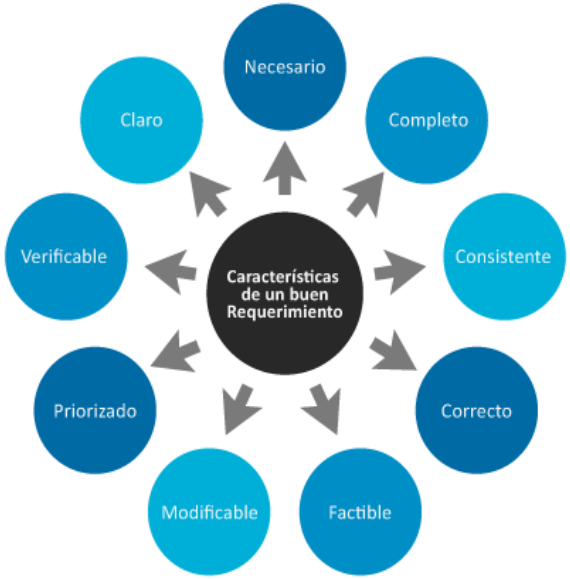
Necesario. Si se tiene alguna duda acerca de la necesidad del requerimiento, se pueden preguntar “¿Qué sería lo peor de no incluirlo?” Si no se encuentra una respuesta o cualquier consecuencia, entonces es probable que no sea un requerimiento necesario.

Completo. Un requerimiento esta completo si no necesita ampliar detalles en su redacción, es decir, si se proporciona la información suficiente para su comprensión.

Consistente. Un requerimiento es consistente si no es contradictorio con otro requerimiento

Correcto. acuerdo entre dos partes. Contiene una sola idea.

Priorizado. Categorizar el requerimiento nos ayuda a saber el grado de necesidad del mismo Esencial/Critico, Deseado, Opcional Verificable



Caracteristica de un buen requisito

Factible. El requerimiento deberá de ser totalmente factible y dentro de presupuesto, calendario y otras restricciones, si se tiene alguna duda de su factibilidad, hay que investigar, generar pruebas de concepto para saber su complejidad y factibilidad, si aun así el requerimiento es no factible hay que revisar la visión del sistema y replantear el requerimiento

Verificable. Si un requerimiento no se puede comprobar, entonces ¿Cómo se sabe si se cumplió con él o no? Debe ser posible verificarlo ya sea por inspección, análisis de prueba o demostración. Cuando se escriba un requerimiento, se deberá de determinar los criterios de aceptación.

Rastreable. la especificación se debe organizar de tal forma que cada función del sistema se pueda rastrear hasta su conjunto de requerimientos correspondiente. Facilita las pruebas y la validación del diseño

Claro. Un requerimiento es conciso si es fácil de leer y entender, su redacción debe ser simple y clara para aquellos que vayan a consultarlo en un futuro

Matriz de Iteraccion

Una matriz de interacción es, simplemente, una matriz de doble entrada donde se cruzan todos los requisitos entre sí, tal y como muestra la Tabla 3. Por cruzar, debe entenderse que para cualesquiera dos requisitos r1 y r2, se debe comprobar si:

- r1 se solapa con r2, esto es, r1 trata aspectos del sistema también tratados en r2. Ello, de ser cierto, daría lugar a problemas de redundancia. Esto es lo que se ha indicado con una S en la Tabla 3.

- r1 está en conflicto con r2, esto es, r1 y r2 son contradictorios. Ello, lógicamente, da lugar a problemas de consistencia interna. Esto es lo que se ha indicado con una C en la Tabla 3.

	r_1	r_2	...	r_n
r_1		C		
r_2				S
...				
r_n				

Tabla 3. Matriz de interacción

Matriz de Iteracion Ejem:

	Matriz de Iteracion									
Requisitos	1	2	3	4	5	6	7	8	9	10
1										
2				S						
3			AMB							
4										
5					ED					
6			S							
7										
8								IRR		
9								S		
10										

Problemas en la aplicación: Matriz de Iteracion

Existen tres problemas principales en la aplicación del método de la matriz de interacción.

La imposibilidad del analista de verificar sus propias creaciones. La solución, en este caso, sería la misma que la indicada anteriormente (permitir que la matriz de interacción fuese aplicada por otra persona y que ésta detectase los defectos en los requisitos y los plasmase en un alista de errores y acciones recomendadas).

Cómo manejar grandes conjuntos de requisitos. La única solución a este problema es descomponer el sistema software en varios subsistemas, con la finalidad de reducir la complejidad del análisis de cada uno de ellos.

Problemas en la aplicación: Matriz de Iteracion

Cómo detectar conflictos y solapamientos. Esto es, dados dos requisitos r1 y r2, ¿cómo puede saber el analista, o la persona que esté aplicando la matriz de interacción, si ambos requisitos son conflictivos o están solapados? Esta pregunta es, no obstante, de fácil respuesta. Para poder aplicar la matriz de interacción, el analista, o la persona que lo substituye, debería poseer un buen conocimiento de la aplicación a construir. Sólo de este modo es posible no cometer errores, por omisión o comisión, a este respecto.

Finalmente, si poseyendo un buen conocimiento de la aplicación, el analista, o la persona que lo substituye, no puede juzgar si dos requisitos r1 y r2 son conflictivos o están solapados3 , lo preferible es considerar que existe un problema potencial entre los requisitos r1 y r2 y estudiar éstos más detenidamente.

Informe de Anomalia

Esto es, la persona que aplica el checklist tiene la responsabilidad de identificar errores y comunicárselos al analista, pero no la de corregirlos por sí mismo. Muy al contrario, esta es responsabilidad del analista, dado que es el que mejor conoce el sistema software a desarrollar. Los errores y las sugerencias de solución acostumbran a plasmarse en un documento como el mostrado en la Tabla 2, denominado Lista de errores y acciones recomendadas.

Nº de requisito	Defectos detectados	Acciones recomendadas
1	Error de estilo, que lleva a ambigüedad	Modificar el texto del requisito de tal forma que diga algo como "El sistema deberá permitir el registro de los fondos bibliográficos"
2	Idem	Idem
11	Ambigüedad	Precisar la duración de las reservas
12	Concisión	No se han identificado diferencias entre profesores y alumnos a todo lo largo de la lista de requisitos. Este requisito se debería eliminar, ya que proporciona ningún tipo de información relevante.
15	Realizabilidad	El sistema no puede realizar automáticamente los préstamos. Quizás se refiere el requisito a que debe proporcionar el máximo de automatización? Precisar, en este caso, o eliminar por irreal.
17	Concisión	Separar lo referido a libros prestados de lo referido a libros reservados.

Tabla 2. Lista de errores y acciones recomendadas

SOCIALIZACION – PREGUNTAS?

Preguntas Y/O Comentario?

Fundamentos de Base de Datos

TALLER: Desarrollo de Software: Casos Prácticos con Mysql y PHP

UNIVERSIDAD CATOLICA DEL CIBAO (UCATECI)
Micro Unidad I – FUNDAMENTOS DE LA INGENIERIA DE SOFTWARE

Ing. Francisco Santana, MTI
fsantana@uce.edu.do

Fundamentos de Base de Datos

Objetivos:

Comprender e implementar los conceptos básicos de base de datos para lograr un modelo de base de datos normalizada.

Contenido de la Unidad

- Conceptualización
- Metodología de Base de Datos
- Modelos de Datos
- Modelo Relacional
- Planilla de la Entidad
- Atributos y Tipos de Atributos
- Resumen

Metodologia de Base de Datos

- El diseño de una base de datos es un proceso complejo que abarca decisiones a muy distintos niveles.

- La complejidad se controla mejor si se descompone el problema en subproblemas y se resuelve cada uno de estos subproblemas independientemente, utilizando técnicas específicas.

- El diseño de una base de datos se descompone en diseño conceptual, diseño lógico y diseño físico.

Metodologia de Base de Datos

El diseño conceptual parte de las especificaciones de requisitos de usuarios y su resultado es el esquema conceptual de la base de datos. Un ***esquema conceptual es*** una descripción de alto nivel de la estructura de la base de datos, independientemente del SGBD que se vaya a utilizar para manipularla.

Un ***modelo conceptual*** *es un* lenguaje que se utiliza para describir esquemas conceptuales. El objetivo del diseño conceptual es describir el contenido de información de la base de datos y no las estructuras de almacenamiento que se necesitarán para manejar esta información.

Modelo de Datos

Es una colección de herramientas conceptuales para describir datos, relaciones entre ellos, semántica asociada a los datos y restricciones de consistencia.

Los diversos modelos de datos que se han propuesto se dividen en tres grupos:

- Modelos lógicos basados en objetos
- Modelos lógicos basados en registros
- Modelos físicos de datos.

Modelo de Datos

- Modelos lógicos basados en registros

  Los modelos lógicos basados en registros se utilizan para describir datos en los modelos conceptual y físico.

A diferencia de los modelos de datos basados en objetos, se usan para especificar la estructura lógica global de la base de datos y para proporcionar una descripción a nivel más alto de la implementación.

Los modelos de datos más ampliamente aceptados son:

- El modelo Relacional
- El modelo de Red
- El modelo jerárquico

Modelo Relacional

El **modelo relacional** será nuestra base de estudio en la materia y los demás modelos los veremos en capítulos mas adelante.

Las bases de datos relacionales son el tipo de bases de datos actualmente más difundido. Los motivos de este éxito son fundamentalmente dos:

1. ofrecen sistemas simples y eficaces para representar y manipular los datos

2. se basan en un modelo, el relacional, con sólidas bases teóricas

El modelo relacional fue propuesto originariamente por E.F. Codd en un ya famoso artículo de 1970. Gracias a su coherencia y facilidad de uso, el modelo se ha convertido en los años 80 en el más usado para la producción de DBMS.

Modelo Relacional

El principal concepto del modelo ER es la **entidad. Una entidad es:**

> Una "cosa" en el mundo real con existencia independiente. Una entidad puede ser un objeto físico (una persona, un auto, una casa o un empleado) o un objeto conceptual (una compañía, un puesto de trabajo o un curso universitario).

Regulares: aquellas que existen por sí mismas y que la existencia de un ejemplar en la entidad no depende de la existencia de otros ejemplares en otra entidad. Por ejemplo "EMPLEADO", "PROFESOR".

Débiles: son aquellas entidades en las que se hace necesaria la existencia de ejemplares de otras entidades distintas para que puedan existir ejemplares en esta entidad.

Ejemplo del modelo relacional:

Empleado

Codigo	Nombre	Dept	Telefono
0001	Pedro Santana	55	555-5555
0002	Manuel Montero	44	222-2222
0003	Jose Paulino	33	111-1111

Departamento

Dept	Nombre	Fecha C.
44	Administracion	01/01/2001
55	Computos	01/01/2002
33	Contabilidad	01/01/2001

Planilla Especificacion de Entidades

Nombre	PROFESOR
Objeto	Almacenar la información relativa de los profesores de la organización.
Alcance	Se entiende como profesor a aquella persona que, contratada por la organización, imparte, al menos, un curso dentro de la misma.
Número de ejemplares	10 profesores
Crecimiento previsto	2 profesores / año
Confidencialidad	1. Nombre y apellidos: Acceso público. 2. Datos personales: Acceso restringido a secretaría y dirección. 3. Salario: Acceso restringido a dirección.
Derechos de Acceso	Para garantizar la total confidencialidad de esta entidad, el sistema de bases de datos deberá solicitar un usuario y una contraseña para visualizar los elementos de la misma.
Observaciones	Los ejemplares dados de baja no serán eliminados de la base de datos; pasarán a tener una marca de eliminado y no serán visualizados desde la aplicación.

Tipos de Atributos

Un Atributo es una propiedad que describe algún aspecto de un objeto. Por ejemplo, una sala de clases tiene un nombre (1-28S), una ubicación, un cupo máximo, etc

Atributos de Valor **Único o Atómicos:** Muchos atributos tienen un único valor para una entidad particular, estos son llamados de valor sencillo o de valor único.

Atributos Multivalor: En otros casos un atributo puede tener un conjunto de valores para una entidad, por ejemplo: un atributo colores para un automóvil o grados universitarios para una persona.

Atributos Derivados: Son aquellos cuyo valor esta determinado por otros atributos. ejemplo: la edad y la fecha de nacimiento de una persona. Para una entidad persona en particular, el valor edad puede ser determinado a partir de la Fecha Actual,

Valor Nulo: (null): En otros casos una entidad puede no tener ningún valor aplicable para un atributo. El valor nulo se utiliza además si desconocemos el valor de un atributo para una entidad particular.

Bibliografia

- Software Requirement, Alan M. Davis
- Effective Prototype, Jonattan Arnolwitz
- Métodos Formales Orientados a Objetos, Francisco Jose Galan, Dept. Lenguajes y Sistemas informáticos. ETSI Informática.
- Requisitos, Master de Ingenieria de Software a Distancia, UPM.
- Tecnicas de Educción de requisitos, Master de Ingenieria de Software a Distancia, UPM.
- Métodos debiles del Análisis, Master de Ingeniería de Software a Distancia, UPM.
- El Paradigma del Modelo Prototipo de la IS, José Luis García Álvarez

SOCIALIZACION – PREGUNTAS?

Preguntas Y/O Comentario?